Brightvine Mortgage Valuation Proposal

Lauren Helbling, Jacob Brown, Kate Stallbaumer

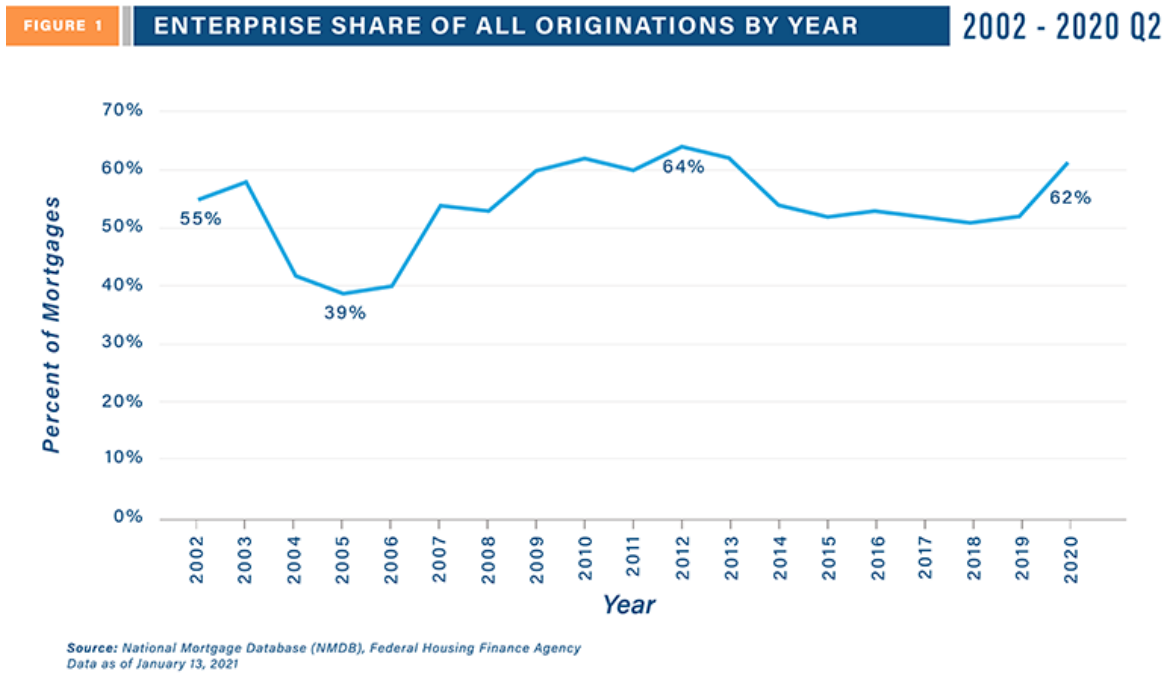
Abstract

The secondary mortgage market is a little known investment opportunity with major benefits. Behind the scenes of a single mortgage, mortgage payments may go between several institutions and investors. When a bank or lending institution has exhausted their capital, third-party investors, often government agencies, offer to purchase mortgages from the lenders so that they can continue lending, and the investors can reap the benefits of the mortgage payments with interest. Brightvine is bringing the secondary mortgage market to the everyday investor by developing a mortgage marketplace through the decentralized power of blockchain. Within this startup, novice and experienced investors alike can purchase and sell mortgages without the tedious transfer of physical paperwork. However, how can an investor quickly recognize which mortgage may be a smart investment, especially as a novice in the industry? Our team was tasked with developing a mortgage valuation tool to assist investors in determining a price point at which a mortgage would be a valuable investment. We evaluated the many factors contributing to the value of an asset, such as details of the home and the interest rate of the lender, in order to predict the ideal buy and sell price points of the mortgage. Upon selecting essential fields and a current mortgage dataset for our training data, we pre-processed the data and iteratively developed a machine learning (ML) solution utilizing clustering and K-Nearest neighbor that interfaces as an API endpoint for usage within the Brightvine platform. Through this technology, investors will be empowered to enter and reap the benefits of the secondary mortgage market on Brightvine. Long term, this technology could be commercialized and sold as a product to financial institutions outside of Brightvine to impact the global secondary mortgage market.

Introduction

Motivation

Hidden behind large corporations and the massive exchange of paperwork, the secondary mortgage market is an undercover but booming business. Within the market, investors and institutions buy and sell mortgages from depositing banks that are seeking more cash to continue lending. Fannie Mae, Freddie Mac, and other government agencies dominate this market, owning upwards of 62% of all mortgages according to the National Mortgage Database [1].



Due to strict underwriting laws, not all mortgages are deemed worthy of being purchased by the federal government. Brightvine is aiming to become another major player in the secondary mortgage market by purchasing and selling mortgages of various levels of risk directly to individual investors. This targets an entirely new market that has been relatively unreached up until now.

Without institutional expertise and financial advising, individual investors may not know which mortgages are worth the risk to invest in and at which price point they should buy or sell. Considering the countless factors involved in a loan, property, and borrower risk, an intelligent model is necessary to predict the value of a mortgage accurately. The development of a machine learning model that offers a predicted price point brings power to individual investors to step into the secondary mortgage market and make intelligent investments. This is a ground-breaking development that could be used globally to transform the world of real estate investments.

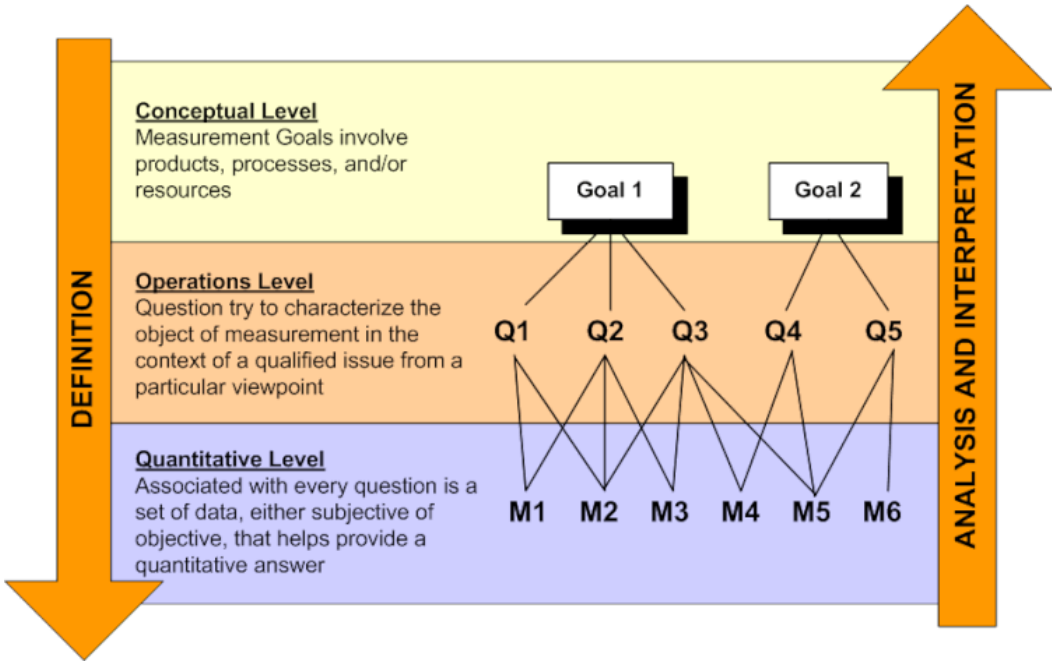
Research Methods

We began our research by following the GQM (goal, question, metric) Model, which specifies three levels of problem definition. On the conceptual level, we defined our goal: ***Investors need a Mortgage Valuation Tool to be featured in the Brightvine Mortgage Desk to assist in determining at which price points a mortgage would be valuable to buy or sell.*** Next, at the operations level, we explored the key questions necessary to reach our end goal, including:

- Which metrics contribute to the value of a mortgage?
- How do we measure the likelihood that a mortgage will foreclose?
- How do we measure the likelihood of a mortgage default?
- How do we predict the future value of a mortgage?
- How do various property types compare in mortgage valuation?

Finally, on the quantitative level, each of these questions is associated with various metrics, which we narrowed down to key metrics below:

- UPB (unpaid principal balance)
- Interest Rate
- Loan Type & Term
- Listing Price
- Property Type
- Property Location & Size
- Occupancy Status
- Borrower Collateral
- Borrower Credit Score
- Economic Conditions
- SOFR (secured overnight financing rate)



After defining these metrics, we will use them to attempt to train different learning models. As we use the data in training, we will compare both the accuracy and time efficiency of the various models in order to choose one to implement for the final product. Some models we will test include the Multi-Layer Perceptron known as a Neural Network with different training methods such as Backpropagation or a Genetic Algorithm, as well as clustering algorithms like K-Means Clustering. During this research, we may also explore dimensionality reduction methods like Forward Selection or Principle Component Analysis which would reduce the datasets to only contain the minimum number of data attributes that still do well to inform the model's prediction.

Hypothesis

The expectation is that the neural network will provide the most accurate predictions for this problem. The process of training and testing using Machine Learning algorithms, Neural Networks, in particular, is discussed further in the background section of this document. For now, it is important to know that the process of creating a prediction given some input will be very quick with a neural network, but the training process can be very time-intensive. The expectation is that training with a genetic algorithm will provide the fastest and most accurate results.

End Product

At the end of our research phase, we will have finalized a selection for an ML model and its training method. We will also have performed tests on the training data to assess the need for each attribute of the dataset. The end product will feature three main aspects. The input processing that the ML model performs, the visualization of this calculation that is sent to the web app, and the training of the model with new data. Two of these are triggered by the user of the Brightvine Mortgage Desk and the third is by an administrator uploading new mortgage data.

When an investor, or another user of the Brightvine Mortgage Desk, opens the details page of a specific mortgage loan, this will trigger our ML model to make a prediction by taking in the input values of the selected mortgage, passing those through the trained model, and producing a prediction. Then, a separate script of code will take the output of the model and transform it back into a value that makes sense in the scope of the investment problem. This script will also send values to an API which the existing Brightvine web app can request data from in order to display the price point suggestion and any corresponding visual representation we design.

When a Brightvine administrator uploads new training data for the model to learn from, this will trigger a re-training of the model. The model will perform its training method using the new data and store new weights and parameters for the newly trained model to use when making new suggestions. We expect this to occur once every week or two to keep up to date with the rapidly changing state of the real estate and secondary mortgage market. In the future, this tool could be improved upon by making this a more automated process to constantly be training the model on newer and more relevant data.

Qualifications

United by our interest in Brightvine, our team of three features a diverse mixture of backgrounds, experiences, and skills.

Lauren Helbling has both research and internship experience in blockchain technology. In the spring of 2022, Lauren joined the first engineering team at Brightvine as a software engineering intern to build the initial products and website for the new startup. Since then, Lauren has found her interests in web development, digital design, and business. With this, Lauren directed the front-end and user-interaction components of the project.

Jacob Brown comes into this project with an interest in data analytics and database procedures. This matches well with the data problem our team faces as we aim to build a predictive model using many numerical features. After completing a Machine Learning course at the end of the 2022 Fall semester, Jacob led the team in choosing and developing a machine learning model for this project.

