Credit Score Classification for Cost-Effective Loan Campaigning

Ryan S. Dunn, Azucena Faus, Sanjay Regi Phillip

Shiley-Marcos School of Engineering, University of San Diego

Abstract

Financial institutions are under immense pressure to maintain safe and adequate lending strategies within a rising interest rate environment. By reviewing data from various current credit profiles, this project seeks to determine a credit classification strategy that can support financial institutions' strategic growth and macroeconomic stability by correctly identifying the. credit risks of potential borrowers. Through exploratory data analysis, the data displayed minimal correlation of modeling inputs, and where appropriate, correlated variables were removed to account for multicollinearity. The baseline model accuracy of all records classified as "No" yielded an accuracy of 82%. The Random Forest and KNN-Manhattan models displayed strong predictive capabilities to determine borrower credit classifications at an accuracy rate of 90% and 91%, and "No" recall scores of 96% and 95%, respectively. The Random Forest model was ultimately chosen as the choice model due to its enhanced explainability over the KNN-Manhattan model. The Random Forest model can produce a feature-importance metric, which allows for enhanced explainability to financial regulators for credit denial and provides critical insight into Current Expected Credit Loss reporting.

Keywords: machine learning, deep learning, Random Forest, neural network, perceptron, KNN, decision trees, credit score, credit classification, interest rate risk

Table of Contents

List of Tables

Credit Score Classification for Cost-Effective Loan Campaigning

Adverse selection in loan markets is a primary concern for lenders, as persons with a higher likelihood of defaulting on debts are more likely to be debt seekers. Additionally, loan seekers may have an incentive to engage in high-risk activities after receiving loan funds which may yield large returns. However, they would be considered an undesirable investment from the bank's perspective (Mishkin, 2010). This was precisely the case during the financial crisis of 2008, where banks and other financial institutions failed to effectively address the asymmetric information presented within a modern economy loan market. This resulted in a significant restructuring of supervisory credit and risk management methodologies, in the form of the Current Expected Credit Losses (CECL) implementation by the Financial Accounting Standards Board (FASB), and subsequent increased supervision and monitoring from banking regulators (Office of the Comptroller of the Currency, 2020).

Researchers from the Federal Reserve Bank of New York released credit data in August 2022, which showed total household debt now being $2 trillion higher than at the start of the pandemic, with aggregate limits on credit card accounts increasing by $100 billion – the largest increase since 2011. Over 200 million new credit accounts were opened in the second quarter of 2022, the highest increase quarter-over-quarter since 2008. Regarding housing debt, 35% of the $758 billion newly originated mortgage debt was with persons holding a credit score under 760 (Federal Reserve Bank of New York, 2022). To prevent another financial crisis, ensure that credit markets do not become frozen, and allow for a smooth transition from the current high rate of inflation, maintaining functional financial services via sound credit lending practices is paramount.

Problem Definition

The current state of the lending market has derived demand for machine and deep learning models that can accurately predict borrowers with a high likelihood to repay debts and subsequently identify high-risk borrowers. The development of such a model can accomplish various needs for financial institutions, such as minimizing the impact on net interest margins within a rising rates environment and mitigating CECL risk associated with exposure to interest-rate risk. Beyond banking, however, a robust machine learning credit classification model will ensure the economy can grow while traversing high inflation rates and minimize the potential for credit rationing and a global recession.

Exploratory Data Analysis and Pre-Processing

The credit classification data consists of 100,000 client records with an output label that has three types of credit score classifications: “Good,” “Standard,” and “Poor.” There were 27 features, including ten categorical and 17 numerical.

Data Cleaning

The dataset had many irregular and missing values that needed correction, imputation, or removal. Irregularities were coerced using the pandas “coerce” function, and unnecessary underscores in data entries were removed via string operations. There were also some features that needed to be transformed into interpretable numerical values or flags via some feature engineering, such as Credit_Age_in_Years and Type_of_Loan.

Additionally, it was found that each client had eight rows of data for the months of January through August. One approach attempted was to average out the data across the eight months. This proved to give a very poor generalized predictive performance. So, the entire dataset was used for processing. Finally, some features were engineered (like Debt_to_Income_Ratio, derived from Outstanding_Debt/Annual_Income) to help derive further insights from the data.

Missing Data

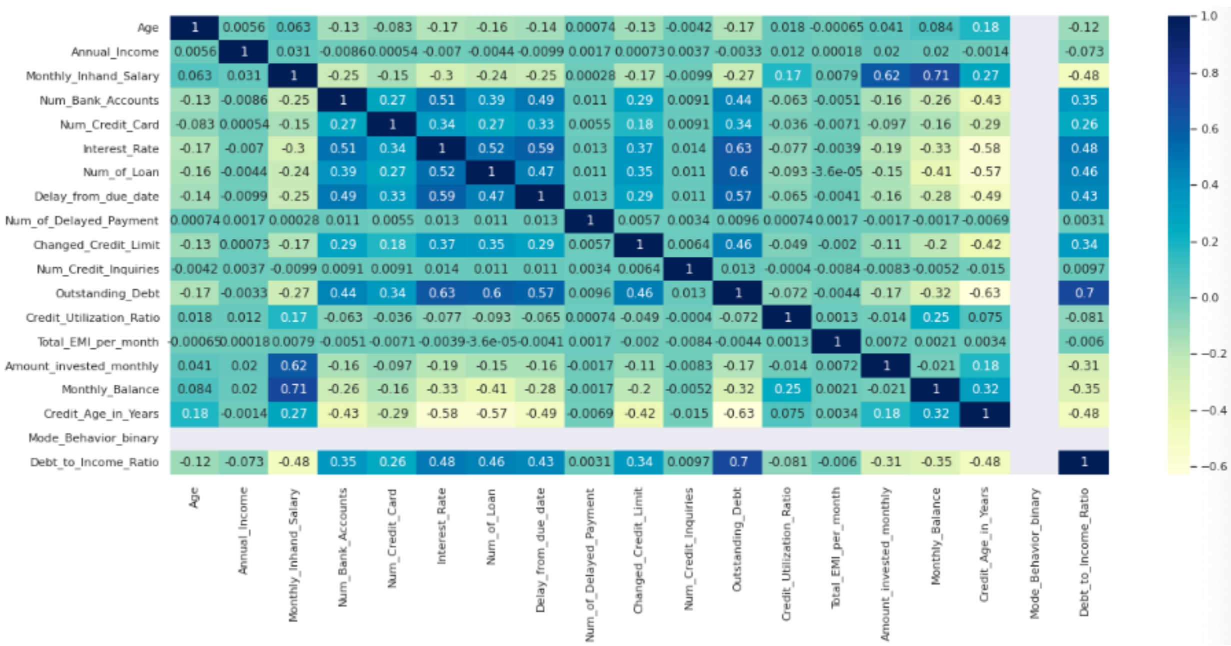
The datset had several missing and null values due to incoherent entries in features Monthly_Inhand_Salary, Age, Credit_Age_in_Years, Num_of_Delayed_Payment, Num_Credit_Inquiries, Monthly_Balance, Amount_Invested_monthly, and Changed_Credit_Limit. These features were imputed after the data split into train and test to prevent data leakage from test to training. The median for each feature with missing values was found from the training data and then used to impute both the training and test datasets before scaling.

