Uncovering Patterns in the LendingClub Dataset: An Exploration of Unsupervised and Supervised Techniques

A Team Project Report for AI 221

Hans Jarett Ong
University of the Philippines Diliman
hjong@up.edu.ph

Rossjyn Fallorina
University of the Philippines Diliman
rcfallorina1@up.edu.ph

1. INTRODUCTION

The LendingClub is a US-based tech company that offers a peer-to-peer lending platform. The company was founded in 2006 and had its IPO in 2014 – the largest tech IPO of that year. Their core business model, the peer-to-peer lending platform, facilitates unsecured personal loans ranging from $1,000 to $40,000.

The LendingClub's primary value proposition is in connecting borrowers and lenders. With their platform, lenders, or investors, could browse through a catalogue of loan listings containing essential information such as loan amount, interest rate, and loan purpose. This allows investors to easily select the loans they want to finance. For this, investors earn returns through the interest generated by the loans, while LendingClub generates its revenue from the transaction fees.

However, in December 2020, the LendingClub made a strategic decision to pivot away from the peer-to-peer lending model and transition towards more traditional financial services. The specific reasons and motivations behind this strategic shift are beyond the scope of this report. Consequently, the dataset available for analysis in this study is limited to information collected until the end of 2020 [1].

2. OBJECTIVES

The objective of this report is to explore the LendingClub dataset using unsupervised and supervised learning techniques. We acknowledge that the scope of this study is limited, and the results, in its current stage, still cannot be used to guide decisions. This project serves as an initial iteration aimed at showcasing the potential of the dataset and outlining avenues for further research.

We aim to use unsupervised learning techniques, including dimensionality reduction, anomaly detection, and clustering, to visualize and identify patterns within the dataset. Additionally, we use supervised learning techniques, i.e. predictive modeling, to predict variables of interest to the business, such as the likelihood of delinquency and its associated factors. The specifics of these techniques are discussed in the next section.

3. PROPOSED METHODOLOGY

The dataset used in this study consists of the accepted loans dataset, covering loan data from 2007 to the third quarter of 2020. It comprises approximately 2.9 million rows and 142 columns making it a very rich dataset [2].

The features within the dataset can be broadly categorized into the following groups:

1. *Borrower Information.* This includes information about the borrowers such as their annual income, employment, home ownership, state of residence. These features capture demographic and socioeconomic aspects of the borrowers.
2. *Loan Details.* These include attributes related to the specific loans, including the loan amount, term (ranging from 3 to 5 years), interest rate, purpose of the loan, and installment amount.
3. *Credit History.* This comprises factors related to the credit history of borrowers such as their FICO scores (a credit scoring system used in the US), number of delinquencies in the past, and the number of credit lines. These variables offer insights into the creditworthiness and financial track record of borrowers.
4. *Loan Payment Status.* This category focuses on the payment and financial aspects of the loans. It includes information such as the loan status (e.g., fully paid, charged off), total payments made by borrowers, and the remaining principal amount. These variables provide an overview of the payment behavior and the current financial status of borrowers.

3.1 Data Pre-processing

Prior to applying the different learning techniques, we first had to pre-process the data to make it suitable for modeling. The pre-processing steps are detailed below:

1. *Missing Values Handling.*
   - Drop columns with more than 90% missing values: Columns with predominantly missing values provide limited information and may introduce low-variance related issues when used in models. Hence, we dropped any columns with more than 90% missing values.
   - Drop rows with missing values: After dropping the columns with mostly missing values, rows containing missing values were also dropped from the dataset. While alternative ways to handle missing values exist, such as imputation, we chose to simply drop these rows given that the dataset is large enough.

2. *Feature Engineering.* The aim of feature engineering is to create new features that are more informative or are more readily useful for the models. For this dataset, the following feature engineering techniques were applied:

    - Extract the year and month from the issue date column: The issue date column, which represents the loan issue date, was converted from string to the date-time format. Then, we extracted the year and month from this column as separate features. This transformation allows us to analyze the temporal patterns and trends in the data more effectively.
    - Convert string columns to numeric: Certain columns, such as the interest rate, loan term, and employment length were initially encoded in string format. We converted these columns to numeric to make them compatible with the models.
    - Make other dates relative to issue date: The last payment date is another date feature we had to handle. To make it comparable across the different customers, we calculated the time difference between the last payment date and the issue date. This transformation provided us with a relative measure, enabling us to analyze the timing of payments with respect to the loan issue date.

3. *Categorical Encoding.* Categorical features such as state, loan purpose, and grade were encoded using the one-hot encoding method. This process converts each categorical variable into multiple binary variables, enabling us to represent these categories numerically.

4. *Dropping Unnecessary Columns.* Columns that do not add relevant information to this study, such as URL and zip code, were dropped.

5. *Removing Highly Correlated Columns.* To avoid rank deficiencies and multicollinearity issues, we identified and removed highly correlated columns. Highly correlated variables provide redundant information and can negatively impact the performance of certain models.

3.2 Unsupervised Learning

In the unsupervised learning component of our study, we explored three techniques: dimensionality reduction, anomaly detection, and clustering. This section discusses the specific methods tried for each technique.

1. *Dimensionality Reduction.* We experimented using three common methods for dimensionality reduction: t-SNE (t-Distributed Stochastic Neighbor Embedding), Isomap, and PCA (Principal Component Analysis). Due to computational constraints, we only used a sample size of 2,500 instances for these methods. This is particularly true for t-SNE, which is not well-suited for large datasets. This sampling approach is justifiable since our primary objective in dimensionality reduction was to gain initial insights through data visualization, rather than an extensive analysis of the complete dataset.

2. *Anomaly Detection.* To identify outliers, we tried three commonly used anomaly detection methods: Isolation Forest, Local Outlier Factor, and Support Vector Machines (SVM). We then selected the best method for outlier detection based on visual inspection of the results of each method.

3. *Clustering.* In the clustering analysis, we used dimensionally reduced data obtained from PCA, considering only the top few principal components. Subsequently, we applied two widely adopted clustering methods, namely k-means and agglomerative clustering. For the k-means clustering, we used the elbow or scree plot in conjunction with calculating silhouette scores to determine the optimal number of clusters. Meanwhile, for the agglomerative clustering, we relied on visual inspection of the dendrogram to determine the most suitable clustering metric and cluster count.

The aim of these unsupervised learning techniques is to uncover patterns, detect anomalies, and identify potential clusters or segments within the dataset. The result of these methods will be presented in the Results and Discussion section.

3.3 Supervised Learning

The latter component of the study dealt with performing supervised learning on the LendingClub dataset. In particular, we aimed to take a closer look at loan delinquencies and how well supervised models could predict these instances for the LendingClub's loan profiles.

1. *Target Variable.* In the LendingClub dataset, a column is included to indicate the number of counts/instances a profile was tagged to be *delinquent* in the past 2 years. Delinquency of a borrower's credit profile is defined to be exceeding 30 days of non-payment from the file's due date. For simplicity, profiles that had at least one delinquency count in the past 2 years were flagged as 1 in the target variable. Conversely, files that have not been delinquent in the past 2 years were tagged as 0. As the target variable is binary in nature, the supervised learning problem takes the form of a binary classification task.

2. *Feature Selection.* After the target variable has been defined, features to be used for the modeling were selected. As this study hoped to generate benchmark results, feature selection was performed mainly through intuition and domain knowledge by selecting commonly-understood and interpretable information. The selected features and target variable are all detailed in Table 1.

3. *Binary Classification.* Since predicting the derived loan delinquency flag is essentially a binary classification task, there are a handful of supervised learning models to choose from. For this study, five binary classification models were studied, namely: Logistic Regression, Ridge Classifier, Linear Discriminant Analysis, Random Forest Classifier, and Extreme Gradient Boosting.