Kate Stallbaumer utilized her background in full-stack development, communications, operational management, and organizational development to spearhead the data selection and pre-processing. Together, the team is well-equipped to accomplish this project.

Our resumes further demonstrate our various abilities:

Lauren Helbling

208.691.6120 | laurengraceh97@gmail.com | lauren-helbling

EDUCATION

Montana State University Expected Graduation, May 2023
Bachelor of Science in Computer Science Bozeman, MT

- **Details**: 3.99 GPA, Honors College, Minors in Mathematics and Data Science
- **Coursework**: Data Structures and Algorithms, Discrete Mathematics, Web Development, Software Engineering, Design Thinking (UI/UX), Data Mining
- **Awards**: Montana State University Presidential Scholar (2019-2023), NRF Foundation Ray Greenly Award (2022), Montana NCWIT Aspirations in Computing Award (2019), Western Aerospace Scholar (2019)

SKILLS

Languages : Python, Java, JavaScript (React), HTML/CSS
Tools : Git, Docker, Agile, Figma, Slack, Webflow, Adobe Design Programs

EXPERIENCE

Brightvine Fall 2021 - Spring 2022
Software Engineering Intern Remote Work

- Collaborated closely with the founding startup team to design, develop, and launch blockchain-powered FinTech products
- Independently developed marketing site from Figma designs using Webflow
- Maintained a work-from-home role while enrolled full-time in college

Montana State University Summer 2021
Research Experience for Undergraduates (REU) Bozeman, MT

- Participated in rigorous 10-week, full time cybersecurity research program
- Developed optimistic concurrency control in blockchain to parallelize execution and **increase throughput by 500%**
- Utilized Python to develop a blockchain simulation to test algorithm efficiency
- Authored research paper **accepted into IEEE Globecom Conference**

Morrison-Maierle Summer 2019
Summer Intern Helena, MT

- Assembled and maintained database of past projects, blueprints, and important materials for universal access
- Received training and certification from **Helena Summer Jobs Program** in work skills, including communication, teamwork, and conflict resolution

PROJECTS

Happy Scrunchie

- Owner of ecommerce business with **over 900 online sales** on Etsy (*happyscrunchie.etsy.com*)
- Design and implement marketing and sales strategies on multiple platforms
- Build traffic with search engine optimization (SEO) and sales analytics

InterVarsity Digital Design

- Create weekly content for Instagram, website, and print media utilizing Canva and Adobe programs

Jacob Brown

20 Hoffman Dr #4, Bozeman, MT 59715 | (208) 590-7765 | Jacob.brown.23@hotmail.com

Education

MONTANA STATE UNIVERSITY | GRADUATING CLASS OF 2023

- **Major**: Computer Science; **Minor**: Data Analytics; Honors College
- **GPA:** 3.96
- **Relevant courses**: Machine Learning; Database Systems; Web Development; UX/UI Design Thinking; Data Mining – Data Analysis in Python; Statistical Computing/Graphical Analysis – Coding in R; Python, Java and C/C++ courses

UNIVERSITY HIGH SCHOOL | SPOKANE, WA | GRADUATING CLASS OF 2019

- 4.0 GPA – Valedictorian – gave speech to 6,000 in McCarthy Athletic Center

Work Experience

EXPERIENCED PRODUCTION OPERATOR | SYNERGY SPORTS | JANUARY 2022 – AUGUST 2022

- Log data for football plays – tracking event time stamps, player participation, and formation maps
- Test logging software and workflow phases for bugs, inconsistencies, and pain points
- Train new loggers in the logging and workflow processes

HEAD ANALYTICS INTERN | MONTANA STATE FOOTBALL | MARCH 2021 – DECEMBER 2021

- Analyze and display play calls used during practice in order to identify tendencies
- Write Python code for more detailed analysis of offensive play-calling to find tendencies in MSU's offense and in an opponent's play-calling to help provide scouting reports from an analytics perspective

RESIDENT ADVISOR | MONTANA STATE | AUGUST 2020 – MAY 2021

- Foster community among residents on my floor of a residence hall
- Work with team of other RAs to ensure safety of building and plan events for residents

SUMMER INTERN | ADVANCED ELECTRONIC DESIGNS | MAY 2020 – JULY 2020

- Electronics/Electrical/Software Engineering Firm
- Assist in various projects for the company and for individual employees
    - Data display and web scraping using Python coding language
    - Constructing custom acrylic resin desktops for employees

Extracurricular Activities

BOY SCOUTS

- Eagle Scout – highest scout rank, earned for exceptional leadership and service
- Senior Patrol Leader of a troop of 60 boys – an elected position, the leader of the whole troop

CHURCH

- Cru, campus ministry
    - Bible Study leader, Co-Student President

UNIVERSITY HIGH SCHOOL TENNIS

- Varsity Tennis – Four Years
    - 2018 Team MVP – 2019 Team Captain

KATE STALLBAUMER

kate.stallbaumer@gmail.com
linkedin.com/in/katestallbaumer
github.com/kstall23
(406) 594-1392
Bozeman, MT

EDUCATION

Montana State University	BS, Computer Science	Aug. 2019 – May 2023
Benedictine College	BA, Political Science	Aug. 2002 – May 2006

COURSEWORK

Advanced Software Engineering	Database Systems	Systems Administration
C, Python, Java	Discrete Structures	Web Development
Computer Science Theory	Human Computer Interaction	UX/UI Design
Computer Security	Social & Ethical Issues in CS	Web Design
Data Structures & Algorithms	Software Engineering	

EXPERIENCE

FOUNDANT TECHNOLOGIES, *Intern* **Bozeman, MT**
Oct. 2019 – Present

- Utilize technical aptitude and problem solving skills to create thorough documentation of client questions and needs, and then resolve their issues by finding software solutions for clients across all of Foundant's software.

FIGURE, *Office Coordinator* **Helena, MT**
Jan. 2019 – Aug. 2019

- Led the everyday office experience for a team of 25 software engineers.
- Tracked expenses and expense reporting.

MONTANA CODE SCHOOL, *Web Development Student* **Missoula, MT**
May 2018 – Aug. 2018

- Experienced in JavaScript, Git, Github, HTML/CSS, NPM, Webpack, Babel and the Agile methodology.
- Developed two web applications using React, Vue, Bootstrap 4, Express, Node, MongoDB and PostgreSQL and presented them at public demonstration days.
- Full-time immersive boot-camp to gain experience in the latest full-stack web development technologies.

MONTANA CONSERVATION VOTERS, *Development Director* **Helena, MT**
Mar. 2016 – Mar. 2018

- Maintained Salesforce-based Luminate CRM database.
- Built and sustained relationships with major donors.
- Participated in electoral and legislative strategy meetings.

MONTANA DEMOCRATIC PARTY, *Candidate Director* **Helena, MT**
Nov. 2015 – Jan. 2016

- Served as interim candidate director/in-state fundraiser for U.S. Congressional candidate.

- Coordinated with campaign consultants and allies to launch all fundraising, research and communications strategies utilizing NGP VAN database.
- Fundraised over $263,800 – more than any previous Democratic candidate for the U.S. House in Montana had raised in their first fundraising period.

MONTANA STATE SENATE, *Legislative Aide* **Helena, MT**
Jan. – April 2015
Jan. – April 2013

- Advised and consulted with Montana Senate Minority Leadership on policy strategy, tracked legislative actions for caucus and coordinated caucus priorities with the Governor's office.
- Coordinated caucus media strategy through press conferences, social media, and briefing materials.
- Hired and managed Senate minority legislative staff of four.

MONTANANS FOR FREE AND FAIR ELECTIONS, *Deputy Campaign Manager* **Helena, MT**
April 2014 – Nov. 2014

- Oversaw development of statewide marketing campaign, managed press relations, acted as campaign spokesperson and wrote press releases.
- Coordinated website and social media presence and developed online strategy in conjunction with a paid and earned media program to convey a relatable message strategy.
- Facilitated statewide board meetings, coordinated editorial board visits and letters to the editor program, recruited and managed third party spokespeople for public meetings.

MONTANANS FOR LEWIS, *Finance Associate/Call Time Manager* **Helena, MT**
Sept. 2013 – April 2014

- Managed donor relationships and tracked programmatic benchmarks which led to the designation as a national top priority campaign by the Democratic Congressional Campaign Committee.
- Developed talking points, prepared speech remarks, coordinated campaign events, planned fundraisers, organized constituent meetings and endorsement interviews.

MONTANA DEMOCRATIC COORDINATED CAMPAIGN, *Field Organizer* **Helena, MT**
- in partnership with ***Senator Jon Tester's Re-election Committee*** May – Nov. 2012

- Contributed seven days a week to a statewide grassroots field operation that generated over 3.2 million voter contact attempts which led to the successful re-election of Senator Jon Tester.
- Recruited, trained and managed over 100 volunteers and ensured they used tested scripts and sophisticated targeting which led to a historically high voter turnout in Lewis and Clark County.

U.S. SENATE DEMOCRATIC STEERING AND OUTREACH COMMITTEE **Washington, DC**
Mar. 2010 – May 2012

Associate Director

- Established and maintained relationships with diverse external stakeholders, including industry and advocacy groups, with an emphasis on labor, womens, seniors, youth, and rural issues.
- Built targeted communications plans for Senate Majority Leadership to strengthen coalitions around legislative priorities.

Outreach Coordinator

- Supported partnership with House Leadership and the White House to organize large outreach events, formal briefings, and summits.
- Oversaw logistics and managed budgets for all outreach initiatives, including roundtables, press conferences, briefings, interviews, and videos.

Background

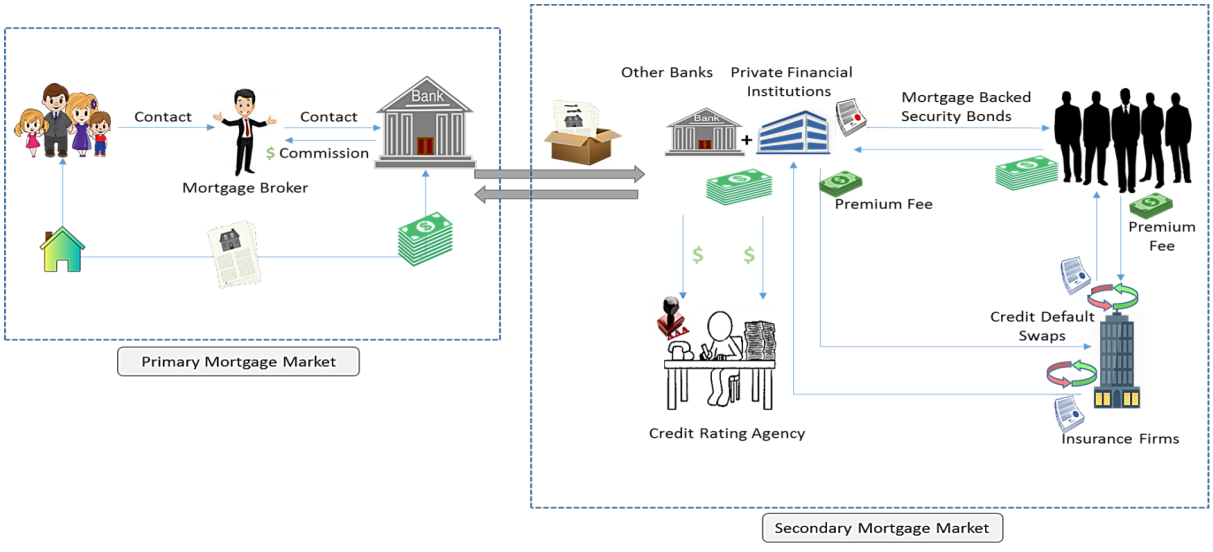
Brightvine aims to help the everyday investor in accessing mortgage investments in the secondary mortgage market. According to Joe Vellanikaran, the founder of Brightvine,

> The main purpose of Brightvine is to increase efficiency and liquidity in the mortgage market, and thus increase home ownership overall. Legacy institutions like Fannie Mae and Freddie Mac were created by the US government to increase home ownership for the middle class. But one unfortunate outcome of new regulations following the subprime mortgage crisis of 2008 is that the cost per mortgage has increased significantly. And those costs are being passed along to the borrower, which in the end means that less people can afford mortgages. However, when we increase efficiency in the securitization process, it increases the amount of capital that's available to provide mortgages, thus bringing prices down and making mortgages more affordable for more people. [10]

Secondary Mortgage Market

The way banks make money is by issuing out loans and collecting high rates of interest. However, once these loans are issued, banks often sell away the rights to these loans in order to regain the capital to issue more loans. Investors can purchase these loans like they would purchase any other investment and begin collecting the payments from the borrower as their way of receiving a return on their investment. Like other investments, there is risk involved, and some players are more willing than others to take bigger risks with a chance of higher reward. Because of this, the purchasing of loans is not done at a set price. The willingness of investors drives the price they are willing to pay.