Exploratory Data Analysis

Since feature selection is of vital importance when it comes to building effective machine learning algorithms, it is critical to find any collinearity between features. To find such relationships, a correlation heatmap was analyzed. A moderate correlation of .71 between Monthly_Inhand_Salary and Monthly_Balance, along with a scatter plot, confirmed the presence of collinearity.

Figure 1

Correlation Heat Map

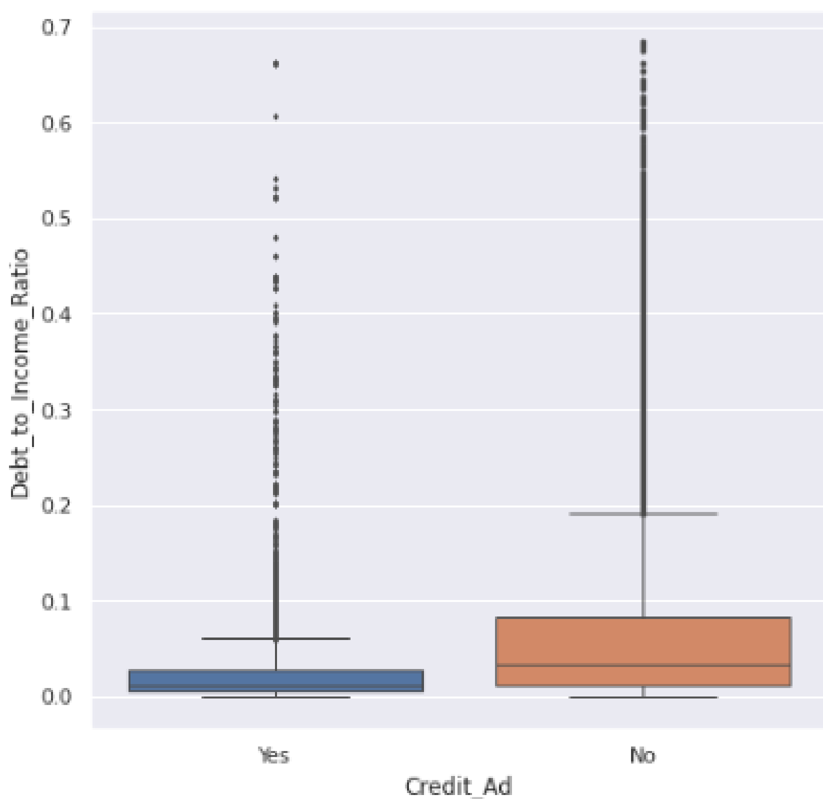


Since Monthly_Inhand_Salary had missing values and did not improve model performance, this feature was removed from consideration for analysis. Histograms for numerical data showed some skew in distributions, so standardization was applied before modeling. Upon observation of the statistical descriptions of the numerical data, obvious outliers were found for features like Num_Credit_Cards, Num_Bank_Accounts, Interest_Rate, Age, and Num_Loans so these were removed from the dataset.

A new output label was derived from here to fit the business problem: Credit_Ad. This output label would be "Yes" when the Credit_Score is "Good" and "No" when the Credit_Score is "Poor" or "Standard." Boxplots confirmed the assumption that Good Credit is highly associated with a low Debt_to_Income_Ratio more so than the standalone features Annual_Income or Outstanding_Debt. Therefore, Annual_Income was not used as a relevant feature for modeling, and the Debt_to_Income_Ratio was used instead.

Figure 2

Debt_to_Income Ration Relationship to Credit_Ad



Data Preprocessing for Modeling

After examining the data and several iterations for improvements in model performance, the dataset used for modeling after outlier removal was 94,641 records with 24 features (nine categorical and 15 numerical). The binary label Credit_Ad was the predicted output. A Stratified Random split was used with 90% training and 10% test data sets. The choice to select a stratified approach was used to maintain the class proportions of the output label in both the training and test sets. About 82% of the output label accounted for the “No” class, and 18% was “Yes.” This means that our model evaluation must include an analysis beyond simple accuracy measurements, like a close look at confusion matrices, f1_scores, recall, and precision for the “Yes” class.

After splitting, numerical features were imputed using the median training measurements for those features and applying a StandardScaler that brings the mean to 0 and standard deviation to 1 (normalization), as well as scaling the data to account for models that are sensitive to distance measurements (like KNN). Categorical features were hot encoded.

Modeling, Methods, Validation, and Performance Metrics

Methods, Validation, and Performance

A collection of eight machine learning algorithms were adapted to create experimental models for this binary classification problem. The algorithms included examples of neural networks, decision trees, gradient descent, and K-Nearest Neighbors methods. This collection of algorithms was intentionally chosen for their distinct characteristics that would prove to be unique and beneficial when predicting class.

Stratified random sampling was performed to partition the training and test dataset to evaluate the performance of each method. Furthermore, 5-fold cross-validation was performed using the partitioned training data to ensure that overfitting would not impact the results of the data. The algorithms were primarily scored using accuracy to find the performance of each model in predicting credit score classifications. Additionally, recognizing that misclassifying bad credit scores would be expensive, the models were also evaluated using precision and recall ensuring that the proportion of false positives predicted by each model was minimized.

Perceptron Model

The perceptron method aims to find a linear separator using stochastic gradient descent, to find a linear separation in the feature space (Brownlee, 2020). This method was chosen to understand if the credit score class could effectively be linearly separated with the feature space provided. For the model experimentation, two perceptron models were created using either balanced or unbalanced weights for the features. When balanced, the algorithm will adjust the weights to be inversely proportional to the class frequencies (Brownlee, 2020).

During model validation, it was observed that for the balanced perceptron, the average cross-validation score of 0.67 matched the accuracy scores of 0.67 on the training data and 0.67 on the test data. The closely matching accuracy scores across cross-validation, training, and the test data suggest that the model does not overfit and would perform similarly on new data matching the same class balances. When validating the unbalanced perceptron model, the model returned much higher accuracy scores with a training accuracy of 0.82 and a test accuracy of 0.82 as well. The model also scored an average cross-validation score of 0.77, suggesting that there may be some overfitting in the train and test results, although the accuracy scores are not too different from the cross-validation scores. Knowing that the target variable of credit score is unbalanced at an almost 80-20 split, it was surprising that the unbalanced perceptron model performed better than the balanced perceptron model as introducing weights should improve performance.

Neural Network

In continuing with neural network models, the next method developed was a multi-layer perceptron (MLP) model, a more complex neural network algorithm when compared to the perceptron method. This type of neural network method can more effectively learn nonlinear and complex relationships. It is advantageous as it is generalized and able to infer relationships from unseen data and works with many different types of data regardless of its specific distribution (Agrawal, 2021).

Following model fitting, validation showed that the MLP neural network performed better than the perceptron models with accuracy scores of 0.84 with the training data and 0.84 with the test data. Additionally, when conducting cross-validation, the observed average cross-validation score was 0.84, which aligned with the training and test accuracy scores suggesting that the model was not overfit.