This section of the study was accomplished with the help of PyCaret, an open-source, low-code Python machine learning library that automates machine learning processes and workflows. As mentioned earlier, five models were chosen as the supervised models for the binary classification problem ("xgboost", "rf", "lr", "ridge", and "lda"). Prior to feeding to the AutoML pipeline, the features of the dataset were subject to minor pre-processing, mainly merging minority classes into larger classes. Since the prevalence of delinquencies in the data was around 18%, the target variable was considered to be imbalanced. Class imbalance was remedied using the SMOTE oversampling technique. Categorical features were also one-hot encoded, and numerical columns were normalized using MinMax normalization. The resulting dataframe was divided into 70% training and 30% testing data, the former being used in training the supervised models. Stratified 10-fold cross-validation was used for performance evaluation of the models, with the main metrics used for model selection being the models' accuracies and ROC-AUC scores.

4. RESULTS AND DISCUSSION

In this section, we present and discuss the results obtained from the aforementioned methods.

4.1 Unsupervised Learning

Dimensionality Reduction

Figure 1 illustrates the two-dimensional representation of the data using various dimensionality reduction techniques, namely t-SNE, Isomap, and PCA. These methods were used to visualize the dataset. To gain further contextual understanding, we explored potential patterns by assigning different colors to the data points based on specific variables. Notably, discernible patterns emerged when categorizing the data by risk grade (Fig. 1a), loan status (Fig. 1b), and unexpectedly, by state (Fig. 1c).

The patterns related to risk grade and loan status were consistently observed across all dimensionality reduction approaches. Particularly, PCA exhibited the most striking patterns in its second principal component (PC2), which effectively captured the gradient of loan risk. Higher PC2 values corresponded to loans with elevated risk scores and unfavorable loan statuses, such as write-offs or late payments. Moreover, the state variable exhibited discernible patterns in the t-SNE visualization, where small clusters of data points with the same state were observed.

Anomaly Detection

Figure 2 displays the results of anomaly detection using Isolation Forest, Local Outlier Factor, and Support Vector Machines. For visualization purposes, the results are presented in their first two principal components. Upon visual inspection, it is evident that the Local Outlier Factor method outperformed the other two approaches by accurately identifying the obvious outliers. In contrast, the remaining two methods exhibited some false positive results.

Clustering

Clustering techniques were used to identify natural groupings within the loan dataset. This segmentation can potentially aid the company in developing personalized and targeted approaches for different client segments. In this study, two clustering techniques, namely k-means clustering and agglomerative clustering, were explored. For both techniques, only the first four principal components of the preprocessed dataset were used, which collectively accounted for approximately 98% of the variance.

Figure 3 shows the k-means clustering technique. To determine the optimal number of clusters, we used the silhouette plot (Figure 3a). Based on this analysis, the best choice was determined to be $k = 4$. The resulting clustering, obtained with this number of clusters, is displayed in Figure 3b.

On the other hand, agglomerative clustering using Ward's method as the distance metric was also employed. The dendrogram shown in Figure 4a was utilized to determine the ideal number of clusters, which was found to be $k = 3$. The resulting clustering is depicted in Figure 4b. Although other distance metrics such as centroid and average linkages were considered (not shown here), Ward's method was selected based on visual inspection of the dendrograms. Ward's method provided the most balanced groupings compared to the other methods, which failed provide meaningful segmentation.

Overall, the results obtained from the dimensionality reduction, anomaly detection, and clustering techniques provide valuable insights into the structure, anomalies, and distinct groups present within the dataset. These findings can serve as a foundation for further analysis and applications such as as risk assessment, fraud detection, and customer segmentation.

4.2 Supervised Learning

The performance of the five binary classifiers on predicting loan delinquency in the LendingClub dataset are summarized in Figure 5. We find that XGBoost performed best among all models, having an accuracy of 0.8158 and ROC-AUC score of 0.6238. Using the same metrics as basis, the random forest classifier is the next best performing model. For both XGBoost and Random Forest, their biggest disadvantage is their significantly longer training times. The three remaining classifiers, Logistic Regression, Ridge Regression, and Linear Discriminant Analysis, showed low to moderate performance across all metrics. However, these three models possessed shorter training times compared to the other, tree-based algorithms. For business use-cases, although the XGBoost classifier has a relatively high training time compared to most models, it more than makes up for it with its superior accuracy and ROC-AUC metrics scores.

While we arrive at XGBoost being the best model in terms of accuracy and ROC-AUC score, it is far from being perfect. After fitting the model once more with the training data, the trained model showed only minimal improvements for the two metrics. A visual representation of the ROC curve for the XGBoost classifier is given by Figure 6a. The confusion matrix in Figure 6b shows the counts of predicted classes against the true delinquent cases. The model is shown to extremely under-predict true delinquents, resulting in a low recall value.

Lastly, feature importance was calculated for the classifier to

Variable Type	Name	Description	Data Type
Feature	loan_amnt	The listed amount of the loan applied for by the borrower	Integer
Feature	term	The number of payments on the loan	Categorical
Feature	int_rate	Interest Rate on the loan	Numeric
Feature	grade	LendingClub assigned loan grade	Categorical
Feature	home_ownership	The home ownership status provided by the borrower during registration or obtained from the credit report	Categorical
Feature	annual_inc	The self-reported annual income provided by the borrower during registration	Numeric
Feature	purpose	A category provided by the borrower for the loan request	Categorical
Feature	emp_length	Employment length in years	Integer
Target	flag_delinq2yrs	Having at least one delinquency (>30 days non-payment past due date) count in the past 2 years	Flag

Table 1: The target variable and features used for supervised learning.

determine the most significant features that improve predictive performance. The one-hot encoded LendingClub grades A and C were shown to have high variable importances among the training features. Profiles with owners that mortgaged or rented their homes also significantly drove model performance. These results offer potential insights on how well LendingClub assigned loan grades and even home ownership types are able to predict the likelihood of a loan profile to have an instance of delinquency.

5. CONCLUSION

In conclusion, this project explored the LendingClub dataset using unsupervised and supervised learning techniques. Data preprocessing and feature engineering techniques were applied to clean and transform the dataset, ensuring the quality and reliability of the analysis.

Unsupervised learning techniques, such as dimensionality reduction, clustering, and anomaly detection, provided valuable insights into the underlying patterns and anomalies within the dataset. These techniques allowed for visualizations and a deeper understanding of the data structure.

Supervised learning methods were tested to predict loan delinquencies using features extracted from the loan profiles in the dataset. With the aid of the AutoML library PyCaret, binary classification on delinquent loan profiles was studied using 5 supervised learning models. The XGBoost classifier was shown to have the highest accuracy and ROC-AUC scores compared to 4 other binary classifiers, albeit with a relatively high training time. Additionally, LendingClub loan grades and home ownership type were seen to have significant impact in driving model performance.

6. RECOMMENDATIONS

This project serves as an exploratory analysis, and there are several avenues for further research and improvement. Here are some recommendations for future work:

1. *Further Feature Engineering*: Explore additional feature engineering techniques to enhance the predictive power of the models. Consider incorporating external data sources to capture more nuanced information about borrowers and loan characteristics.

2. *Feature Selection*: Perform rigorous feature selection to identify the most influential features for loan delinquency prediction. Utilize techniques such as feature importance, correlation analysis, or step-wise selection to identify the most relevant features and eliminate any redundant or irrelevant ones.

3. *Hyperparameter Tuning and Model Optimization*: Conduct thorough hyperparameter tuning and model optimization to improve the precision and overall performance of the predictive models. Utilize techniques like grid search, random search, or Bayesian optimization to find the optimal set of hyperparameters for each model.