The image below illustrates the many players involved in the secondary mortgage market and how often money changes hands while the original borrower rarely notices a change.



Mortgage Glossary

Within Brightvine's Mortgage Desk, many individual mortgages are listed for investors to pursue. Attached to these mortgages are hundreds of data fields related to the loan, the borrower, and the property. During the lending process, a bank performs a process called underwriting where they analyze the risk of lending to the borrower and make a decision whether to offer the loan at all. To perform this, they collect information from the borrower about their credit, employment, assets, and other factors. Because the bank collects this information when they originate the loan, these data points are also available with the loan in Brightvine's Mortgage Desk so they can be used to assess the risk of purchasing the loan in the secondary market.

These points will be crucial to this project as they will make up most of the input we analyze in order to provide the suggested price point for purchasing a given loan. Some of the features are described here:

- Credit
    - Credit scores
    - Credit history, including derogatories of all tradelines, mortgage-related delinquencies, etc.
    - Liability assessment
    - Borrower fraud report
- Collateral
    - LTV - Loan to Value ratio (loan on a property compared to property value)
    - Collateral Score (from an appraisal of the property)
    - Collateral condition /health /safety issues (from appraisal and/or inspection)
- Income
    - DTI - Debt to Income Ratio of the borrower
    - Length of employment
    - Receipt of consistent income (from the last two years)
    - Asset assessment

Machine Learning

To understand the methods described in this document, it is important to be familiar with the language of Machine Learning and the terms commonly used. Machine learning as a whole is a branch of artificial intelligence that focuses on the use of data and algorithms, trained through the use of statistical methods, to make classifications or predictions. The methods often imitate the way that humans, or even the natural world around us, operate, learn, grow, and advance [2].

There are many different types of models to choose from when beginning a machine learning problem. And within those, there are many variations of each type of model. But, when selecting a model to use for any machine learning problem, the ***No Free Lunch*** principle must be

considered. The theorem states that all algorithms perform equally, on average. When comparing the success of different models, how well it performs is determined by how aligned the learning algorithm is with the actual data. [3] At the end of the day, once the models are tuned, they all perform the same. So, one must implement some preference bias and just pick one and roll with it. The biggest thing to consider for this preference is the time and space complexity of the different models and how those align with the performance ability of the machine in use.

Another factor in the preference decision is ***Occam's Razor***. Sometimes referred to as the principle of parsimony, this encourages developers to select the simpler model when given the choice. When all things are equal with performance and complexity, select the model that is less complicated to implement or understand. Keep it simple.

One process that has been mentioned already is ***tuning***. This process involves the parameters that are associated with a particular ML model. The parameters are values that must be passed into the model that affect the various calculations and equations in different ways. Tuning these parameters involves running a model and all of its training processes, checking the success of the model when evaluating the test data, and tweaking the parameters. The tuning process is complete when a reasonable number of parameter values have been tested in combination with each other and the values that yielded the best results are selected. These parameter values are then passed into the algorithm whenever it is used to test a new data point.

The ***training*** process is the most crucial phase of model preparations. Without training, the model never learns and can never provide the desired insights. When a model is trained, its calculations, weights, and other values are tweaked over a process of many iterations until the equations best describe the data being used for training. Some models need only to look at the training data as a whole and use the distributions of feature values as data to inform their decision-making process. This is still training. Other more complicated models look at each data point individually, try to make a prediction using random weight values in a complicated calculation, and then use a different complicated calculation to change those weight values to better lead the model to the right output value. Many of the training algorithms involve complex calculus and don't look like they should work. Then after some iterations and some patience, the performance of the model begins improving and you can watch your model learn.

For this project, we will be finding data with the desired input features that will help create an informed price suggestion. With this, we will be seeking out data from past mortgage sales that will be labeled with a value that this mortgage was bought and sold at previously. When training a model with data like this, it is called ***supervised learning***. When the training data set has actual truth values associated with each example, the model can use this to learn the ways in which it is making good predictions and the ways in which it is performing poorly.

The first of two models we will test uses a ***K-Nearest Neighbors*** algorithm to make a decision based on the points closest to the test point. There is no real training involved in this method other than simply reading the training data. This algorithm operates on the assumption that similar points exist close together. When making a prediction, the model finds the distance between the test point and each of the points in the training data. The parameter k determines how many points to consider as the nearest neighbors to this test point. Once the neighborhood is determined, the value assigned to the test point is found by checking the most common class in the neighborhood or by averaging the true values of the neighbors.

The second option for our model implements a ***Neural Network*** to perform the prediction calculation and is considerably more complicated. The network contains layers of nodes. Every node in one layer is connected to each node in the next layer. These connections are each assigned a weight value which acts as the key piece in forming a prediction. These weight values are what the model 'learns' through the training process. At the input layer, there is a node for each attribute in a data point. At the output layer, the number of nodes depends on the type of problem you are trying to solve. For a regression problem like the one in this project, there is only one output node that holds the value of the prediction. Each node in the network takes into account the weighted values it is receiving from the preceding layer, sums them up, and squishes the value down to be between zero and one. This process takes place in every node in every layer of the network.

To provide more flexibility within the model, ***hidden layers*** can be added. The number of hidden layers and the number of nodes in each hidden layer are all tunable parameters. Sometimes adding more layers helps the prediction by adding more opportunities for the model to provide special treatment to different attributes to make them more or less impactful on the overall prediction. However, it can also make the model too complex, causing it to take much longer to reach an optimally trained state without seeing much improvement over a simpler model. Hence, the reason these values are tunable and should be tested as such. The biggest benefit of using this model is that performing a prediction on a new test point is very quick and could provide the user of Brightvine's app rapid feedback after opening the details of a loan.

If the neural network ends up being the model of choice, we will have to consider a couple of different training methods. The first one we have experience with is called ***Backpropagation***. This method involves testing each point in the training data, comparing the prediction to the actual value, and using the difference to suggest changes in the weights that would allow the model to better predict the correct value. This is repeated for every point in the training set, and all of the weight suggestions are averaged and applied to the model's weights. In order to see improvement, many iterations are required as the weights shift very slightly after each run through the training set. Each point is vying for the model to change to best suit itself and the model is eventually swayed towards a solution by the majority of the training data.

Another option for training uses a ***Genetic Algorithm*** that behaves just like a species in the wild evolving over time through the theory of survival of the fittest. In this method, many different networks are created to form a population. Each member of this population has its own set of weights. Over many iterations, each network is tested, and the best-performing models create new offspring by sharing some of their weight values with each other. These new offspring replace some of the worst-performing networks in the population. Over time, the best-performing models survive, reproduce, and introduce some new variability in their weights, and eventually, a very well-performing model emerges with weight values trained to fit the training data associated with the problem. This best-performing model and its weight values survive the evolution process and are then used to create future predictions.

These are different types of machine learning models and training processes that were tested in the research stages of this project during the Fall and Spring semester. Again, because of the No Free Lunch policy, no model will ever clearly outperform the others once it is perfectly trained and tuned. In our research, runtime and space complexity will be heavily considered when selecting the final model.

Prior Work

All investors are interested in the likelihood of a mortgage defaulting, in which case a borrower is no longer able to make payments on their mortgage. Before the time of computers, a great amount of time was committed to manually screening loan details for signs of default. With the development of computers, and further, machine learning, time is now being devoted to developing a computational algorithm that can make these predictions for us.

Due to the many stakeholders involved, many parties have researched machine-learning solutions in the field of mortgage default prediction. Existing research includes Akindaini's exploration of the machine learning methods to predict default classification of mortgages, finding the random forest model to perform with 95.68% accuracy utilizing the publicly-sourced Fannie Mae Mortgage Dataset [4]. Lai of Jiliang University found the AdaBoost ML model to achieve 100% accuracy in the Xiamen International Bank dataset [5]. Beyond default prediction, mortgage investors are interested in the cumulative value of a mortgage, which involves an accurate valuation of the property. Niu et. al presents a successful property valuation system with a back propagation neural network model [6]. We aim to explore a similar machine learning approach to encompass the valuation of a mortgage.

Schedule

Throughout development, we will be using Trello boards for project management in alignment with two-week sprints and bi-weekly meetings with Brightvine. We selected the Scrum life cycle because it matches Brightvine's existing methods with two-week sprints and supports the iterative methods necessary for machine learning development. There are six basic phases of machine learning development, shown here:

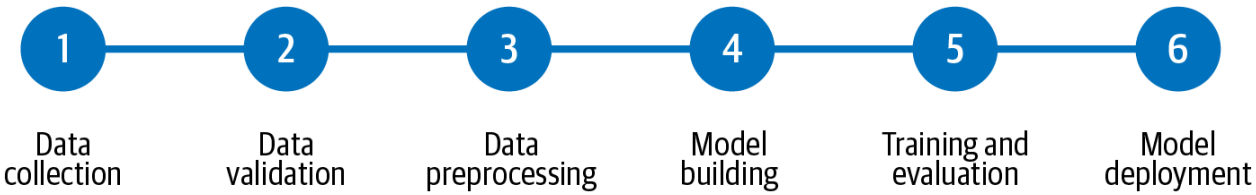


Figure 1: Machine learning development workflow, as described in article about ML design patterns [7]

While we will be operating under an iterative process, it is unlikely that we will return to the beginning once we reach model deployment. Our team will be moving forward along this path, but will likely bounce back and forth between a few phases before fully moving on to the next. Each two-week sprint will be treated more like a timeline to complete a task, rather than a time to fully produce a prototype and further develop that prototype in later sprints. While the process is iterative, it may look more like an iterative variation of the waterfall method.

We will add flexibility for our student schedules by using a private Trello board as opposed to Brightvine's company-wide project management technology, Jira. All group members will be responsible for creating and updating Trello boards regularly.