Decision Tree

Decision trees are algorithms that attempt to leverage rules to make decisions. The provided features are used to create “Yes” or “No” questions that can provide rules for classification. An advantage of decision trees is that they are more easily interpretable and allow one to clearly see the decision-making progress of the algorithm (Bento, 2021). Thus, this method was chosen as it would allow classification predictions but also visualize the decision-making process of the model.

During model validation, the method performed poorly on the training data with only an accuracy of 0.67 but performed significantly better on the test data set at 0.86. The cross-validation score was also high, matching the test accuracy with an average cross-validation score of 0.86. The discrepancy between the training and test accuracy scores may be due to decision trees being sensitive to small changes in the data. Although stratified random sampling was used, it is possible that an unobserved imbalance in the dataset was the reason for the discrepancy and shows why decision trees may be difficult to work with.

Random Forest

Random forests are an ensemble method that can be seen as an extension of decision trees that builds several uncorrelated trees to identify the best rule set (Yiu, 2021). The random forest model leveraged for this method scored the highest possible accuracy for the training data at 1.0 and scored at

0.90 for the test data. The average cross-validation score was 0.91, suggesting that overfitting of the test data was avoided, although with an accuracy of 1.0, the model overfit the training data.

K-Nearest Neighbors

Another leveraged method is the K-Nearest Neighbors (KNN). The KNN method predicts classification by assuming that data points that share proximity to each other on the hyperplane are indeed the same class. This method finds the distance between points and classifies a data point based on the distance between the point and its neighboring points (Harrison, 2019).

Two KNN models were created using two types of distance calculating techniques. The first distance calculation was the Manhattan method which calculates grid-like distance and typically performs better on high dimensionality data. The other distance measurement method leveraged was Euclidean distance which measures the distance between two points directly.

The KNN model using Manhattan distance performed slightly better than the Euclidean distance model with a test accuracy score of 0.91 compared to 0.88. This is likely due to the Manhattan method's more robust form of calculating distance. However, the Manhattan distance KNN had the longest training time of any method leveraged for this project.

SGD Classifier

The final algorithm leveraged was the Stochastic Gradient Descent (SGD) method. The SGD method is another linear model that attempts to find a linear separator in the feature space. SGD methods perform better on larger datasets as they are significantly quicker in training (Patlolla, 2018). For the SGD method, 5-fold cross-validation was performed for a range of alpha hyperparameters to tune the model and find the best performing value of alpha. Using the best performing value of alpha, the strongest performing SGD model provided an accuracy of 0.84 on the test data.

Modeling Results and Findings

The best performing models were the KNN model using Manhattan distance and the Random Forest model. The KNN-Manhattan model slightly outperformed the Random Forest model in terms of testing accuracy. To get a better assessment, a deeper analysis of the classification reports for each model was performed. Surprisingly, the precision and recall scores for both models, for both the “Yes” and “No” classes, matched very closely and did not suggest a significant difference between the models. The F-1 scores were also closely related and did not suggest any significant differences.

Table 1

Model Accuracy Score

Model	Accuracy
KNN Manhattan	0.9064
Random Forest	0.9044
KNN Euclidean	0.8833
Decision Tree	0.8602
SGD log loss	0.838
Neural Network	0.8372
Unbalanced Perceptron	0.818
Balanced Perceptron	0.673

Table 2

KNN–Manhattan Classification Report

KNN Manhattan	Precision	Recall	F1 Score	Support
No	0.94	0.95	0.94	7782
Yes	0.74	0.72	0.73	1683
Macro Average	0.84	0.83	0.84	9465
Weighted Average	0.91	0.91	0.91	9465

Table 3

Random Forest Classification Report

Random Forest	Precision	Recall	F1 Score	Support
No	0.93	0.96	0.94	7782
Yes	0.76	0.67	0.71	1683
Macro Average	0.85	0.81	0.83	9465
Weighted Average	0.9	0.9	0.9	9465

Final Model

With both models performing similarly across accuracy, precision, and recall, other factors were considered outside of the typical performance metrics. Ultimately, the Random Forests model is the best model for this problem because of its quicker model training time and better interpretability. In finance, regulators require companies to explain reasons for credit approval decisions. Additionally, Random Forest models also do not require an extensive process to find optimal hyperparameters, as does KNN.

Conclusion

With high accuracy and specifically a strong ability to correctly classify persons that should not receive loan offers, the Random Forest model is suggested for adoption by financial institutions desiring to manage interest rate risk and the asymmetric information problem. By continuing to fine-tune model inputs and the enhancement of feature engineering by data scientists, credit analysts will be better equipped to assess incoming credit applicate risk, ensure to not over-anticipate lending market risks, minimize capital required to be held in reserves, and provide valuable insight to regulators during semi-annual audits. These realized monetary gains will strengthen quarterly performance, increase investor confidence, and allow for maximizing shareholder returns.

References

Agrawal, S. (2021, July 19). *Understand the basics of artificial neural network*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/07/understanding-the-basics-of-artificial-neural-network-ann/

Bento, C. (2021, June 28). *Decision tree classifier explained in real-life: Picking a vacation destination*. Toward Data Science. https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575

Brownlee, J. (2020, December 11). *Perceptron algorithm for classification in Python*. Machine Learning Mastery. https://machinelearningmastery.com/perceptron-algorithm-for-classification-in-python/

Federal Reserve Bank of New York. (2022, August 2). *Total household debt surpasses $16 trillion in Q2 2022; mortgage, auto loan, and credit card balances increase*. https://www.newyorkfed.org/newsevents/news/research/2022/20220802

Harrison, O. (2018, September 10). *Machine learning basics with the K-nearest neighbors algorithm*. Toward Data Science. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

Kuhn, M., & Johnson, K. (2016). *Applied predictive modeling*. Springer.

Larose, C., & Larose, D. (2019). *Data science using Python and R*. John Wiley & Sons, Inc.

Mishkin, F. S. (2010). *The economics of money, banking & financial markets* (9th ed.). Addison-Wesley.

Office of the Comptroller of the Currency. (2020, October 1). *Current expected credit losses: Final rule*. https://www.occ.treas.gov/news-issuances/bulletins/2020/bulletin-2020-85.html

Paris, R. (2022, June 22). *Credit score classification*. Kaggle. Retrieved July 15, 2022, from https://www.kaggle.com/datasets/parisrohan/credit-score-classification

Patlolla, V. (2018, November 28). *How to make SGD classifier perform as well as logistic regression using Parfit*. Toward Data Science. https://towardsdatascience.com/how-to-make-sgd-classifier-perform-as-well-as-logistic-regression-using-parfit-cc10bca2d3c4

Yiu, T. (2019, June 12). *Understanding random forest: How the algorithm works and why it is so effective*. Toward Data Science. https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Appendix

Data Retrieval and Enviroment Setup

Load Packages and Libraries

In [ ]:

```
%matplotlib inline
import pandas as pd
import numpy as np

import os
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import matplotlib as mpl
import matplotlib.pyplot as plt
import random
import statsmodels.tools.tools as stattools
import statsmodels.api as sm
from scipy import stats
from scipy.stats import mode

from sklearn import metrics
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, LabelEncoder, StandardSca
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score, train_test_split, StratifiedKFold
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, plot_confus
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.linear_model import Perceptron
from sklearn.impute import KNNImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import SGDClassifier
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pa
ndas.util.testing is deprecated. Use the functions in the public API at pandas.testing ins
tead.
  import pandas.util.testing as tm
```