4. *Deeper Analysis*: Analyze specific loan characteristics, borrower profiles, or economic factors to gain deeper insights into loan delinquency patterns. Investigate how factors such as loan amount, interest rate, loan

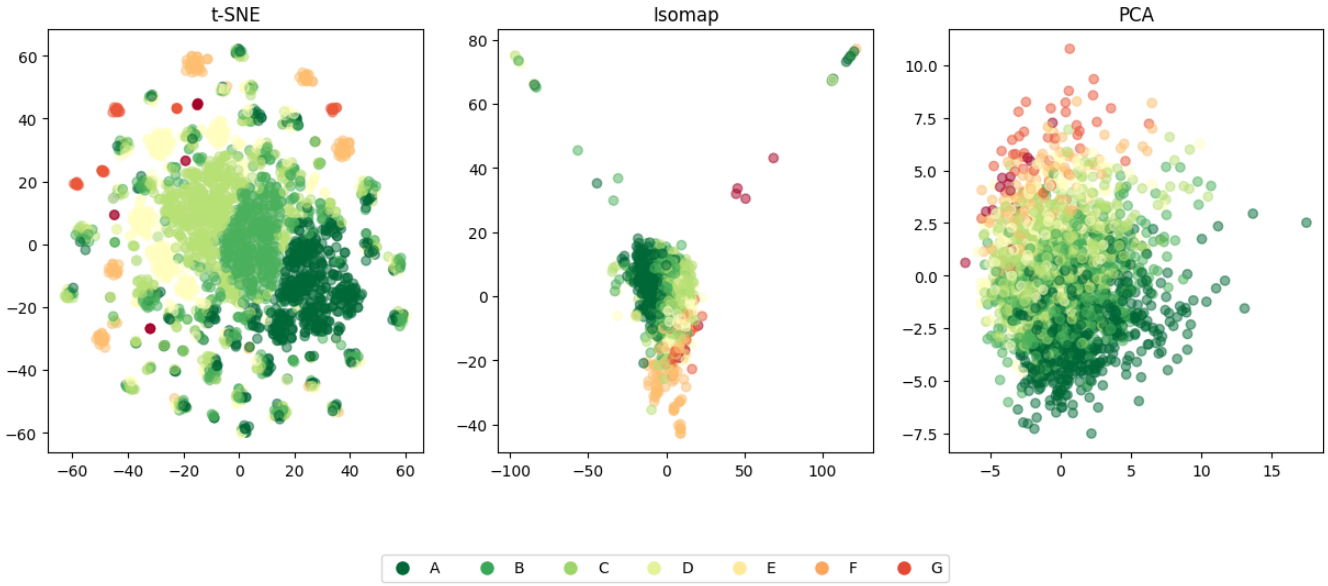
purpose, borrower income, credit history, and economic indicators impact loan delinquencies. This analysis can provide valuable insights for risk assessment and decision-making.

Overall, this initial exploration of the LendingClub dataset demonstrates its potential for further analysis and refinement. By applying advanced machine learning techniques and conducting deeper investigations into loan characteristics and borrower profiles, we can gain valuable insights into loan delinquency patterns and improve risk assessment in the lending industry.

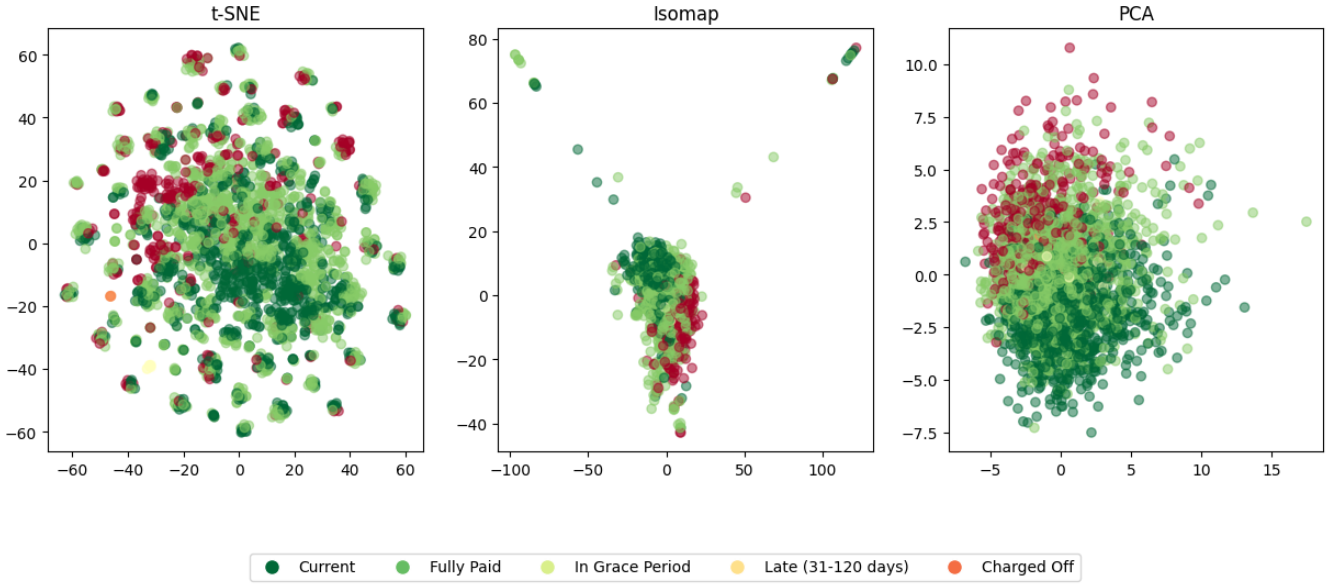
7. REFERENCES

[1] "Lendingclub," https://en.wikipedia.org/wiki/LendingClub, Accessed 2023, wikipedia Article.

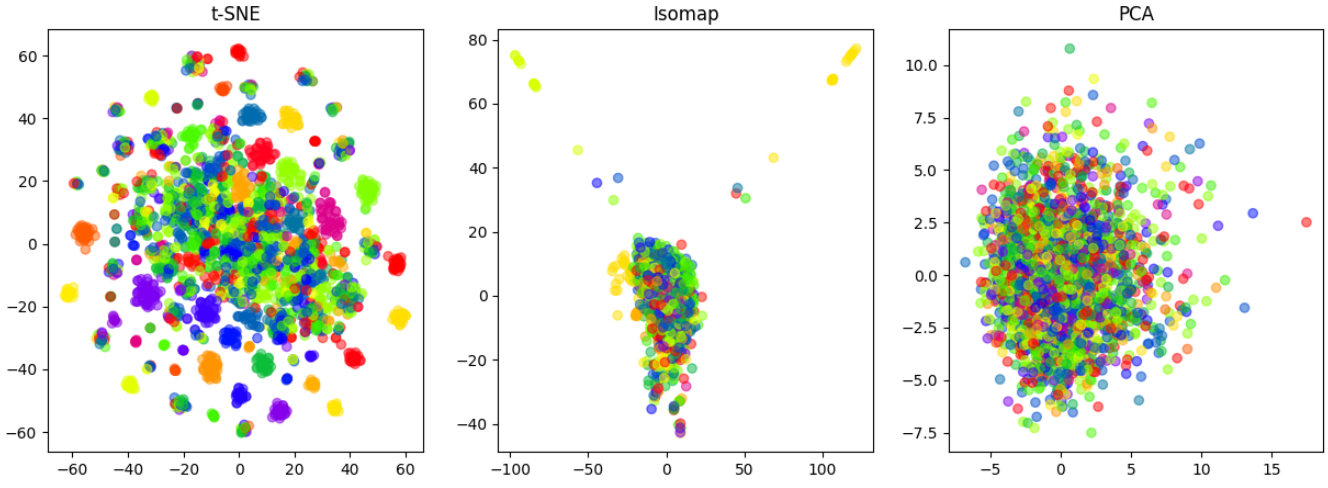
[2] Lending Club, "Lending club 2007-2020q3," https://www.kaggle.com/datasets/ethon0426/lending-club-20072020q1, 2020, kaggle Dataset.