Our strict project timeline is below:

Beyond these deadlines, our work schedule can further be divided into these personal tasks:

In the planning stage, Lauren designated three hours per week for meetings, research, and writing. She primarily focused on the abstract, presentation slides, and proposal statements. In the spring, Lauren will designate five hours per week for development, testing, and meetings each week. This will break down into two hours of meetings and three hours of independent work. She will encourage the team to set and meet deadlines and manage the Trello board.

In the planning stage, Jacob dedicated two hours per week to meetings, research, and writing for the proposal document. He focused on providing background for the machine learning processes that may be used next semester as well as contributing to diagrams and program design. In the spring, Jacob will shift his focus to implementation and research to understand the relevant datasets and create the predictive model. During this time, he will dedicate five hours per week to writing code, meeting with group members, and meeting with the stakeholder.

In the planning stage, Kate designated three hours per week for meetings, research, and writing. She primarily focused on the abstract and proposal statement. During the spring semester, Kate designated three hours per week for development, testing, working with the datasets, team meetings and addressing the needs of the stakeholder.

Proposal Statement

User Stories

To help guide our design and implementation plan, we created the following user stories:

- As an investor, I want to know whether or not a given loan is a good investment so that I can make smart investing decisions.
- As an investor, I want to know an appropriate price for a given loan based on other similar loans so that I know whether or not I'm getting a good deal.
- As Brightvine, I want a tool that can be accessed in our existing Mortgage Desk so that our platform can be as useful as possible, making it attractive to investors.
- As Brightvine, I want this tool to use new and relevant data so that our suggestions are fresh and accurate.

Functional Requirements

The Mortgage Valuation Tool needs to output a recommended buy/sell price point for a specified mortgage and display it in mortgage details. For this to occur, the tool must:

- Accept essential mortgage details as input
- Accept current economic and real estate conditions as input
- Analyze mortgage valuation with machine learning solution
- Output price point as an API endpoint
- Feed price point into mortgage details frontend component

In addition to this core feature, if we have the time, a user should be able to batch-select many mortgages and view a composite valuation price point. This adds the following requirements:

- Analyze composite mortgage valuation with machine learning solution
- Display composite price point

Each investor has their own view of risk and what contributes to a valuable mortgage purchase. Therefore, users would benefit from the flexibility to change valuation parameters within the Brightvine Mortgage Desk. This functionality includes allowing investors to:

- View key valuation fields
- Adjust importance of fields
- Adjust risk tolerance

Nonfunctional Requirements

Beyond the functional requirements, we aim to meet a series of non-functional requirements specified below:

- Security
    - By definition, our valuation tool works directly with real mortgage data and personal information, including credit scores and home addresses. Due to the confidential nature of this data, security, including data encryption, will be a top priority.
- Storage
    - Developing predictive machine-learning algorithms requires immense amounts of training data. However, for the sake of efficiency, we need to carefully consider the storage requirements of such data and cap the amount of data processed at a time. This may include taking representative samples of mortgages across different regions and property types in order to minimize file sizes.
- Compatibility
    - The tool must be fully compatible with existing Brightvine products and web browsers.
- Scalability
    - The Mortgage Valuation Tool will initially be utilized solely within the Brightvine Mortgage Desk, first with Brightvine investors, and then with public investors. In the future, however, this tool could be utilized by larger corporations in the secondary mortgage market. Through development, we hope to create a tool that is usable not only within Brightvine products but also extendable to external products.
- Usability
    - We aim for the Mortgage Valuation Tool to be used without error by most users.
    - We will apply the Pareto Principle, which, in the case of usability, states that "80% of your users use 20% of your features." [8]. The result of this is displaying the primary 20% of the software capability clearly enough that a novice user will correctly interact with the product without instruction. The additional 80% of capabilities, such as adjusting risk tolerance and preferred mortgage properties, will be utilized by approximately 20% of users and have more margin for complexity.
- Data Integrity
    - When aiding in major financial investments, this tool must always provide reliable results and perform as expected. A thorough testing-harness should be implemented to ensure data integrity and reasonable predictive results.

Performance Requirements

The mortgage market is a rapidly changing environment as it interacts with the global economy, national housing market, regional statuses, and individual money management. Due to this dynamic nature, we aim for the data upload and processing time of the model to be minimal so that frequent data updates can be made daily. While industry leaders in FinTech are able to track going rates such as stock market prices *by the minute*, we are aiming for a modest timeframe of hourly updates. However, we aim for full data processing to take no more than ten minutes.

Interface Requirements

The Mortgage Valuation tool consists of seven interfaces:

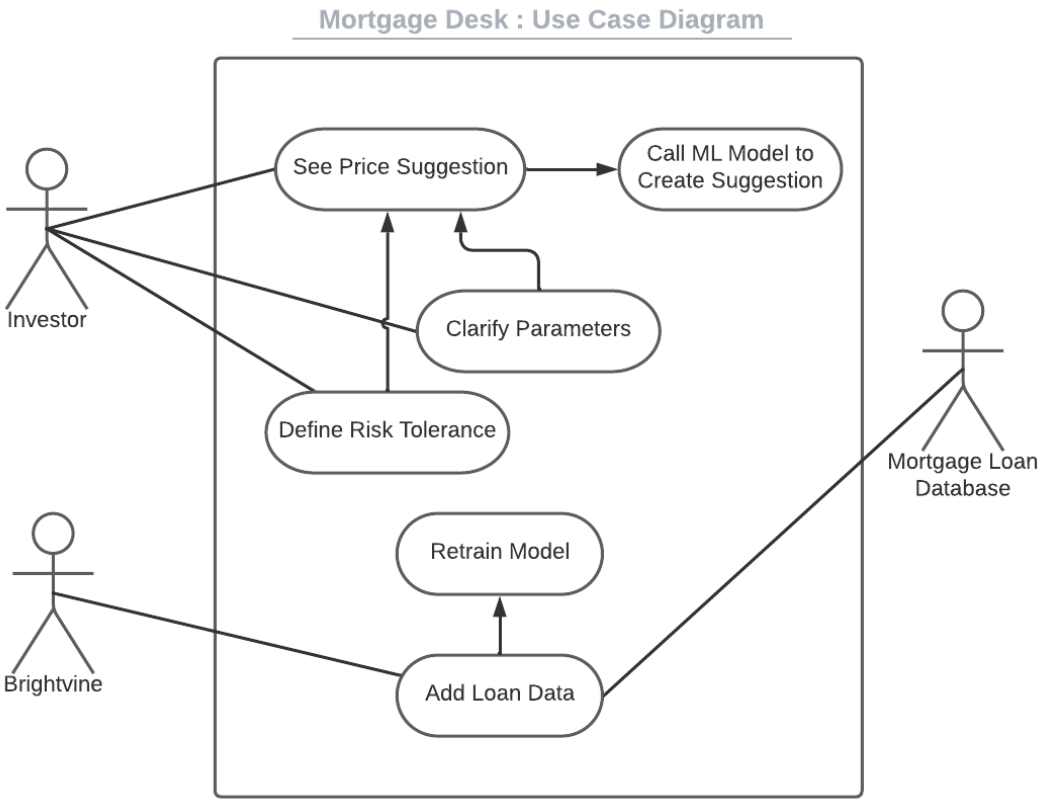
- **Investor**
- **Brightvine admin**
- **Client**
- **Server**
- **Database**
- **API**

Between these interfaces, the following interactions occur:

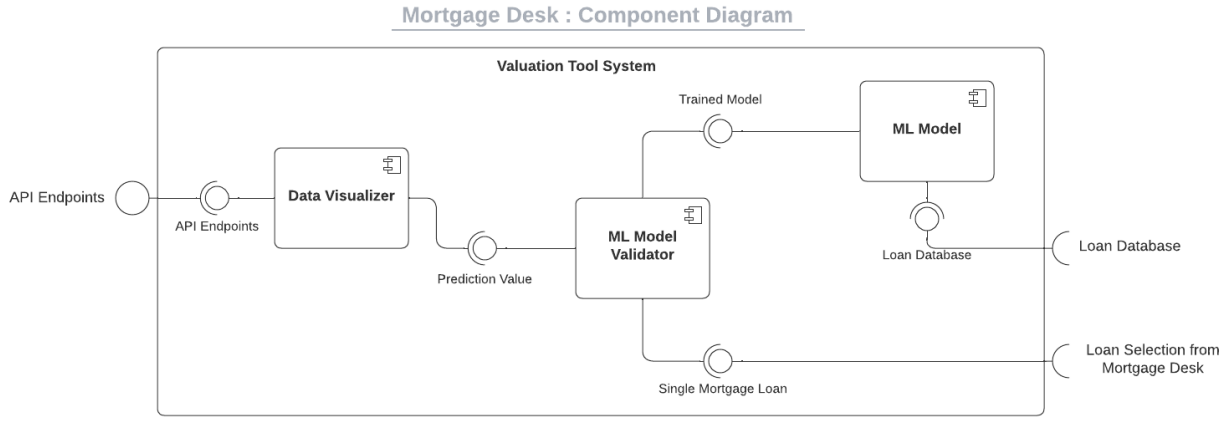
- The **Investor** selects mortgage details on the **client** interface.
- The **Brightvine admin** uploads loan data to the **server**.
- The **client** displays the predicted price point to the **investor**.
- The **client** sends mortgage details to the **API** via POST request.
- The **client** retrieves the predicted price point from the **API** via GET request.
- The **API** passes mortgage details to the **server**.
- The **API** returns the predicted price point from the **server** to the **client**.
- The **server** uploads loan data to the **database**.
- The **database** passes new data to the **ML model**.
- The **ML model** outputs the predicted price point to the **server**.