Data Upload

In [ ]:

```
# mount google drive for data upload
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

```
raw_df = pd.read_csv('/content/drive/MyDrive/ADS504 - Final/train.csv', header=0)
credit_df = raw_df
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: DtypeWarnin
g: Columns (26) have mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Exploratory Data Analysis & PreProcessing

Data Summary Statistics

In [ ]:
```
credit_df.shape
```

Out[ ]:
```
(100000, 28)
```

Feature Engineering

Convert Misclassified String to Numeric

In [ ]:
```
# convert "credit history age" feature to years
def conv_credit_age(x):
    if pd.isnull(x):
        return x
    spt = x.split(' ')
    yr = int(spt[0].split('yr')[0])
    return (yr)

#apply the function and assign to new variable
credit_df['Credit_Age_in_Years'] = credit_df['Credit_History_Age'].apply(conv_credit_age)

# Added by Susy to have ability to impute missing values using KNNImpute later
credit_df['Credit_Age_in_Years']=pd.to_numeric(credit_df['Credit_Age_in_Years'])

#delete the previous column for Credit History Age
del credit_df['Credit_History_Age']
credit_df.head(2)
```

Out[ ]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_E
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	

2 rows × 28 columns

Data Cleansing

In [ ]:
```
# replace non-numeric values with null values
credit_df['Amount_invested_monthly']=pd.to_numeric(credit_df['Amount_invested_monthly'], e
credit_df['Monthly_Balance']=pd.to_numeric(credit_df['Monthly_Balance'], errors='coerce')

# clean invalid Credit_Mix values
credit_mix_choices = ('Good', 'Bad', 'Standard')

credit_df.loc[~credit_df['Credit_Mix'].isin(credit_mix_choices), 'Credit_Mix'] = np.nan

# convert values to category
credit_df['Credit_Mix'] = credit_df['Credit_Mix'].astype('category')
```

```
In [ ]: credit_df.loc[~credit_df['Payment_Behaviour'].str.contains('payments'), 'Payment_Behaviou
        credit_df['Payment_Behaviour'] = credit_df['Payment_Behaviour'].astype('category')
```

```
In [ ]: # replace invalid characters
        # There are a few errors where '_' is at end or beginning of count

        credit_df['Num_of_Loan'] = pd.to_numeric(credit_df['Num_of_Loan'].str.replace('_', ''))
        credit_df['Annual_Income'] = pd.to_numeric(credit_df['Annual_Income'].str.replace('_', '')
        credit_df['Num_of_Delayed_Payment'] = pd.to_numeric(credit_df['Num_of_Delayed_Payment'].st
        credit_df['Outstanding_Debt'] = pd.to_numeric(credit_df['Outstanding_Debt'].str.replace('_
        credit_df['Age'] = pd.to_numeric(credit_df['Age'].str.replace('_', ''))
        credit_df['Changed_Credit_Limit'] = pd.to_numeric(credit_df['Changed_Credit_Limit'].str.re


        credit_df.loc[credit_df['Occupation'].str.contains('_'), 'Occupation'] = 'Other'
        credit_df['Monthly_Inhand_Salary'] = pd.to_numeric(credit_df['Monthly_Inhand_Salary'])
```

```
In [ ]: # Account for unrealistic outliers
        # These are likely to be some bug in data collection

        # Number of loans feature
        credit_df.loc[credit_df["Num_of_Loan"] > 9, "Num_of_Loan"] = np.NaN

        # Replaces negatives with 0
        credit_df['Num_of_Loan'] = credit_df['Num_of_Loan'].apply(lambda x : x if x > 0 else 0)

        # Age feature
        credit_df.loc[credit_df["Age"] > 120, "Age"] = 0
        credit_df.loc[credit_df["Age"] < 18, "Age"] = 0
        credit_df['Age'] = credit_df['Age'].replace(0, np.NaN)
```

Accommodating for Time Series Data

```
In [ ]: # USING FULL DATASET

        credit_df = credit_df[(credit_df['Num_Bank_Accounts'] < 100) & (credit_df['Num_Bank_Accour
        #credit_df_new.shape

        credit_df = credit_df[(credit_df['Num_Credit_Card'] < 100) & (credit_df['Num_Credit_Card']
        #credit_df_new.shape

        credit_df = credit_df[(credit_df['Interest_Rate'] < 50) & (credit_df['Interest_Rate'] > -1
        credit_df.shape

Out[ ]: (94641, 28)
```

```
In [ ]: # USING ALL DATASET

        credit_df['Payment_Behaviour'] = credit_df['Payment_Behaviour'].astype('string')
        credit_df['Payment_of_Min_Amount'] = credit_df['Payment_of_Min_Amount'].astype('string')
        credit_df['Occupation'] = credit_df['Occupation'].astype('string')


        credit_df['Payment_Behaviour'] = credit_df['Payment_Behaviour'].astype('category')
        credit_df['Payment_of_Min_Amount'] = credit_df['Payment_of_Min_Amount'].astype('category')
        credit_df['Occupation'] = credit_df['Occupation'].astype('category')
```

```
In [ ]: # USING ENTIRE DATASET
```

```
pan = {"['High_spent_Medium_value_payments']" : 'High_Spend',"['High_spent_Large_value_pay
       "['High_spent_Small_value_payments']": 'High_Spend', "['Low_spent_Large_value_payme
       "['Low_spent_Small_value_payments']" : 'Low_Spend', "['Low_spent_Medium_value_payme
credit_df['Mode_Behavior_binary'] = credit_df['Payment_Behaviour'].map(pan)
```

In [ ]:

```
# USE ENTIRE DATASET STRIP SOME FEATURES

unnecessary_features = ['Payment_Behaviour', 'Name', 'Month', 'SSN',
                        ]

credit_df = credit_df.drop(labels = unnecessary_features, axis=1)
```

In [ ]:

```
credit_df.describe()
```

Out[ ]:

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate
count	86596.000000	9.464100e+04	80454.000000	94641.000000	94641.000000	94641.000000
mean	34.440771	1.757294e+05	4196.572372	5.404222	5.605446	14.531091
std	10.150058	1.423242e+06	3186.064195	2.960657	2.976443	8.739028
min	18.000000	7.005930e+03	303.645417	0.000000	0.000000	1.000000
25%	26.000000	1.944141e+04	1625.265833	3.000000	4.000000	7.000000
50%	34.000000	3.758034e+04	3096.066250	5.000000	5.000000	13.000000
75%	42.000000	7.281486e+04	5961.745000	7.000000	7.000000	20.000000
max	118.000000	2.419806e+07	15204.633333	99.000000	99.000000	34.000000

Creating Debt to Income Ratio

In [ ]:

```
# USING ENTIRE DATASET
# Debt/income
# invested/income,

credit_df['Debt_to_Income_Ratio']=credit_df['Outstanding_Debt']/credit_df['Annual_Income']
```

EDA Plots

In [ ]:

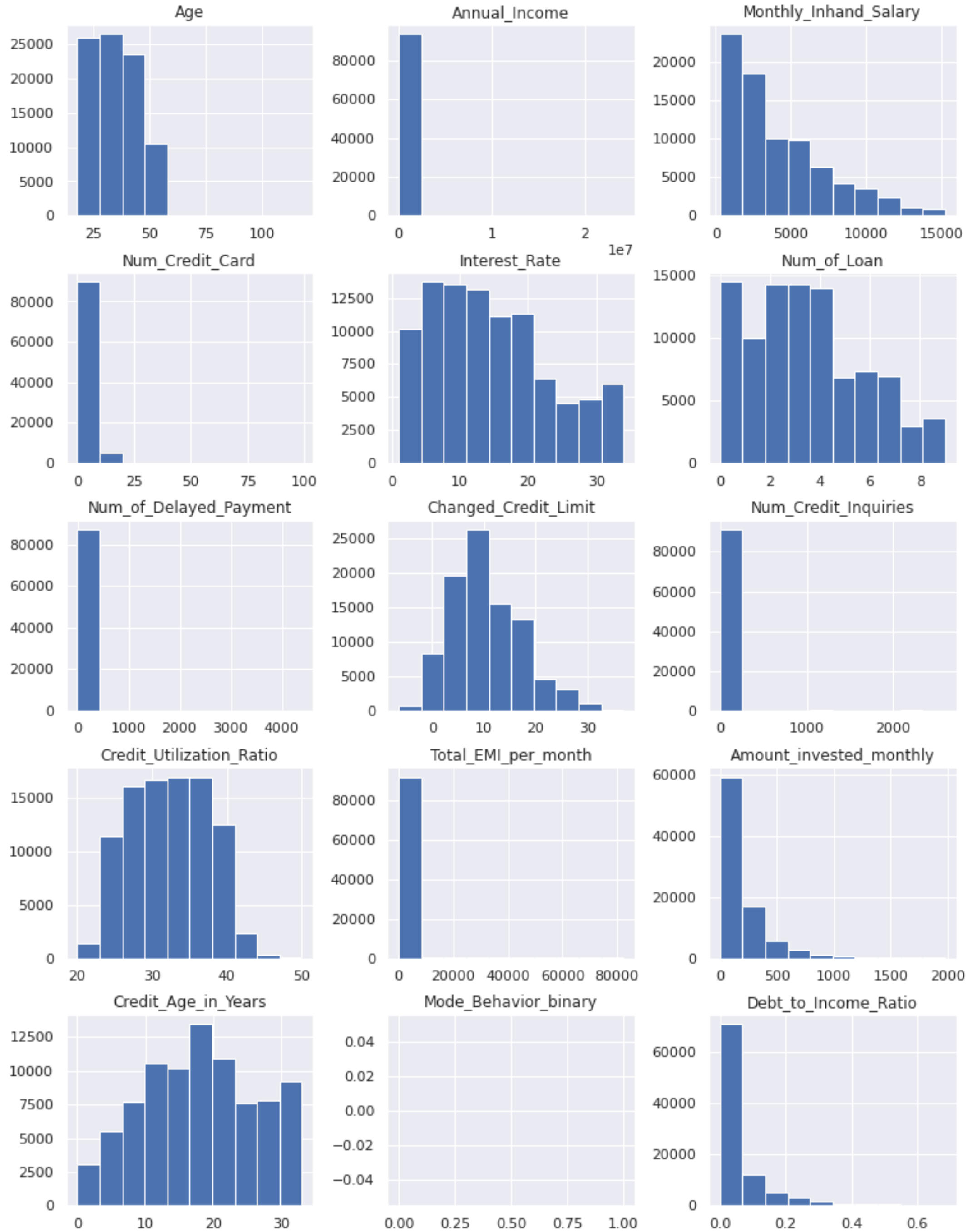
```
credit_df.shape
```

Out[ ]:

```
(94641, 26)
```

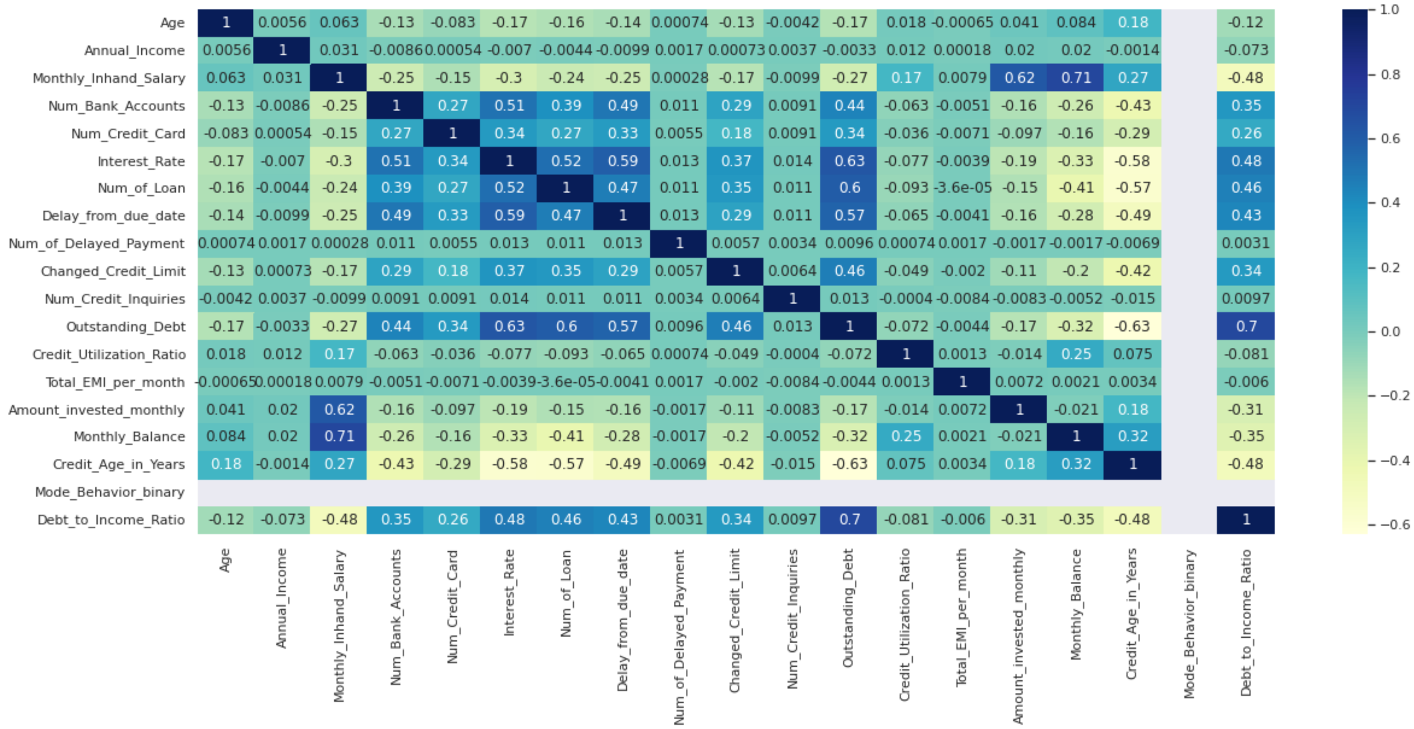
In [ ]:

```
# USING ENTIRE DATASET
credit_df.hist(figsize = (17,17));
```