(a) t-SNE colored by risk grade



(b) Isomap colored by loan status



(c) PCA colored by state

Figure 1: Dimensionality reduction techniques colored by risk grade, loan status, and state. The figures show the results from t-SNE (a), Isomap (b), and PCA (c), respectively.

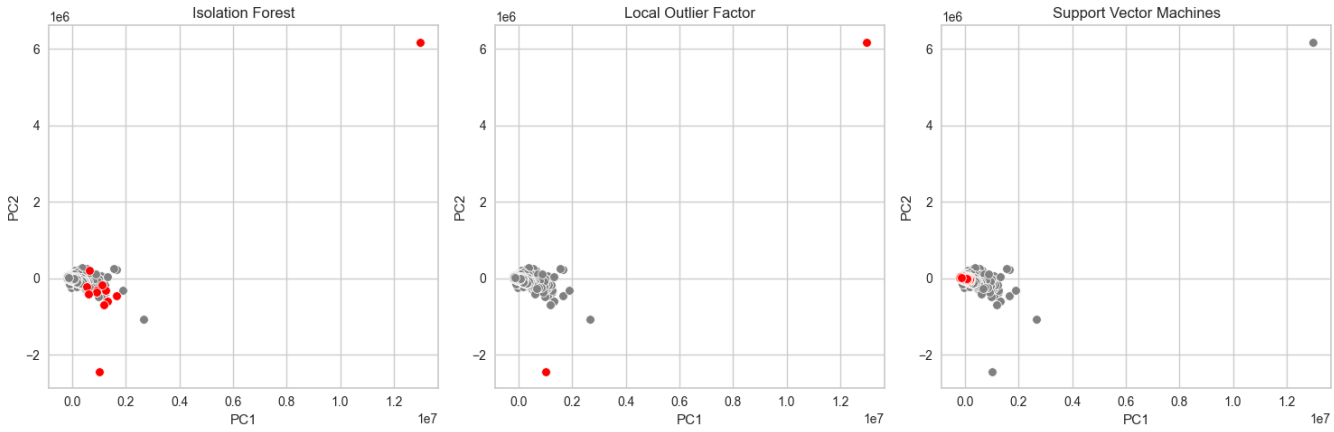
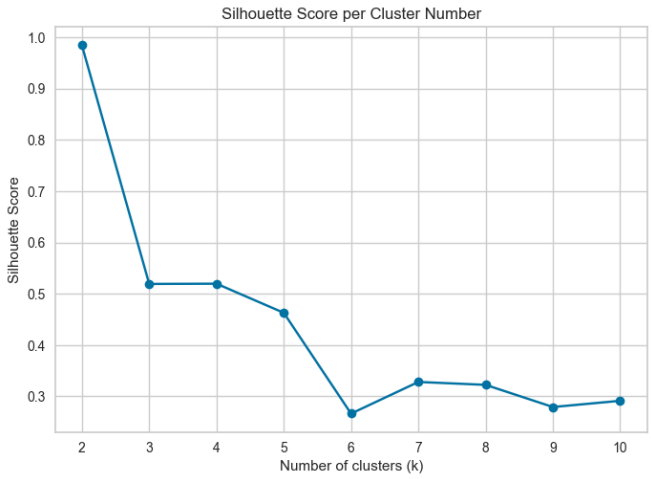
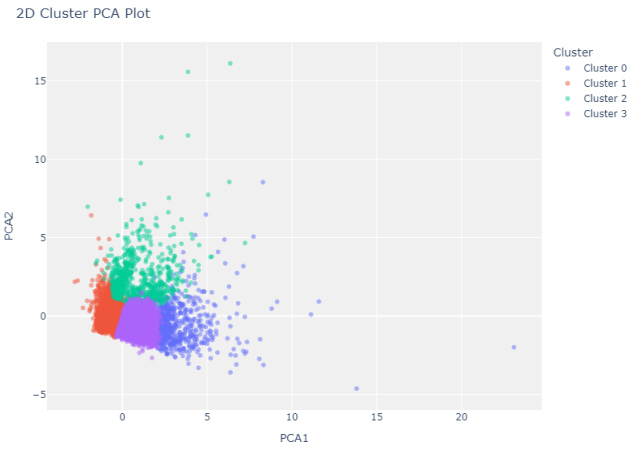


Figure 2: Results of Anomaly Detection using Isolation Forest, Local Outlier Factor, and Support Vector Machines

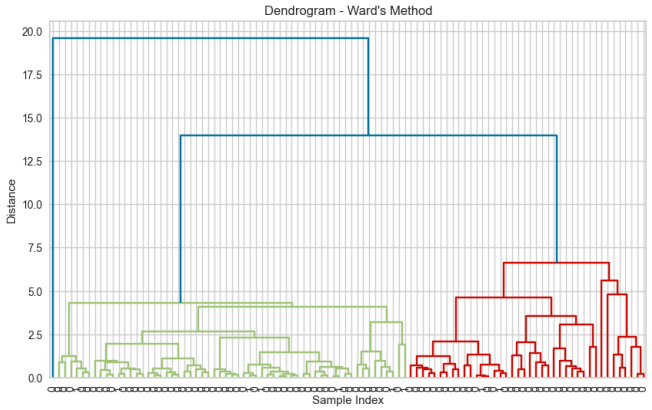


(a) Silhouette plot for determining the optimal number of clusters

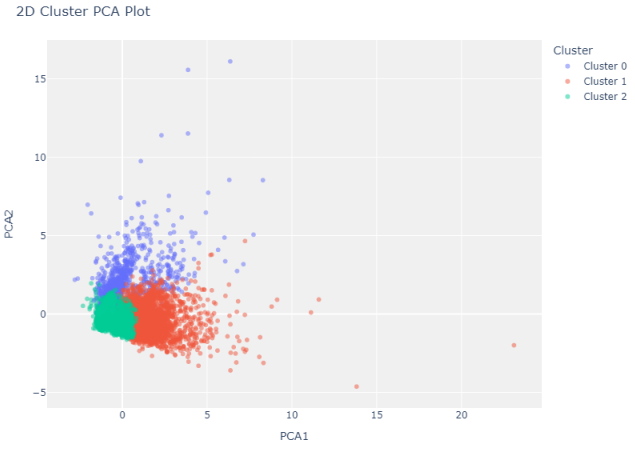


(b) K-means clustering results with $k = 4$

Figure 3: Results of K-means Clustering. The optimal number of clusters is $k = 4$, as shown in Figure 3b.