Architectural Design Documents

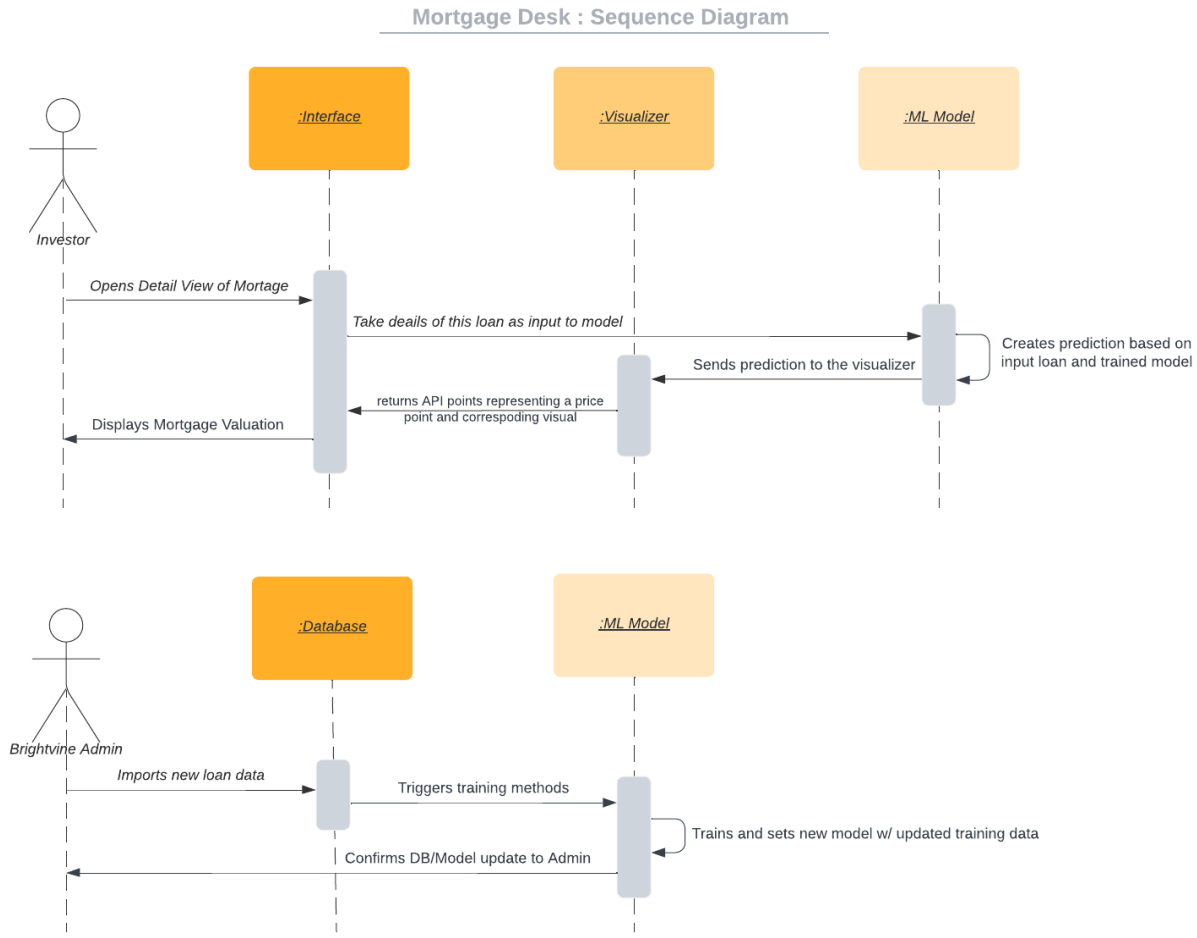
Below are three design documents that outline the proposed design for the system developed during the spring semester. The Use Case Diagram outlines the actors in play when certain functionality is being performed.



The Component Diagram shows how functions perform within the context of the overall system and how data serves as input and output for these functions.



The Sequence Diagram outlines the flow of the data and overall program, showing which actions receive or send certain information to different components of the system from the interface to the trained machine learning model.



Within this system, mentioned is a visualizer object. Due to the nature of python and machine learning programming, the program will not be object-oriented in the way that java programs are, so we will not be using true objects and therefore a class diagram would not be effective. But this visualizer object will simply be a compartmentalized group of functions that take in the output of the ML model and turn it into valuable information to be sent to an API and later the interface. This may include de-normalizing the prediction into a real price point suggestion, depending on the preprocessing of data that occurs, and assigning a corresponding visual representation.

Development Standards

Tech Stack: Considering that our product will be integrated with existing Brightvine software, our selected technology stack consists of similar technologies to those already used within the company, in addition to some solution-specific technologies:

- Machine Learning: Python, NeuroLab library
- Back-End: Flask
- Front-End: HTML, CSS, JavaScript

Design Pattern: When designing the structure of the API, it is important to consider it's usability and readability for any programmer. For our design pattern, we decided to follow the REST API design pattern, which is a standardized approach to designing app routes that rely on a set of HTTP methods and carefully selected keywords. In my Flask application, we implemented this pattern to provide a consistent and predictable interface for clients to interact with the API. We used the HTTP methods GET, POST, and DELETE to perform actions and created hierarchical endpoints to represent the information being displayed. For example, the endpoint '/<loan_id>' represents a specific loan resource that can be retrieved using the GET method, while the endpoint '/<loan_id>/delete' represents a specific loan resource that can be deleted using the DELETE method. By implementing the REST API design pattern in our code, we created a standardized and reliable interface with improved maintainability and scalability.

Design Tradeoffs: Our project mostly deals with backend development, drawing data from a database, and preparing a machine learning model to perform predictive calculations. We will be dealing with very little front-end development. In fact, the one design decision we made was to not implement a complex frontend framework, such as React.js that is used by Brightvine. Our frontend code, utilizing vanilla HTML, CSS, and JS, is sufficient to demonstrate the functionality of our solution but is not intended to be transferred to Brightvine.

For Brightvine, on a mortgage loan's details page, the price point suggestion provided by the ML model must be displayed. One option we faced was to write an independent bot that can be plugged into any webpage and would contain all the necessary front-end code to be displayed by the web browser right away; like a React component. After discussion with our team and our stakeholder, we have decided to instead use an API to display the data needed for the web browser rather than a transferable component. Because of this decision, Brightvine can have access to the data provided by the ML model anywhere in their system. All they have to do is trigger the model to run by sending it the input from a given loan. This provides more flexibility, as the model's data can be used anywhere by simply requesting it from an API. This ultimately serves the stakeholder better as they can more seamlessly use their existing frameworks, design elements, and themes to create their own label that will draw from the valuable information our project provides.

Expected Results

Bringing all of these ideas together, we aim to have a working prototype of a Mortgage Valuation Tool that accurately predicts a buy/sell price point given mortgage details and user input. Within the scope of this class, we expect the following specific results for success:

1. The Mortgage Desk can successfully retrieve price point data and visualization from the Mortgage Valuation API.
2. The ML Model outputs a price point that is accurate to mortgage market standards.

Evaluation of a supervised machine learning model can be performed in numerous ways. Often a type of loss function is used which takes in both the predicted and actual outputs of the rows in the dataset and returns the value of the loss function. For our case, we will be using Mean-Squared Error (MSE) and a modified accuracy calculation for numerical prediction values. The model will be optimized by minimizing the MSE. Unfortunately, the MSE value is difficult to interpret and does not provide a good way to verbalize the success of our prediction in terms of a percentage success rate. The value alone is fairly meaningless to a human evaluator, but when compared to the MSE of previous iterations in the training process, it is still a very effective metric for finding convergence when training an ML model.

Because of the limitations of MSE, we will try to employ a modified accuracy metric for understanding the success rate of our prediction algorithm. This will include using an alpha level, likely $\alpha = 0.1$, as a margin of error where the model's prediction will be deemed correct if it falls within the range around the actual value. With the time allotted to this project, we expect to reach an accuracy of 90%.

As a result of this project, we expect Brightvine to benefit through increased trust and engagement by investors within their Marketplace platform. By providing a provably accurate price point prediction, investors can be sure of their investment decisions without external advising, in turn broadening the accessibility of mortgage investments to a variety of socioeconomic and education levels. This ties in directly with the mission of Brightvine -- to "uncloak the black box of opportunity" to all investors [11]. The Mortgage Valuation Tool should do just that, distilling the complexity of the secondary mortgage market into a price point that provides a step up to novice and seasoned investors alike.

Beyond Brightivine, we anticipate our Mortgage Valuation Tool could have a significant impact on the secondary mortgage market and financial technology industries globally. Such a tool could be implemented in other mortgage marketplace software or used by private investors as they aim to gain the best insights on their own. The tool could also be fine-tuned by Brighvine or other entities to take in more specialized input and parameters to provide more sophisticated

suggestions. The scalability and relevance of this tool and the methods behind it are nearly endless as the excitement around machine learning continues to grow.

Final Results

Throughout the fall semester, our team conducted extensive research into the problem and necessary requirements, began to research a solution, and crafted detailed proposal documents. During the subsequent spring semester, our team continued our research efforts, focussing on selecting a suitable dataset, identifying appropriate prediction methods, and implementing a solution prototype. As we developed our solution, our team pivoted from the original plan of using a neural network with a multi-layer perceptron model trained by either backpropagation or a genetic algorithm like particle swarm optimization. Instead, due to the nature of the datasets available to us, we selected an unsupervised learning approach-- a K-Means clustering algorithm, followed by a K-Nearest Neighbors prediction that leveraged the reduced dataset generated by the clustering process.

Data and Preprocessing

We selected the Single-Family Mortgage-Level Properties, high-cost single-family mortgages purchased and securitized by the Enterprises (National File C) from the Federal Housing Finance Agency dataset because the FHFA is an independent federal agency responsible for regulating and supervising the Federal Home Loan Bank System and the government-sponsored enterprises Fannie Mae and Freddie Mac. The FHFA website provides access to various datasets related to the housing and mortgage markets, including the FHLBank Public Use Database, the National Mortgage Database, and the House Price Index. These datasets are made available to the public to support research and analysis of the U.S. housing market and economy. Researchers and analysts can download the datasets in various formats and use them to conduct studies on a wide range of topics related to housing finance, mortgage lending, and housing affordability.
We then preprocessed the data to include the following essential fields: Total Monthly Income Amount, Loan Acquisition Actual Unpaid Balance Amount, LTV Ratio Percent, Borrower Count, Note Rate Percent, Note Amount, Housing Expense Ratio Percent, Total Debt Expense Ratio Percent, Borrower 1 Credit Score Value, Borrower 2 Credit Score Value.

After the dataset was selected, downloaded, and columns were chosen based on the value and usability of the data, Jacob began the process of data evaluation. The main problem with the public data we could find was that it did not come with columns describing anything about the performance of the loan or the reliability of the borrowers. Due to the nature of machine learning, this essentially eliminated the use of any supervised learning techniques like the ones we proposed. When using a model like a neural network, in order for the machine to learn and the model be trained, it needs to know whether or not it is doing a good job of predicting a value. With no data telling us or the machine what makes a good borrower or what makes a loan a reliable investment, there was no way for a ML model to train itself towards improving performance. Thus, we had to move into exploring unsupervised learning techniques for our solution.

First, Jacob performed some exploratory analysis on the dataset. The data was read from a csv into a data structure called a DataFrame using the pandas library for python. A few columns used values to represent missing data, so the corresponding rows were removed. After analyzing the distribution of values in each column, it was deemed appropriate to treat points with values outside of three times the interquartile range as outliers and remove them from the dataset. Together, these methods were able to effectively clean the dataset while maintaining 98.3% of the points in our 236,758 point training set.

While each column seemed to have information that would be valuable to estimating the value of a loan, exploration revealed that several columns were very highly correlated. For example, the Note Amount (initial loan amount) and Loan Acquisition Actual UPB Amount (remaining unpaid balance when acquired by secondary market investor) were the same in a vast majority of the columns and so were highly correlated. Similarly, the Borrower Count and Borrower 2 Credit Score Value were highly correlated because most loans had only one borrower, which always leads to the credit score value for borrower number 2 being N/A and therefore represented by a 9 in the dataset.

Despite the correlation, it is still valuable to maintain this information in the dataset. We still want to know when the unpaid balance has changed since origination because that shows proactive payments by the borrower. We still want to know the credit score of the second borrower when applicable. In order to remove some of the correlation without dropping data features, we employed the dimensionality reduction method of principal component analysis (PCA) through the use of the decomposition class of the sci-kit learn python library. In order to preserve 90% of the explained variance in the data, we only reduced the data from eleven attributes down to six.

Throughout the training and testing processes, we utilized remote repositories to imitate remote databases for storing, reading, and updating data. Initially, the training and testing data were all stored in this repository. After training, new information was pushed to this repository including the clusters of data, the cluster centroids, and the actual python objects used to clean, standardize, and cluster the data. During testing, the python objects and appropriate data cluster were pulled from the repository and read into the algorithm to be used for the prediction.