In [ ]:

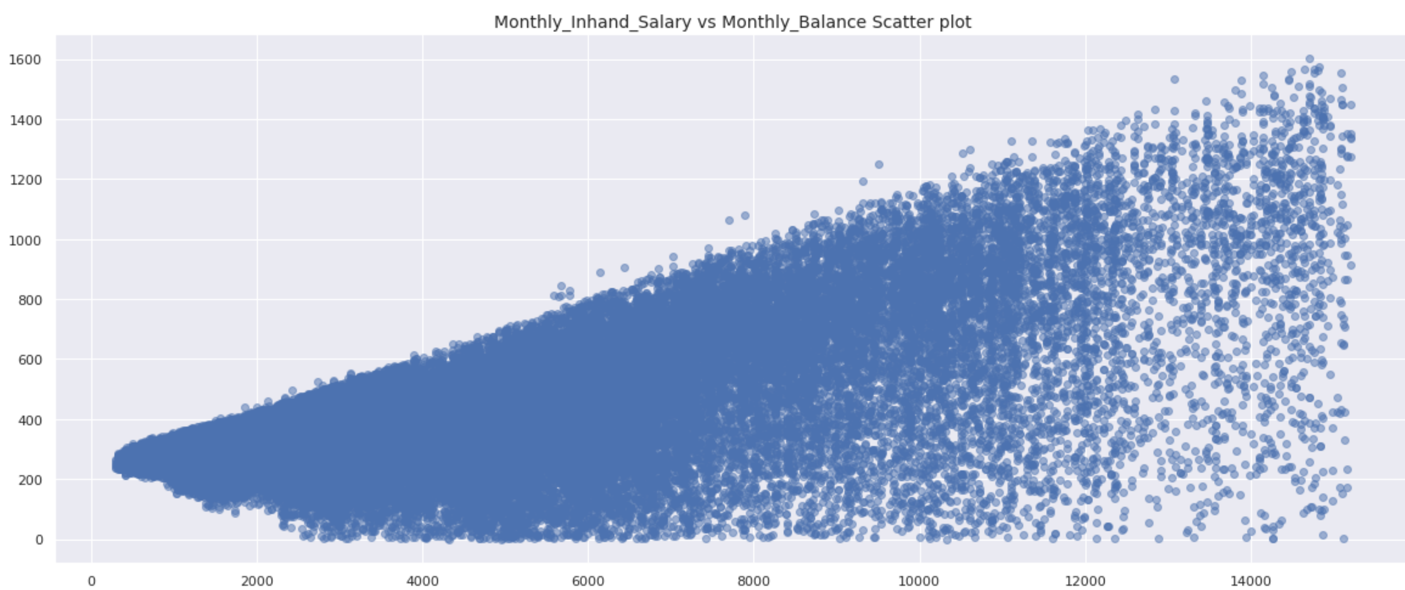
```
# USING ENTIRE DATASET
# plot the heatmap and annotation on it
#Correlation matrix
corr_matrix_credit = credit_df.corr()
sns.set(rc = {'figure.figsize':(20,8)})
sns.heatmap(corr_matrix_credit, cmap="YlGnBu", annot=True)
plt.show()
```



EDA Scatterplots

In [ ]:

```
# USING ENTIRE DATASET
plt.scatter(credit_df['Monthly_Inhand_Salary'], credit_df['Monthly_Balance'], alpha=0.5)
plt.title('Monthly_Inhand_Salary vs Monthly_Balance Scatter plot', fontsize = 14)
plt.show()
```

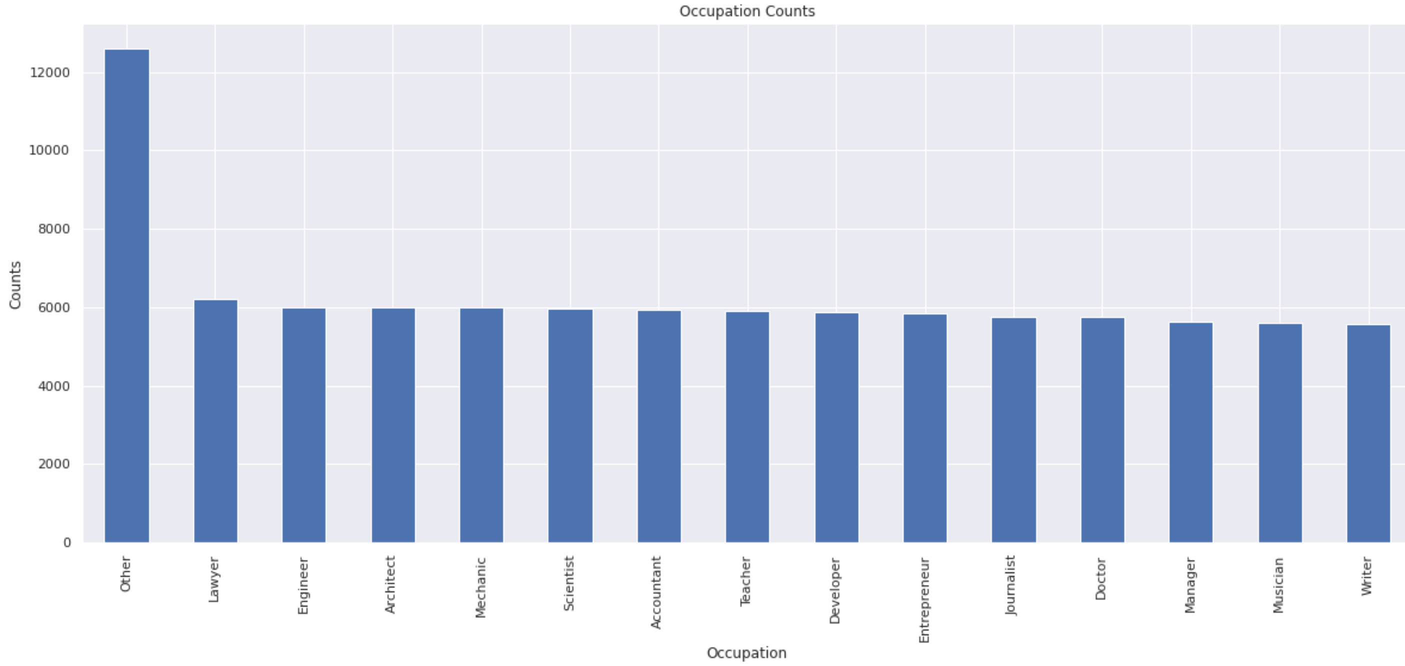


Moderate linear correlation with 0.71 correlation index between the two variables, Monthly_Inhand_Salary and Avg_Balance_per_month

---

In [ ]:

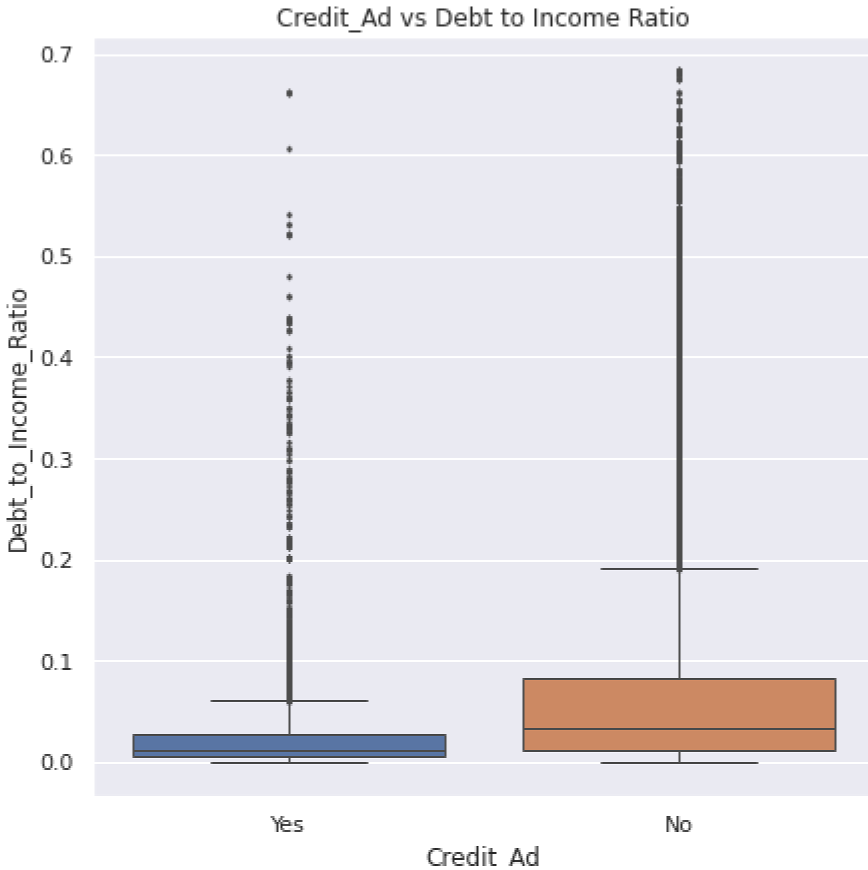
```
credit_df['Occupation'].value_counts().plot(kind='bar')
plt.title('Occupation Counts')
plt.xlabel('Occupation')
plt.ylabel('Counts')
plt.show()
```



EDA Boxplots

In [ ]:
```
# USE ENTIRE DATASET BINARY LABEL

#  Combine Poor and Standard categories:
# Different from numerical binary option
# Using 'Yes' to denote we should advertise
# to these clients:

pan = {'Poor' : 'No','Standard' : 'No',
       'Good': 'Yes'}
credit_df['Credit_Ad'] = credit_df['Credit_Score'].map(pan)
```

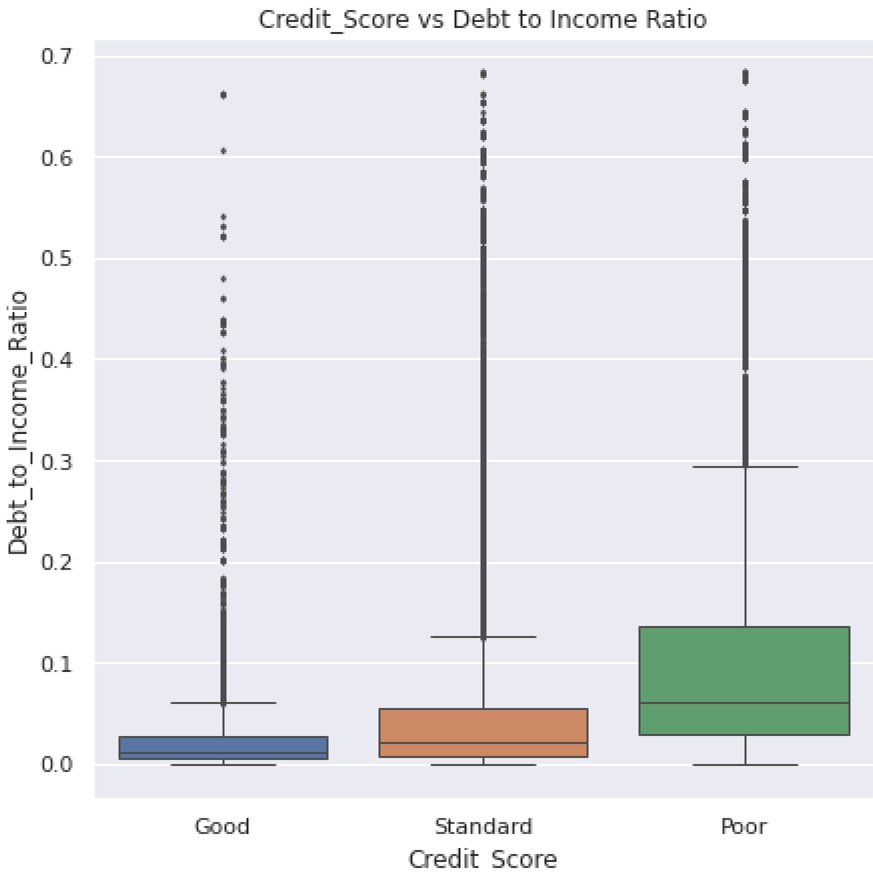
In [ ]:
```
plt.figure(figsize=(7,7))
ax = sns.boxplot(x='Credit_Ad',y='Debt_to_Income_Ratio',data=credit_df,linewidth=1,fliersi
_ = ax.set_xticklabels(ax.get_xticklabels())
plt.title('Credit_Ad vs Debt to Income Ratio')


plt.figure(figsize=(7,7))
ax = sns.boxplot(x='Credit_Ad',y='Monthly_Inhand_Salary',data=credit_df,linewidth=1,fliers
#_ = ax.set_xticklabels(ax.get_xticklabels())
plt.title('Credit_Ad vs Monthly Inhand Salary')

plt.show()
```