(a) Dendrogram for determining the best number of clusters

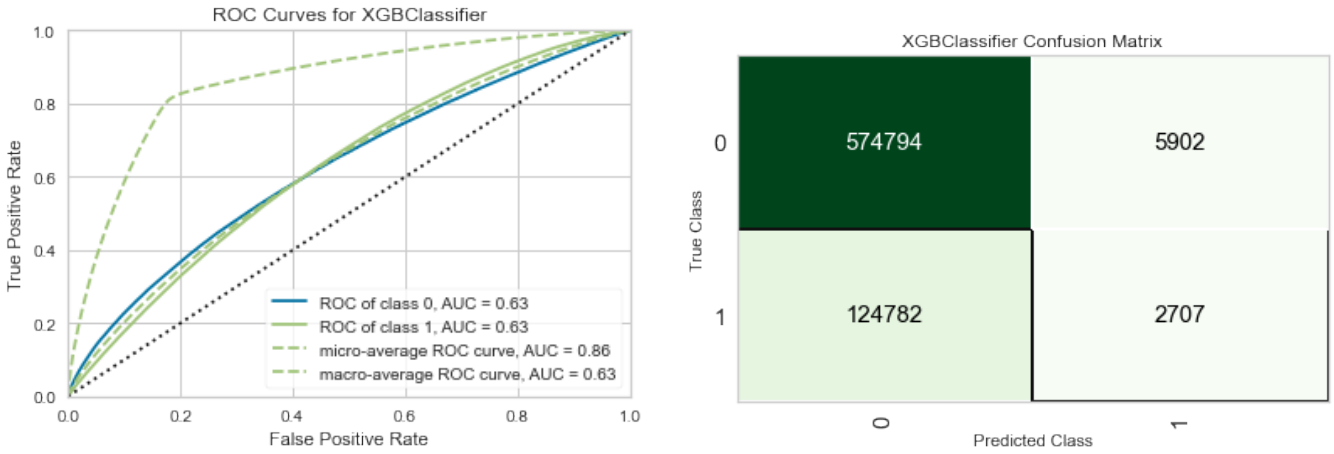


(b) Agglomerative clustering results with $k = 3$

Figure 4: Results of Agglomerative Clustering. The best number of clusters is $k = 3$, as shown in Figure 4b.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
xgboost	Extreme Gradient Boosting	0.8158	0.6238	0.0188	0.3076	0.0354	0.0150	0.0350	154.2620
rf	Random Forest Classifier	0.7791	0.5722	0.1002	0.2343	0.1404	0.0365	0.0408	256.1000
lr	Logistic Regression	0.5633	0.6127	0.6073	0.2300	0.3336	0.0981	0.1238	74.2810
ridge	Ridge Classifier	0.5583	0.0000	0.6058	0.2273	0.3306	0.0931	0.1182	42.2630
lda	Linear Discriminant Analysis	0.5583	0.6073	0.6056	0.2273	0.3305	0.0931	0.1181	39.1790

Figure 5: Performance metrics of the 5 binary classification models.



(a) ROC-AUC curve of the XGBoost classifier

(b) Confusion matrix of the XGBoost classifier

Figure 6: Model evaluation plots of the XGBoost classifier fit with training data.

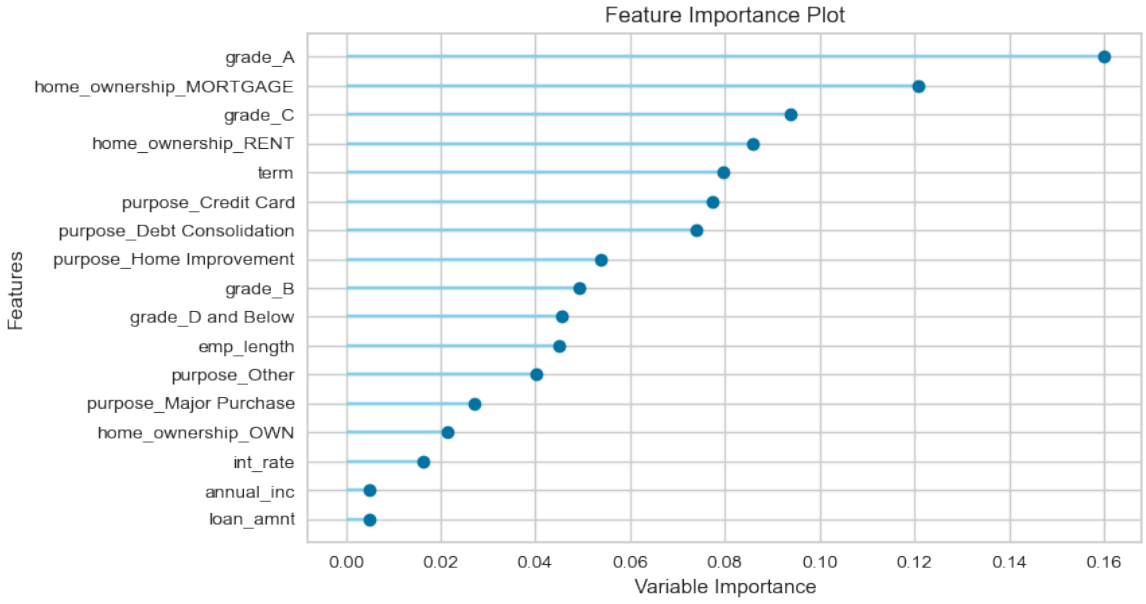


Figure 7: Feature importance plot of the training features.

Appendix A - Preprocessing

June 24, 2023

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
accepted_df = pd.read_csv("data/accepted_2007_to_2020.csv", index_col=0)
# rejected_df = pd.read_csv("data/rejected_2007_to_2018Q4.csv")
```

1 Accepted Loans

Note: Looking at columns with the same number of nulls gives us an idea of which variables were "generated" together.

In [ ]:
```
accepted_df = accepted_df[~accepted_df.loan_amnt.isna()]
```

In [ ]:
```
df = accepted_df.set_index("id")

# Handle missing values
df.dropna(
    axis=1, thresh=len(df) * 0.9, inplace=True
)  # Drop columns with more than 90% missing values
df.dropna(inplace=True)  # Drop rows with any missing values

# Feature engineering
df["issue_d"] = pd.to_datetime(df["issue_d"])  # Convert issue date to datetime
df["year"] = df["issue_d"].dt.year  # Extract year from issue date
df["month"] = df["issue_d"].dt.month  # Extract month from issue date


# Convert int_rate to numerical
df["int_rate"] = df["int_rate"].str.rstrip("%").astype("float") / 100.0
df["term"] = df["term"].apply(lambda x: int(x.split()[0]))
df["emp_length"] = df["emp_length"].str.extract(r"(\d+)")
df["emp_length"] = pd.to_numeric(df["emp_length"], errors="coerce")
# Feature engineering - Extract year and month from issue date
df["issue_d"] = pd.to_datetime(df["issue_d"])
df["issue_year"] = df["issue_d"].dt.year
```

```python
# Calculate credit history length
df["earliest_cr_line"] = pd.to_datetime(df["earliest_cr_line"])
df["credit_history_length"] = df["issue_year"] - df["earliest_cr_line"].dt.year
# Convert revol_util to numeric
df["revol_util"] = df["revol_util"].str.rstrip("%").astype("float") / 100.0

# Calculate the difference between last payment and issue date in years
df["last_pymnt_d"] = pd.to_datetime(df["last_pymnt_d"])
df["last_pymnt_issue_diff"] = (df["last_pymnt_d"] - df["issue_d"]).dt.days // 365

# Calculate the difference between last credit pull and issue date in years
df["last_credit_pull_d"] = pd.to_datetime(df["last_credit_pull_d"])
df["last_credit_pull_issue_diff"] = (
    df["last_credit_pull_d"] - df["issue_d"]
).dt.days // 365

# Convert "debt_settlement_flag" and "hardship_flag" to numeric
df["debt_settlement_flag"] = (df["debt_settlement_flag"] == "Y").astype(int)
df["hardship_flag"] = (df["hardship_flag"] == "Y").astype(int)