Model Training

Without the ability to train a neural network using the data's ground truth, we turned to clustering methods to provide insights. The chosen algorithm, K-Means, clusters the data into a set number of clusters by randomly assigning points to be cluster centers, then repeatedly assigning points to their nearest clusters and recalculating cluster centers until the model converges and the centers no longer change. Since there is no correct answer when clustering unclassified data, there are

two different evaluation metrics used to determine the effectiveness of the clustering: inertia value and silhouette score.

Inertia is essentially an overall measure of how far the data points are from their cluster centers. This value should be minimized. The silhouette score incorporates both how far a point is from all of its cluster mates and how far it is from the points in the other clusters. The comparison of the cohesion and separation of clusters results in a value between -1 and 1 that should be maximized. When plotting these scores in tandem for different cluster counts, the point where there is an elbow in the inertia graph and a maximum in that area in the silhouette graph is the optimal number of clusters that should be used for clustering a dataset.

That point was determined to be at eight clusters for our mortgage dataset. However, we ran into trouble with calculating the silhouette score, which is relevant for the prediction method explored later in this paper. In order to calculate this value, each point in the dataset must be compared to every other point, and their distances computed. With a dataset as large as the mortgage dataset being used in this project, the python script running this exploratory analysis failed to produce silhouette scores for more than two or three cluster counts in any extended period of time. It was determined to be information that we could proceed without in regards to finding an optimal number of clusters, however it provided crucial insight into the kind of time complexity we were working with when computing distances between every single point in the dataset.

Proceeding with the information learned about clustering, the data was sorted into eight clusters using the K-Means algorithm through the use of the cluster class of the sci-kit learn python library. This was to serve as the training for a future testing/prediction algorithm. New points could then be read into the program and be assigned to the cluster into which they best fit. It is important to note that clustering algorithms sort data into like clusters. This is not classification. It makes no claim to be classifying points into certain classes, only clustering the data together based on similarity.

Price Point Prediction

Without the ability to run a point through a trained neural network for prediction, the clustering provides a much less sophisticated method for predicting a price point. After assigning the new test point to a cluster based on the centroids of the previously trained clusters, all of the data points from that cluster were read into the program from the database. By taking the assigned price of each of the points in this cluster, the average price was assigned to the test point as its prediction value. This was based on the assumption that the clusters represented a tightly knit group of data points with very similar attributes like loan amount, monthly income, and loan to value ratio.

It was discovered, however, that this is a very ineffective method of prediction. The assumption that each cluster represents a tightly knit group of data points is likely false considering the massive amount of data being used. Through this clustering method, each cluster consisted of about 30,000 points. Also, since a test point is always assigned to one of these eight clusters, there would only be eight possible outputs for our prediction algorithm. Surely, not every mortgage loan is best represented by one of these eight values. So, we found that the supposedly optimal eight clusters were not actually very meaningful and that averaging the prices across a huge number of points would not be very insightful.

Continuing with the assumption that very similar points will be grouped closely together, we aimed to select only a few of the closest points to generate a prediction value. This method is called K-Nearest Neighbors (KNN). The model simply searches the entire dataset to find the given number of points that are closest to the given test point. As we discovered during data exploration, iterating through the entire dataset to compare distances is a very time-complex process and would not be ideal for providing a prediction within the parameter of our non-functional requirement of speed. Instead, since we already tested the ability to assign a new point to a cluster using the centroids, we then used that assigned cluster as a reduced dataset within which to perform the KNN search. And, since we were no longer looking for an optimal number of clusters for the K-Means algorithm, we increased the cluster count to 30 in order to produce datasets of around 8,000 points – a modest task for the KNN search.

This method allows the prediction algorithm to compare the point to only 30 values (cluster centroids) and in doing so remove nearly 97% of the data points from consideration – all of which would never be one of the test point's few closest neighbors considering they wouldn't even be in the same 8,000-point cluster. The KNN algorithm is able to search through this cluster of data to find the K-sized neighborhood in no time at all.

The size of this neighborhood was another tested parameter. We ran the search for several different points and looked at the neighorhoods for sizes ranging from five to ten. When ten points were included in the neighborhood, the farthest points started to have some very significant differences from the test point in a few columns. While some diversity is important for the prediction, it was determined that a K of seven would provide the largest neighborhood for prediction without reaching too far away from the test point and losing too much similarity. The assigned prices of each of these seven neighbors is averaged and that value is assigned to the test point as a prediction value. We believe this provides the best insight into a meaningful prediction given the data available.

Data Display and Interpretation

Over the course of several months, our team worked diligently to create a working demo of our mortgage valuation tool using Flask, HTML, CSS, and JavaScript. We began by designing a home page that allowed users to upload a singular mortgage in CSV format. Once the dataset was uploaded, it was passed to our REST API endpoint which would display all the mortgage details on a new page. We then linked this web application to the machine learning scripts through a series of buttons in which a user could retrain the model and generate a new prediction leveraging the K-Means clustering and K-Nearest Neighbors search algorithms. The results were then displayed on a separate page where users could view the mortgage details and predicted price point. We also included details on the changing value of the mortgage, and how it compares to similar mortgages.

When reading the prediction value of a single mortgage loan, if the prediction value is higher than the given price of the loan, an investor can know that this loan is priced lower than other very similar loans that have been sold. This does not necessarily mean we are giving the investor the “all-clear” to purchase the loan right away, but it does give them valuable insight into how this specific investment opportunity compares to other similar options. If the prediction value is lower than the given price, it may be an indication to the investor that this specific loan is overpriced and to continue searching for a better deal. Within the details of the loan, it also alerts the investor if the loan payments are current or delinquent and whether the value of the property has seen significant appreciation or depreciation. A delinquent loan is not always a bad investment, but it is very important to be aware of. Similarly, a loan on an appreciating house should become a much higher priority for an investor than one with a decreasing property value. Overall, we successfully created a functional and user-friendly interface that demonstrates how our solution quickly and accurately determines the value of a mortgage investment.

Appendix

Model: Database handling, preprocessing, training, and predicting

- GitFunctions.py
- PreProcessing.py
- trainingModel.py
- predictionModel.py

```
import pandas as pd
import git
from git import Repo
import os

# ------------------------------------------------------------
# Check that we are in the MortgageValuation directory and change if not
def checkDirectory():
    current_dir = os.getcwd()
    head_tail = os.path.split(current_dir)
    if head_tail[1] != "MortgageValuation":
        exit("ERROR ::::: Code is not running in the MortgageValuation repository directory!!")

```

```python
# ------------------------------------------------------------------

# ------------------------------------------------------------------
# Check for a conneciton to the dataset repository, should be in same folder as this repo, but not in this repo
# ==CREATE== a connection if there isn't one
# ===PULL=== current remote repository status
def getRepo():
    os.chdir('..')
    repo_dir = os.getcwd() + "/Brightvine_model_files"  # path of data repo
    try:
        repo = Repo(repo_dir)  # works if it exists
    except:  # if error, enters this chunk to initialize it
        print("Repo does not yet exist locally")
        print(".....Creating and Cloning now.....")
        # initiate new Git repo
        git.Git(repo_dir).clone("https://github.com/jjbrown23/Brightvine_model_files.git")
        repo = Repo(repo_dir)
    finally:
        assert not repo.bare

    repo.remotes.origin.pull()  # pull most recent repo down locally

    return repo, repo_dir


# ------------------------------------------------------------------

```

```
# ----------------------------------------------------------------
# ==LOAD== some data into a DataFrame
def readData(repo_dir, input_file_name, columns):
    data = pd.read_csv(os.path.join(repo_dir, input_file_name), sep=',', names=columns, header=0)
    print("...Reading an input file from remote repo...")
    print("....{} data instances with {} attributes....".format(data.shape[0], data.shape[1]))
    return data


# ----------------------------------------------------------------

# ----------------------------------------------------------------
# ==PUSH== new changes back to the remote repo,
# added a bunch of prints to show repo status throughout
def pushRepo(repo, output_file_name, commit_msg):
    print("\n...file edited")
    print(repo.git.status())

    if type(output_file_name) == str:  # if only one file, will show up as a string
        add_files = [output_file_name]  # and it needs to be thrown in a list to be read by git.add
    if type(output_file_name) == list:  # if multiple files, will show up as a list
        add_files = output_file_name  # and it's already in list form for git.add

    repo.index.add(add_files)
    print("\n...file added")
    print(repo.git.status())

    repo.index.commit(commit_msg)
    print("\n...file committed")
    print(repo.git.status())

    repo.remotes.origin.push()
    print("\n...file pushed")
    print(repo.git.status())
# ----------------------------------------------------------------
```

```python
import pandas as pd
import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
from tensorflow import keras
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

# VARIABLES - found in previous exploratory analysis
# NUM_CLUSTERS = 8
NUM_CLUSTERS = 30
NUM_PCS = 6

''' ============================================================ '''
''' === Functions for handling some preprocessing of the data === '''
''' ============================================================ '''

def removeMissing(data):
    print("\n....Dropping rows with Missing Values....")
    start = len(data)
    data = data.dropna()
    data = data[data['HousingExpenseToIncome'] != 999]
    data = data[data['TotalDebtToIncome'] != 999]
    end = len(data)
    if start - end == 0:
        print(".......No missing values, no rows dropped")
    else:
        print("......." + str(start-end) + " rows dropped with at least one missing value")
    return data


```

```
def removeExtremeOutliers(data, columns):
    print("\n....Removing extreme outliers....")
    print(".......Dropping rows with values outside the whiskers of length 3 x IQR\n")

    # gather quartile limits for each column
    limits = {}
    for col in columns:
        limits[col] = (3 * (data[col].quantile(.75) - data[col].quantile(.25))) + data[col].quantile(.75)

    # remove outliers from each column
    start = len(data)
    thisStart = len(data)
    for col in columns:
        data = data[data[col] < limits[col]]

        end = len(data)
        print("+++ Outliers removed from " + col + " column: " + str(thisStart - end))
        thisStart = end

    print("\n" + str(start-end) + " rows with outliers dropped from the dataset")
    print("...{} rows remaining...".format(end))

    return data
```

```
58  def normalize(data, columns):
59      print("\n....Standardizing the columns of data....")
60      print(".......Setting each column's values to have mean of 0 and std of 1")
61      years = data['Year']
62      data = data.drop(columns=['Year'])
63      columns.remove('Year')
64
65      ss = preprocessing.StandardScaler()
66      standData = ss.fit_transform(data)
67
68      # reshape years to concatenate to data
69      # years = np.asarray(years).reshape(len(years), 1)
70      # standData = np.concatenate((years, standData), axis=1)
71      # columns.insert(0, 'Year')
72
73      # transform back into a DataFrame so we can still use column headers
74      standDataFrame = pd.DataFrame(standData, columns=columns)
75
76      return standDataFrame, columns, ss
77
```

```python
import numpy as np
import pandas as pd
from datetime import datetime
import os, sys
import pickle

backend_dir = os.path.join(os.getcwd(), "backend/machineLearning")
sys.path.append(backend_dir)
import PreProcessing as pp
import GitFunctions as gf

from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# VARIABLES - found in previous exploratory analysis
from PreProcessing import NUM_CLUSTERS
from PreProcessing import NUM_PCS

```

```
# ------------------------------------------------

def getData():
    # Check that we are in the MortageValuation directory
    gf.checkDirectory()

    # Check for a conneciton to the dataset repository, should be in same folder as this repo, but not in this repo
    # ==CREATE== a connection if there isn't one
    # ===PULL=== current repo files/status
    repo, repo_dir = gf.getRepo()

    # ==LOAD== some data into a DataFrame
    input_file_name = "NewData.csv"
    fullColumns = ['Year', 'MonthlyIncome', 'UPBatAcquisition', 'LTVRatio', 'BorrowerCount', 'InterestRate', 'OriginationValue', 'HousingExpenseToIncome', 'TotalDebtToIncome',
    fullData = gf.readData(repo_dir, input_file_name, fullColumns)    # read data into pandas DataFrame

    # Run some preprocessing methods on this data for clustering
    no_missing_data = pp.removeMissing(fullData)

    columnsForOutliers = ['MonthlyIncome', 'UPBatAcquisition', 'LTVRatio', 'BorrowerCount', 'InterestRate', 'OriginationValue', 'HousingExpenseToIncome', 'TotalDebtToIncome']
    no_outliers_data = pp.removeExtremeOutliers(no_missing_data, columnsForOutliers)

    columnsForClustering = ['Year', 'MonthlyIncome', 'UPBatAcquisition', 'LTVRatio', 'BorrowerCount', 'InterestRate', 'OriginationValue', 'HousingExpenseToIncome', 'TotalDebtT
    reduced_data = no_outliers_data[columnsForClustering]
    print("\n....Reduce dataset to columns relevant for clustering....")
    print(".......{} data instances with {} attributes....".format(reduced_data.shape[0], reduced_data.shape[1]))

    std_data, columns, ss = pp.normalize(reduced_data, columnsForClustering)
    # trainData, testData = pp.create_folds(std_data)

    return no_outliers_data, std_data, columns, ss, repo, repo_dir

```

```
# ------------------------------------------------

def getClusters(data):
    # Reduce Dimensionality with PCA
    print("\n...Performing PCA...")
    pca = PCA(n_components = NUM_PCS)
    pca_data = pca.fit_transform(data)
    print(".....Data reduced to {} attributes.....".format(pca_data.shape[1]))

    # Use KMeans to cluster the PCA-ed
    print("\n...Performing K-Means Clustering...")
    kmeans = KMeans(n_clusters = NUM_CLUSTERS, n_init = 10)
    clusters = kmeans.fit_predict(pca_data)
    print(".....Data grouped into {} clusters using the K-Means algorithm.....".format(NUM_CLUSTERS))

    return kmeans.cluster_centers_, clusters, pca, kmeans

# ------------------------------------------------

def storeClusterData(repo_dir, std_centroids, columns, cluster_labels, fullData, ss, pca, kmeans):

    print("\n\n...Writing cluster data and data processing objects to files in the remote database repository...")

    # all file names for the three database files and data manipulation objects
    file_names = ["StandardizedCentroids.csv", "ReadableCentroids.csv", "ClusterLabels.csv", "ss.pkl", "pca.pkl", "kmeans.pkl"]

    ''' ================================================================================
        Turns out that storing these three files to the database is pointless, they won't be used in prediciton,
        just good for human visualization of the process
        ================================================================================ '''
    # Throw standardized centroids into a DataFrame and write it to a csv
    std_cent_df = pd.DataFrame(std_centroids, columns=["PC"+str(x) for x in range(1,NUM_PCS+1)])        # create a df
    std_cent_df.to_csv(os.path.join(repo_dir, "ModelOutputFiles", file_names[0]))          # write to csv in database repo directory

```

```
    # Translate standardized centroids back to original values that actually mean something, throw into a DataFrame
    read_cent_df = pd.DataFrame(ss.inverse_transform(pca.inverse_transform(std_centroids)), columns=columns)    # create a df
    read_cent_df.to_csv(os.path.join(repo_dir, "ModelOutputFiles", file_names[1]))          # write to csv in database repo directory

    # Store cluster labels for every point in the dataset as a csv
    np.savetxt(os.path.join(repo_dir, "ModelOutputFiles", file_names[2]), cluster_labels, delimiter=', ', fmt='% s', header="label")

    # ================================================================================================== #

    ''' ============================================================================================
        These cluster files and the model files will make up our "database" and will be pulled by the prediction script
        ============================================================================================ '''
    # Store each data by cluster
    for i in range(NUM_CLUSTERS):
        mask = cluster_labels == i
        cluster_df = fullData[mask]
        cluster_file = str(i) + "ClusterData.csv"
        cluster_df.to_csv(os.path.join(repo_dir, "ModelOutputFiles", cluster_file))

    # Store StandardScaler and PCA objects
    pickle.dump(ss, open(os.path.join(repo_dir, "ModelOutputFiles", file_names[3]), 'wb'))
    pickle.dump(pca, open(os.path.join(repo_dir, "ModelOutputFiles", file_names[4]), 'wb'))
    pickle.dump(kmeans, open(os.path.join(repo_dir, "ModelOutputFiles", file_names[5]), 'wb'))


    return file_names

```

```
112 # ------------------------------------------------
113
114 def trainingClustersDriver():
115
116     print("\n==== Preprocessing Data ====")
117
118     # set up repo, load in the data, run preprocessing
119     no_outliers_data, data, columns, ss, repo, repo_dir = getData()
120
121     print("\n\n==== Training/Clustering ====")
122     # Cluster the training data
123     std_centroids, cluster_labels, pca, kmeans = getClusters(data)
124
125     # Write cluster centroids to csv's in standardized and readable form
126     # Also write the cluster labels to a file
127     # Also save the data manipulation objects to a file for later use on test data (StandardScaler and PCA objects)
128     file_names = storeClusterData(repo_dir, std_centroids, columns, cluster_labels, no_outliers_data, ss, pca, kmeans)
129
130     # Push these files to the 'database'
131     msg = datetime.now().strftime("%d-%m-%y %H:%M") + " Cluster Database Update"
132     gf.pushRepo(repo, 'ModelOutputFiles/', msg)
133     os.chdir('MortgageValuation')
134
135 # ----------------------------------------
136 if __name__ == "__main__":
137     trainingClustersDriver()
```

```python
import numpy as np
import pandas as pd
from datetime import datetime
import os, sys

backend_dir = os.path.join(os.getcwd(), "backend/machineLearning")
sys.path.append(backend_dir)
import PreProcessing as pp
import GitFunctions as gf

from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.neighbors import NearestNeighbors
import pickle

import warnings

# VARIABLES - found in previous exploratory analysis
from PreProcessing import NUM_CLUSTERS
from PreProcessing import NUM_PCS
FILES_PATH = 'MortgageValuation/backend/database/uploadedFiles'

```

```python
# ------------------------------------------------

def getClusterData():
    # Check that we are in the MortageValuation directory
    gf.checkDirectory()

    # Check for a conneciton to the dataset repository, should be in same folder as this repo, but not in this repo
    # ==CREATE== a connection if there isn't one
    # ===PULL=== current repo files/status
    repo, repo_dir = gf.getRepo()

    ''' No longer need the cluster centroids - now we pull the kmeans model from the training script and use it for clustering the new point'''
    # ==LOAD== the cluster centroids into a DataFrame
    # input_file_name = "ModelOutputFiles/StandardizedCentroids.csv"
    # centroids = gf.readData(repo_dir, input_file_name, columns=["PC"+str(x) for x in range(1,NUM_PCS+1)])    # read data into pandas DataFrame

    ''' No longer need cluster labels - now we pull all the data for one cluster in a later step '''
    # ==LOAD== the cluster labels into a DataFrame
    # input_file_name = "ModelOutputFiles/ClusterLabels.csv"
    # cluster_labels = list(gf.readData(repo_dir, input_file_name, columns=['Label'])['Label'])   # read labels into a dataframe, convert to list

    # ==LOAD== the data manipulation objects for manipulating test data points
    object_files = ['ss.pkl', 'pca.pkl', 'kmeans.pkl']
    ss = pickle.load(open(os.path.join(repo_dir, "ModelOutputFiles", object_files[0]), 'rb'))
    pca = pickle.load(open(os.path.join(repo_dir, "ModelOutputFiles", object_files[1]), 'rb'))
    kmeans = pickle.load(open(os.path.join(repo_dir, "ModelOutputFiles", object_files[2]), 'rb'))


    return repo, repo_dir, ss, pca, kmeans

```

```
53  # ------------------------------------------------
54
55  def getTestPoint(repo_dir):
56
57      # ==LOAD== the test point from somewhere
58      input_file_name = "TestPoint1.csv"
59      fullColumns = ['Year', 'MonthlyIncome', 'UPBatAcquisition', 'LTVRatio', 'BorrowerCount', 'InterestRate', 'OriginationValue', 'HousingExpenseToIncome', 'TotalDebtToIncome',
60      fullPoint = gf.readData(repo_dir, input_file_name, fullColumns)     # load the data point into a dataframe
61
62      #print(fullPoint['UPBatAcquisition'].iloc[0])
63
64      return fullPoint
65
```

```
# ------------------------------------------------

def provideSuggestion(point, repo_dir, ss, pca, kmeans, fullPoint):
    # Determine which cluster the new point belongs to
    [cluster] = kmeans.predict(point)

    # ==LOAD== the data associated with that cluster from the database repo
    cluster_file_name = "ModelOutputFiles/" + str(cluster) + "ClusterData.csv"
    fullColumns = ['Year', 'MonthlyIncome', 'UPBatAcquisition', 'LTVRatio', 'BorrowerCount', 'InterestRate', 'OriginationValue', 'HousingExpenseToIncome', 'TotalDebtToIncome',
    full_cluster_data = gf.readData(repo_dir, cluster_file_name, fullColumns)     # load the data point into a dataframe

    # # Determine a price suggestion based on average price of cluster-mates
    # price = fullClusterData['Price'].mean()

    # Determine a price suggestion by running K-Nearest-Neighbor within the cluster data
    # Have to prepare the cluster data for distance comparisons
    columnsForClustering = ['MonthlyIncome', 'UPBatAcquisition', 'LTVRatio', 'BorrowerCount', 'InterestRate', 'OriginationValue', 'HousingExpenseToIncome', 'TotalDebtToIncome'
    reduced_cluster_data = full_cluster_data[columnsForClustering]
    std_cluster_data = pd.DataFrame(ss.transform(reduced_cluster_data), columns=columnsForClustering)
    pca_cluster_data = pca.transform(std_cluster_data)

    # Then create a new knn object for finding the nearest neighbors
    knn = NearestNeighbors(n_neighbors=7)
    knn = knn.fit(pca_cluster_data)                    # fit the object with the 'training data'
    distances, [indices] = knn.kneighbors(point)       # find the nearest points by passing in the 'test data point'

    # And make a prediction based on these neighbors
    neighbors = full_cluster_data.iloc[indices]
    price = neighbors['Price'].mean()
```

```
    # Add flags if the loan is currently Delinquent or has significant (app/dep)reciation
    fullPoint = pd.Series(fullPoint.iloc[0])

    if fullPoint['Performance'] == "Current":
        delinq = False
    elif fullPoint['Performance'] == "Delinquent":
        delinq = True

    if fullPoint['ValueChange'] >= 20:           # flagging if value has appreciated by at least 25%
        appr = True
        depr = False
    elif fullPoint['ValueChange'] <= -20:        # flagging if value has depreciated by at least 25%
        depr = True
        appr = False
    else:
        appr = False
        depr = False

    return price, delinq, appr, depr, neighbors['Price']