# I made issue year categorical to capture trends, e.g. loans back in 2008 might behav
categorical_cols = [
    "grade",
    "sub_grade",
    "home_ownership",
    "verification_status",
    "purpose",
    "addr_state",
    "initial_list_status",
    "loan_status",
    "issue_year",
    "application_type",
]
df = pd.get_dummies(df, columns=categorical_cols)


columns_to_drop = [
    "emp_title",
    "url",
    "title",
    "zip_code",
    "issue_d",
    "last_pymnt_d",
    "last_credit_pull_d",
    "pymnt_plan",
    "earliest_cr_line",
]
```

```
        df.drop(columns=columns_to_drop, inplace=True)
```

In [ ]:
```
# check that everything is numeric
df.dtypes.value_counts()
```

In [ ]:
```
# check that there are no null values
df.isna().sum().value_counts()
```

In [ ]:
```
df.shape
```

In [ ]:
```
df.to_pickle("data/preprocessed_df.pickle")
```

Appendix B - Unsupervised Learning

June 24, 2023

```
In [ ]: import os
        import pickle

        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        from tqdm import tqdm

        import pycaret
        pycaret.__version__

        from sklearn.preprocessing import StandardScaler, LabelEncoder
        from sklearn.decomposition import PCA, KernelPCA
        from sklearn.manifold import TSNE, Isomap
        from sklearn.metrics import silhouette_score
```

```
In [ ]: df = pd.read_pickle("data/df_filtered.pickle")
        normed_df = pd.DataFrame(
            StandardScaler().fit_transform(df), columns=df.columns, index=df.index
        )
```

```
In [ ]: df_raw = pd.read_pickle("data/accepted_df_raw.pickle")
```

Methods to condsider: - PCA - Kernel PCA - Locally Linear Embedding (LLE) - t-SNE - K-Means - Hierarchical Clustering - DBSCAN - Gaussain Mixture Models - Kernel Density Estimation (KDE) - One-Class SVM

1 Dimensionality Reduction

```
In [ ]: # Original code to obtain transformed data
        df_sample = normed_df.sample(2500, random_state=2023)

        tsne = TSNE(n_components=2, perplexity=30, random_state=42)
        isomap = Isomap(n_components=2, n_neighbors=10)
        pca = PCA(n_components=2)
```

```
df_tsne = tsne.fit_transform(df_sample)
df_isomap = isomap.fit_transform(df_sample)
df_pca = pca.fit_transform(df_sample)

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

axs[0].scatter(df_tsne[:, 0], df_tsne[:, 1], color="black")
axs[0].set_title("t-SNE")

axs[1].scatter(df_isomap[:, 0], df_isomap[:, 1], color="black")
axs[1].set_title("Isomap")

axs[2].scatter(df_pca[:, 0], df_pca[:, 1], color="black")
axs[2].set_title("PCA")

plt.tight_layout()
plt.show()
```

In [ ]:
```
columns_of_interest = ["grade", "loan_status", "addr_state"]
df_sample = df_sample.join(df_raw[columns_of_interest])
```

In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib as mpl
# Convert the categorical grades to numerical labels
label_encoder = LabelEncoder()
encoded_grades = label_encoder.fit_transform(df_sample["grade"])
alpha = 0.5

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

# Reverse the color map
cmap = plt.cm.RdYlGn
reversed_cmap = cmap.reversed()

# Scatter plot for t-SNE
scatter_tsne = axs[0].scatter(
    df_tsne[:, 0], df_tsne[:, 1], c=encoded_grades, cmap=reversed_cmap, alpha=alpha
)
axs[0].set_title("t-SNE")

# Scatter plot for Isomap
scatter_isomap = axs[1].scatter(
    df_isomap[:, 0], df_isomap[:, 1], c=encoded_grades, cmap=reversed_cmap, alpha=alpha
)
axs[1].set_title("Isomap")

# Scatter plot for PCA
scatter_pca = axs[2].scatter(
```

```
        df_pca[:, 0], df_pca[:, 1], c=encoded_grades, cmap=reversed_cmap, alpha=alpha
    )
    axs[2].set_title("PCA")

    # Create a legend
    legend_labels = label_encoder.classes_
    legend_elements = [plt.Line2D([0], [0], marker='o', color='w', label=label,
                                  markerfacecolor=reversed_cmap(i / len(legend_labels)), ma
                       enumerate(legend_labels)]
    fig.legend(legend_elements, legend_labels, loc='center', bbox_to_anchor=(0.5, -0.1),
               ncol=len(legend_labels))

    plt.show()
```

In [ ]:
```
    import matplotlib.pyplot as plt
    import matplotlib as mpl

    # Convert the categorical grades to numerical labels
    label_encoder = LabelEncoder()
    # label_encoder.classes_ = np.append(
    #     label_encoder.classes_[1:], label_encoder.classes_[0]
    # )
    encoded_grades = (label_encoder.fit_transform(df_sample["loan_status"]) - 1) % 5
    alpha = 0.5

    fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

    # Reverse the color map
    cmap = plt.cm.RdYlGn
    reversed_cmap = cmap.reversed()

    # Scatter plot for t-SNE
    scatter_tsne = axs[0].scatter(
        df_tsne[:, 0], df_tsne[:, 1], c=encoded_grades, cmap=reversed_cmap, alpha=alpha
    )
    axs[0].set_title("t-SNE")

    # Scatter plot for Isomap
    scatter_isomap = axs[1].scatter(
        df_isomap[:, 0], df_isomap[:, 1], c=encoded_grades, cmap=reversed_cmap, alpha=alpha
    )
    axs[1].set_title("Isomap")

    # Scatter plot for PCA
    scatter_pca = axs[2].scatter(
        df_pca[:, 0], df_pca[:, 1], c=encoded_grades, cmap=reversed_cmap, alpha=alpha
    )
    axs[2].set_title("PCA")
```

```
        # Create a legend
        legend_labels = np.append(label_encoder.classes_[1:], label_encoder.classes_[0])
        legend_elements = [
            plt.Line2D(
                [0],
                [0],
                marker="o",
                color="w",
                label=label,
                markerfacecolor=reversed_cmap(i / len(legend_labels)),
                markersize=10,
            )
            for i, label in enumerate(legend_labels)
        ]
        fig.legend(
            legend_elements,
            legend_labels,
            loc="center",
            bbox_to_anchor=(0.5, -0.1),
            ncol=len(legend_labels),
        )

        plt.show()
```

```
In [ ]: import matplotlib.pyplot as plt
        import matplotlib as mpl

        # Convert the categorical grades to numerical labels
        label_encoder = LabelEncoder()
        # label_encoder.classes_ = np.append(
        #     label_encoder.classes_[1:], label_encoder.classes_[0]
        # )
        encoded_grades = label_encoder.fit_transform(df_sample["addr_state"])
        alpha = 0.5

        fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

        # Reverse the color map
        cmap = plt.cm.prism
        reversed_cmap = cmap.reversed()

        # Scatter plot for t-SNE
        scatter_tsne = axs[0].scatter(
            df_tsne[:, 0], df_tsne[:, 1], c=encoded_grades, cmap=reversed_cmap, alpha=alpha
        )
        axs[0].set_title("t-SNE")
```

```
        # Scatter plot for Isomap
        scatter_isomap = axs[1].scatter(
            df_isomap[:, 0], df_isomap[:, 1], c=encoded_grades, cmap=reversed_cmap, alpha=alpha
        )
        axs[1].set_title("Isomap")

        # Scatter plot for PCA
        scatter_pca = axs[2].scatter(
            df_pca[:, 0], df_pca[:, 1], c=encoded_grades, cmap=reversed_cmap, alpha=alpha
        )
        axs[2].set_title("PCA")

        # Create a legend
        legend_labels = label_encoder.classes_
        legend_elements = [
            plt.Line2D(
                [0],
                [0],
                marker="o",
                color="w",
                label=label,
                markerfacecolor=reversed_cmap(i / len(legend_labels)),
                markersize=10,
            )
            for i, label in enumerate(legend_labels)
        ]
        # fig.legend(
        #     legend_elements,
        #     legend_labels,
        #     loc="center",
        #     bbox_to_anchor=(0.5, -0.1),
        #     ncol=len(legend_labels),
        # )

        plt.show()
```