```

```
# ------------------------------------------------

def testOnePointDriver():

    #print("Hey look, maybe someday this will provide a prediction point.")

    # set up repo, load in the cluster data
    repo, repo_dir, ss, pca, kmeans = getClusterData()

    # load in new test point
    fullPoint = getTestPoint(repo_dir)

    # preprocess the test point using data manipulation objects from training data
    point = testPointProcessing(fullPoint, ss, pca)

    # provide suggestion - place point in a cluster and draw pricing data from cluster members
    suggestionNumber, delinq, appr, depr, neighbors = provideSuggestion(point, repo_dir, ss, pca, kmeans, fullPoint)

    print("Suggested price: ", str(suggestionNumber))
    print("Flags: ", delinq, appr, depr)
    print("Neighbors: ", neighbors)

    return suggestionNumber

```

Demo: Flask Routes from Index.py

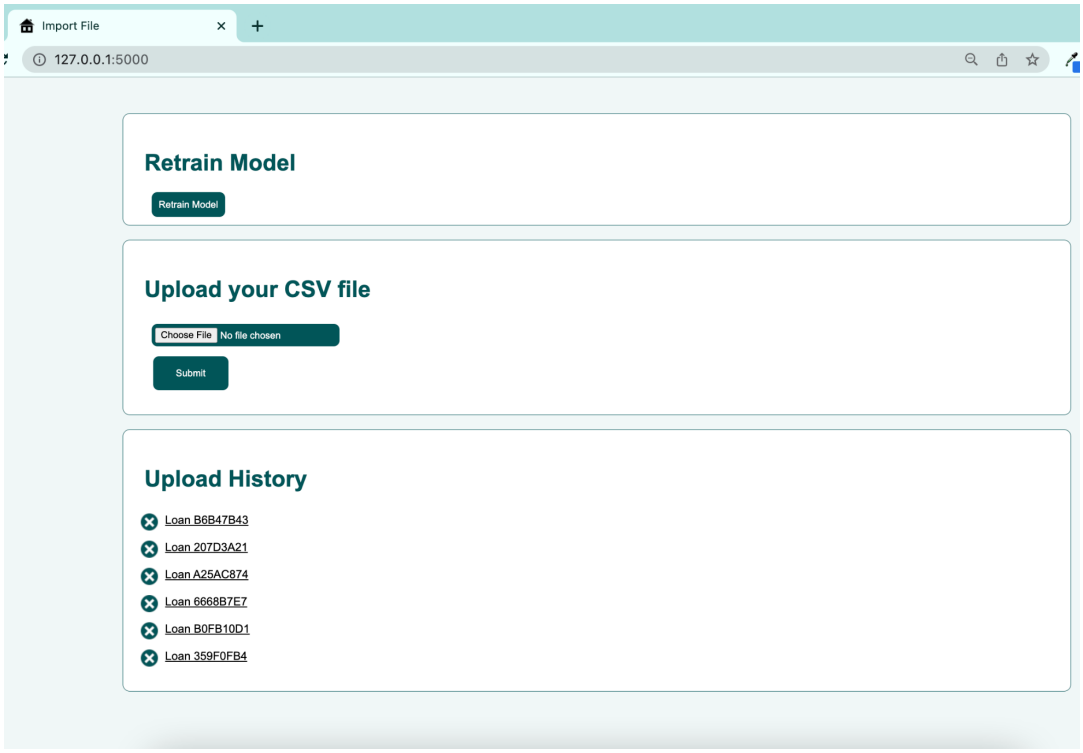
```python
# Home Page
@app.route('/')
def index():
    return render_template('index.html', filenames=get_filenames())


# Return File History
@app.get("/")
def get_filenames():
    files = os.listdir(FILES_PATH)
    return files
```

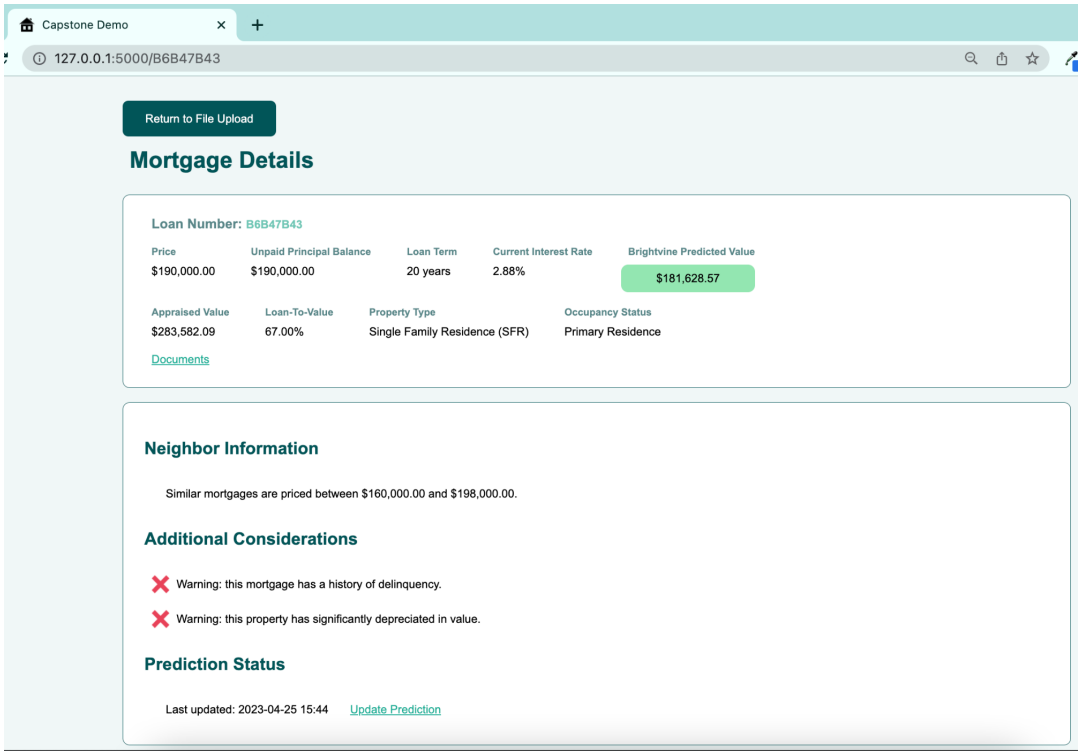
```python
# Upload Files
@app.post("/")
def uploadFiles():
    pattern = re.compile(r'.*\.csv$')
    uploaded_file = request.files['file']
    if pattern.match(uploaded_file.filename):
        loanID = str(uuid.uuid4()).split('-')[0].upper()
        file_path = os.path.join(FILES_PATH, loanID+'.csv')
        uploaded_file.save(file_path)
    else:
        flash("Please select a valid csv file before submitting.")
        return redirect('/')
    return redirect('/' + loanID + '/load')


# Train Model
@app.get('/train')
def train():
    trainingModel.trainingClustersDriver()
    return redirect('/')
```

```python
# Generate Predicted Value
@app.get('/<loanID>/predict')
def predict(loanID):
    time.sleep(2)
    file_name = loanID + '.csv'
    file_path = os.path.join(FILES_PATH, file_name)
    df = pd.read_csv(file_path)
    df['PPP'], df['delinq'], df['appr'], df['depr'], neighbors = predictionModel.testFromUpload(file_name)
    df['loanID'] = loanID
    df['last_updated'] = time.strftime("%Y-%m-%d %H:%M")
    df['neighbor_min'] = neighbors.min()
    df['neighbor_max'] = neighbors.max()
    df.to_csv(file_path)
    return jsonify("Prediction Complete!")
```

```python
# View Specified Mortgage
@app.get('/<loanID>')
def mortgageDetails(loanID):
    file_name = loanID + '.csv'
    file_path = os.path.join(FILES_PATH, file_name)
    df = pd.read_csv(file_path)
    data = df.to_dict('records')[0]
    for item in ['Price', 'UPBatAcquisition', 'PPP', 'OriginationValue', 'PropertyValue', 'CurrentPropertyValue',
                 'neighbor_min', 'neighbor_max']:
        if 'PPP' not in data.keys():
            return redirect('/' + file_name + '/load')
        if not isinstance(data[item], str):
            data[item] = "${:,.2f}".format(data[item])
    for item in ['InterestRate', 'LTVRatio']:
        if not isinstance(data[item], str):
            data[item] = "{:.2%}".format(data[item] / 100)
    return render_template('mortgageDetails.html', data=data, file_name=file_name)

# Delete Specified File
@app.route("/<loanID>/delete")
def delete_file(loanID):
    file_name = loanID + '.csv'
    os.remove(os.path.join(FILES_PATH, file_name))
    return redirect('/')
```

Demo: Running Screenshots





References

[1] *What Types of Mortgages Do Fannie Mae and Freddie Mac Acquire?* Federal Housing Finance Agency. (2021, April 14). Retrieved November 29, 2022, from https://www.fhfa.gov/Media/Blog/Pages/What-Types-of-Mortgages-Do-Fannie-Mae-and-Freddie-Mac-Acquire.aspx.

[2] IBM Cloud Education. (2020, July 15). *What is machine learning?* Retrieved November 30, 2022, from https://www.ibm.com/cloud/learn/machine-learning.

[3] Sewell, M. (n.d.). *No Free Lunch Theorems*. Retrieved from http://www.no-free-lunch.org/.

[4] Akindaini, Bolarinwa. (2017). *Machine learning applications in mortgage default prediction.* University of Tampere.

[5] Lai, L. (2020). *Loan Default Prediction with Machine Learning Techniques.* 2020 International Conference on Computer Communication and Network Security (CCNS).

[6] Niu, Jiafei. (2019). *An intelligent automatic valuation system for real estate based on machine learning.* In Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing (AIIPCC '19). Association for Computing Machinery, New York, NY, USA, Article 12, 1–6. https://doi.org/10.1145/3371425.3371454.

[7] Chen, C. (2020, December 27). *Machine Learning Design Patterns: Reproducibility*. Github. Retrieved November 11, 2022, from https://changyaochen.github.io/ML-design-pattern-1/.

[8] Douglas, S. (2019, September 5). *The Pareto Principle and UX*. Usability Geek. Retrieved November 11, 2022, from https://usabilitygeek.com/pareto-principle-and-ux/.

[9] Kolamanvitha. (2021, July 19). *Design Patterns for Machine Learning*. Medium. Retrieved November 11, 2022, from https://towardsdatascience.com/design-patterns-for-machine-learning-410be845c0db.

[10] Hartman, J. (2022, July 28). *Joe Vellanikaran of Brightvine on 5 things you need to succeed in the Modern World of Finance &...* Medium. Retrieved from https://medium.com/authority-magazine/joe-vellanikaran-of-brightvine-on-5-things-you-need-to-succeed-in-the-modern-world-of-finance-5c4eed060bc8.

[11] *About Us*. (2022) Brightvine. Retrieved November 11, 2022, from https://www.brightvine.com/about-us.