In [ ]:
```
df = pd.read_pickle("data/df_filtered.pickle").sample(10000, random_state=2023)

# Perform PCA
pca = PCA(n_components=10)
pca.fit(df)

# Obtain the explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_

# Plot the explained variance ratio
plt.plot(np.cumsum(explained_variance_ratio))
plt.xlabel("Number of Principal Components")
```

```
plt.ylabel("Cumulative Explained Variance")
plt.title("Explained Variance Ratio")

# Determine the optimal number of principal components
cumulative_variance = np.cumsum(explained_variance_ratio)
threshold = 0.95  # Set your desired threshold for explained variance

# Find the index where cumulative variance exceeds the threshold
num_components = np.argmax(cumulative_variance > threshold) + 1

# Perform PCA with the optimal number of components
pca = PCA(n_components=num_components)
df_transformed = pca.fit_transform(df)

# Access the principal components
principal_components = pd.DataFrame(
    data=df_transformed, columns=[f"PC{i}" for i in range(num_components)]
)

# Access the explained variance ratio for each component
component_variance_ratio = pd.DataFrame(
    data=explained_variance_ratio[:num_components], columns=["Explained Variance Ratio
)

# Print the explained variance ratio for each component
print(component_variance_ratio)
# Plot dotted lines
plt.axvline(x=num_components - 1, color="r", linestyle="--")
plt.axhline(y=cumulative_variance[num_components - 1], color="r", linestyle="--")

# Add label for explained variance
plt.text(
    num_components - 0.75,
    cumulative_variance[num_components] - 0.03,
    f"{cumulative_variance[num_components-1]*100:.2f}%",
    color="r",
)

plt.show()
# Print the selected number of components
print(f"Selected number of components: {num_components}")
```

In [ ]:
```
pca = PCA(n_components=4)
df_transformed = pca.fit_transform(df)
```

2 Anomaly Detection

In [ ]:
```
from pycaret.anomaly import *

# Load the dataset (replace 'data_array' with your NumPy array)
data = df_transformed

# Convert the NumPy array to a DataFrame
df = pd.DataFrame(data, columns=[f"PC{i}" for i in range(1, 5)])

# Initialize the PyCaret setup
anomaly_setup = setup(df, normalize=True)

# Create and evaluate the models
iforest = create_model("iforest")
lof = create_model("lof")
svm = create_model("svm")

# Get the decision scores or probabilities
iforest_scores = iforest.decision_scores_
lof_scores = lof.decision_scores_
svm_scores = svm.decision_scores_

# Manually adjust the anomaly detection thresholds
iforest_threshold = 0.2  # Adjust the threshold value as per your requirement
lof_threshold =  10  # Adjust the threshold value as per your requirement
svm_threshold = 0  # Adjust the threshold value as per your requirement

# Generate the predictions based on the adjusted thresholds
iforest_preds = iforest_scores > iforest_threshold
lof_preds = lof_scores > lof_threshold
svm_preds = svm_scores < svm_threshold

# Plotting the anomalies
plt.figure(figsize=(15, 5))

# Plotting Isolation Forest
plt.subplot(1, 3, 1)
sns.scatterplot(data=df, x="PC1", y="PC2", color="gray")
sns.scatterplot(data=df[iforest_preds], x="PC1", y="PC2", color="red")
plt.title("Isolation Forest")

# Plotting Local Outlier Factor
plt.subplot(1, 3, 2)
sns.scatterplot(data=df, x="PC1", y="PC2", color="gray")
sns.scatterplot(data=df[lof_preds], x="PC1", y="PC2", color="red")
plt.title("Local Outlier Factor")
```

```
        # Plotting Support Vector Machines
        plt.subplot(1, 3, 3)
        sns.scatterplot(data=df, x="PC1", y="PC2", color="gray")
        sns.scatterplot(data=df[svm_preds], x="PC1", y="PC2", color="red")
        plt.title("Support Vector Machines")

        plt.tight_layout()
        plt.show()
```

In [ ]:
```
        # Plotting the anomalies
        plt.figure(figsize=(15, 5))

        # Plotting Isolation Forest
        plt.subplot(1, 3, 1)
        sns.scatterplot(data=df, x="PC1", y="PC3", color="gray")
        sns.scatterplot(data=df[iforest_preds], x="PC1", y="PC3", color="red")
        plt.title("Isolation Forest")

        # Plotting Local Outlier Factor
        plt.subplot(1, 3, 2)
        sns.scatterplot(data=df, x="PC1", y="PC3", color="gray")
        sns.scatterplot(data=df[lof_preds], x="PC1", y="PC3", color="red")
        plt.title("Local Outlier Factor")

        # Plotting Support Vector Machines
        plt.subplot(1, 3, 3)
        sns.scatterplot(data=df, x="PC1", y="PC3", color="gray")
        sns.scatterplot(data=df[svm_preds], x="PC1", y="PC3", color="red")
        plt.title("Support Vector Machines")

        plt.tight_layout()
        plt.show()
```

In [ ]:
```
        # Plotting the anomalies
        plt.figure(figsize=(15, 5))

        # Plotting Isolation Forest
        plt.subplot(1, 3, 1)
        sns.scatterplot(data=df, x="PC2", y="PC3", color="gray")
        sns.scatterplot(data=df[iforest_preds], x="PC2", y="PC3", color="red")
        plt.title("Isolation Forest")

        # Plotting Local Outlier Factor
        plt.subplot(1, 3, 2)
        sns.scatterplot(data=df, x="PC2", y="PC3", color="gray")
        sns.scatterplot(data=df[lof_preds], x="PC2", y="PC3", color="red")
        plt.title("Local Outlier Factor")
```

```
# Plotting Support Vector Machines
plt.subplot(1, 3, 3)
sns.scatterplot(data=df, x="PC2", y="PC3", color="gray")
sns.scatterplot(data=df[svm_preds], x="PC2", y="PC3", color="red")
plt.title("Support Vector Machines")

plt.tight_layout()
plt.show()
```

3 Clustering

```
In [ ]: from pycaret.clustering import *
```

```
In [ ]: df_transformed = df_transformed[~lof_preds]
```

3.1 K-Means

```
In [ ]: # df_sample = df.sample(2500, random_state=2023)
        s = setup(df_transformed, session_id=221, normalize=True)
```

```
In [ ]: models()
```

```
In [ ]: kmeans = create_model("kmeans")
```

```
In [ ]: plot_model(kmeans, plot="elbow")
```

```
In [ ]: # Define a list of cluster numbers (k) to evaluate
        cluster_nums = range(2,11)

        # Initialize empty lists to store silhouette scores and plot coordinates
        silhouette_scores = []
        x_coords = []
        y_coords = []

        # Iterate over cluster numbers
        for k in tqdm(cluster_nums):
            # Create a clustering model with the current k
            kmeans = create_model('kmeans', num_clusters=k, verbose=False)

            # Assign clusters to the data points
            assign_clusters = assign_model(kmeans)

            # Extract the cluster labels
            labels = assign_clusters['Cluster']

            # Calculate the silhouette score
            score = silhouette_score(df_transformed, labels)
```

```
            # Store the silhouette score for the current k
            silhouette_scores.append(score)

            # Store coordinates for plotting
            x_coords.append(k)
            y_coords.append(score)

        # Plot the silhouette scores
        plt.plot(x_coords, y_coords, marker='o')
        plt.xlabel('Number of clusters (k)')
        plt.ylabel('Silhouette Score')
        plt.title('Silhouette Score per Cluster Number')
        plt.show()
```

```
In [ ]: kmeans = create_model("kmeans", num_clusters=4)
        plot_model(kmeans, plot="cluster")
```

3.2 Agglomerative Clustering

```
In [ ]: import pandas as pd
        from sklearn.preprocessing import StandardScaler
        from sklearn.cluster import AgglomerativeClustering
        from scipy.cluster.hierarchy import dendrogram, linkage
        import matplotlib.pyplot as plt

        # Normalize the dataframe
        scaler = StandardScaler()
        df_normalized = pd.DataFrame(scaler.fit_transform(df_transformed)).sample(100)

        # Perform agglomerative clustering with centroid linkage
        model_centroid = linkage(df_normalized, method='centroid')

        # Perform agglomerative clustering with average linkage
        model_average = linkage(df_normalized, method='average')

        # Perform agglomerative clustering with Ward's method
        model_ward = linkage(df_normalized, method='ward')

        # Plot the dendrogram for centroid linkage
        plt.figure(figsize=(10, 6))
        dendrogram(model_centroid, labels=labels_centroid, leaf_font_size=10)
        plt.title("Dendrogram - Centroid Linkage")
        plt.xlabel("Sample Index")
        plt.ylabel("Distance")
        plt.show()

        # Plot the dendrogram for average linkage
        plt.figure(figsize=(10, 6))
```

```
        dendrogram(model_average, labels=labels_average, leaf_font_size=10)
        plt.title("Dendrogram - Average Linkage")
        plt.xlabel("Sample Index")
        plt.ylabel("Distance")
        plt.show()

        # Plot the dendrogram for Ward's method
        plt.figure(figsize=(10, 6))
        dendrogram(model_ward, labels=labels_ward, leaf_font_size=10)
        plt.title("Dendrogram - Ward's Method")
        plt.xlabel("Sample Index")
        plt.ylabel("Distance")
        plt.show()
```

```
In [ ]: hclust = create_model("hclust", linkage="ward", num_clusters=3)
```

```
In [ ]: plot_model(hclust, "cluster")
```

Appendix C - Supervised Learning

June 25, 2023

In [ ]:
```
import pandas as pd
import numpy as np

from pycaret.classification import *
```

In [ ]:
```
# Bug with latest version of pycaret - need to revert to old version of sklearn
#!pip install --user --force-reinstall scikit-learn==1.2.1
```

In [ ]:
```
raw_df = pd.read_csv("./Loan_status_2007-2020Q3.gzip", compression=None, low_memory=Fa
```

0.0.1 I. Preprocessing Step

In [ ]:
```
accepted_df = raw_df[~raw_df.loan_amnt.isna()]
```

In [ ]:
```
df = accepted_df.copy()
```

In [ ]:
```
# Drop columns with more than 90% missing values
df.dropna(axis=1, thresh=len(df) * 0.9, inplace=True)

# Drop rows with any missing values
df.dropna(inplace=True)
```

In [ ]:
```
# Interest Rate
df["int_rate"] = df["int_rate"].str.rstrip("%").astype("float") / 100.0

# Employment Length
df["emp_length"] = df["emp_length"].str.extract(r"(\d+)")
df["emp_length"] = pd.to_numeric(df["emp_length"], errors="coerce")
```

In [ ]:
```
# Application Purpose
df["purpose"] = df["purpose"].str.replace("_", " ")
df["purpose"] = df["purpose"].str.title()
```

In [ ]:
```
# Application Purpose
df["purpose"] = np.where(df["purpose"] == "Car", "Major Purchase", df["purpose"])
df["purpose"] = np.where(df["purpose"] == "Vacation", "Major Purchase", df["purpose"])
df["purpose"] = np.where(df["purpose"] == "Small Business", "Major Purchase", df["purpo
df["purpose"] = np.where(df["purpose"] == "Wedding", "Major Purchase", df["purpose"])
```

```
df["purpose"] = np.where(df["purpose"] == "Moving", "Other", df["purpose"])
df["purpose"] = np.where(df["purpose"] == "House", "Other", df["purpose"])
df["purpose"] = np.where(df["purpose"] == "House", "Other", df["purpose"])
df["purpose"] = np.where(df["purpose"] == "Educational", "Other", df["purpose"])
df["purpose"] = np.where(df["purpose"] == "Renewable Energy", "Other", df["purpose"])
df["purpose"] = np.where(df["purpose"] == "Medical", "Other", df["purpose"])
```

In [ ]:
```
# Grade
df["grade"] = np.where(df["grade"] == "D", "D and Below", df["grade"])
df["grade"] = np.where(df["grade"] == "E", "D and Below", df["grade"])
df["grade"] = np.where(df["grade"] == "F", "D and Below", df["grade"])
df["grade"] = np.where(df["grade"] == "G", "D and Below", df["grade"])
```

In [ ]:
```
# Home Ownership Type
df["home_ownership"] = np.where(df["home_ownership"] == "ANY", "OTHER", df["home_owners
df["home_ownership"] = np.where(df["home_ownership"] == "NONE", "OTHER", df["home_owne
```

0.0.2 II. Supervised Learning - Set Up

In [ ]:
```
# Target: Delinquent in last 2 years (flag)
df["flag_delinq2yrs"] = np.where(df["delinq_2yrs"] == 0, 0, 1)
```

In [ ]:
```
supervised_features = [
    "loan_amnt",
    "term",
    "int_rate",
    "grade",
    "home_ownership",
    "annual_inc",
    "purpose",
    "emp_length",
    "flag_delinq2yrs"
]
```

In [ ]:
```
df_fit = df[supervised_features]
```

In [ ]:
```
# home_ownership = "OTHER" is 0.1%
df_fit = df_fit[df_fit["home_ownership"] != "OTHER"]
```

In [ ]:
```
df_fit.head()
```

In [ ]:
```
get_config("X_train_transformed")
```

In [ ]:
```
# Check if target variable is balanced or not
df_fit["flag_delinq2yrs"].mean()
```

In [ ]:
```
supervised_setup = setup(
    df_fit,
    target = 'flag_delinq2yrs',
```

```
    categorical_features = ["term", "grade", "home_ownership", "purpose"],
    normalize = True,
    normalize_method = "minmax",
    fix_imbalance = True,
    session_id = 221
)
```

0.0.3 III. Supervised Learning - Training

```
In [ ]: supervised_models = compare_models(include = ["lr", "ridge", "lda", "xgboost", "rf"], s
```

```
In [ ]: save_compare_models = pull()
        save_compare_models.reset_index().to_csv("table_compare_supervised_models.csv", index=F
```

```
In [ ]: print(supervised_models)
```

```
In [ ]: xgb = create_model("xgboost")
```

```
In [ ]: save_model(xgb, 'xgb_model')
        print(xgb)
```

```
In [ ]: xgb = load_model("xgb_model")
```

0.0.4 IV. Supervised Learning - Performance Metrics

```
In [ ]: evaluate_model(xgb)
```

```
In [ ]: evaluate_model(xgb)
```

```
In [ ]: evaluate_model(xgb)
```

```
In [ ]: evaluate_model(xgb)
```

```
In [ ]: evaluate_model(xgb)
```

0.0.5 V. Supervised Learning - Hyperparameter Tuning

```
In [ ]: #tuned_xgb = tune_model(xgb)
```

```
In [ ]: #save_model(tuned_xgb, 'tuned_xgb_model')
        #print(tuned_xgb)
```

```
In [ ]: #evaluate_model(tuned_xgb)
```

```
In [ ]: #evaluate_model(tuned_xgb)
```