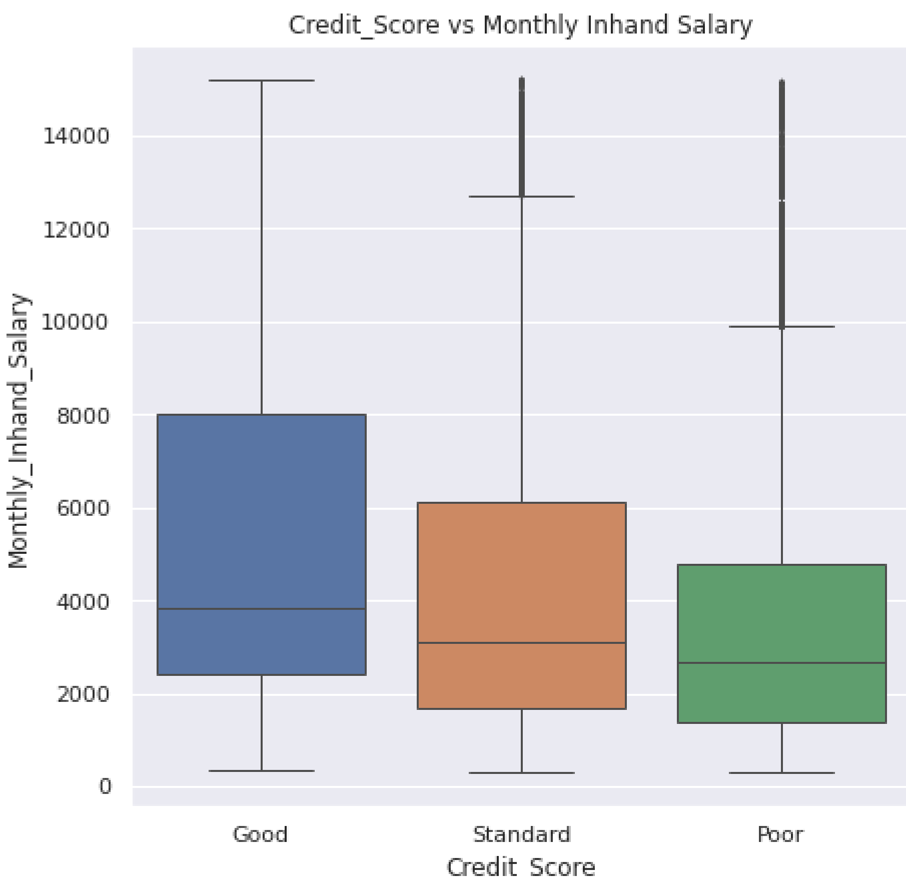


In [ ]:

```
plt.figure(figsize=(7,7))
ax = sns.boxplot(x='Credit_Score',y='Debt_to_Income_Ratio',data=credit_df,linewidth=1,flie
_ = ax.set_xticklabels(ax.get_xticklabels())
plt.title('Credit_Score vs Debt to Income Ratio')
```

```
plt.figure(figsize=(7,7))
ax = sns.boxplot(x='Credit_Score',y='Monthly_Inhand_Salary',data=credit_df,linewidth=1,fli
plt.title('Credit_Score vs Monthly Inhand Salary')
#_ = ax.set_xticklabels(ax.get_xticklabels())
plt.show()
```



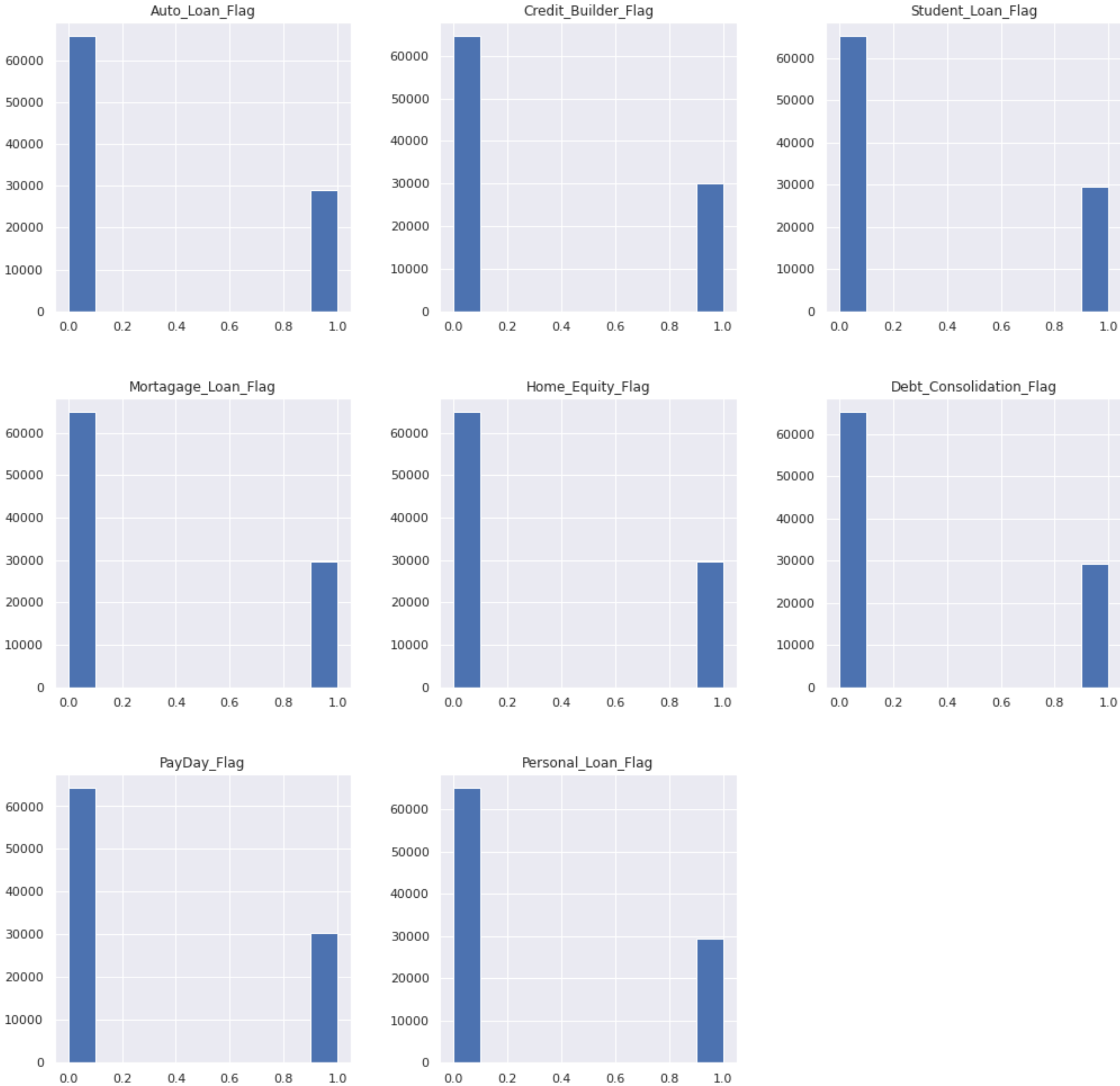


Loan Type Feature Engineering

In [ ]:
```
# USING ENTIRE DATASET LOAN TYPE FLAGS

#parse out the different loand types within the "Type_of_Loan" column
credit_df['Auto_Loan_Flag'] = credit_df['Type_of_Loan'].astype(str).map(lambda x: 1 if "Au
credit_df['Credit_Builder_Flag'] = credit_df['Type_of_Loan'].astype(str).map(lambda x: 1 i
credit_df['Student_Loan_Flag'] = credit_df['Type_of_Loan'].astype(str).map(lambda x: 1 if
credit_df['Mortagage_Loan_Flag'] = credit_df['Type_of_Loan'].astype(str).map(lambda x: 1 i
credit_df['Home_Equity_Flag'] = credit_df['Type_of_Loan'].astype(str).map(lambda x: 1 if '
credit_df['Debt_Consolidation_Flag'] = credit_df['Type_of_Loan'].astype(str).map(lambda x:
credit_df['PayDay_Flag'] = credit_df['Type_of_Loan'].astype(str).map(lambda x: 1 if "Payda
credit_df['Personal_Loan_Flag'] = credit_df['Type_of_Loan'].astype(str).map(lambda x: 1 if
```

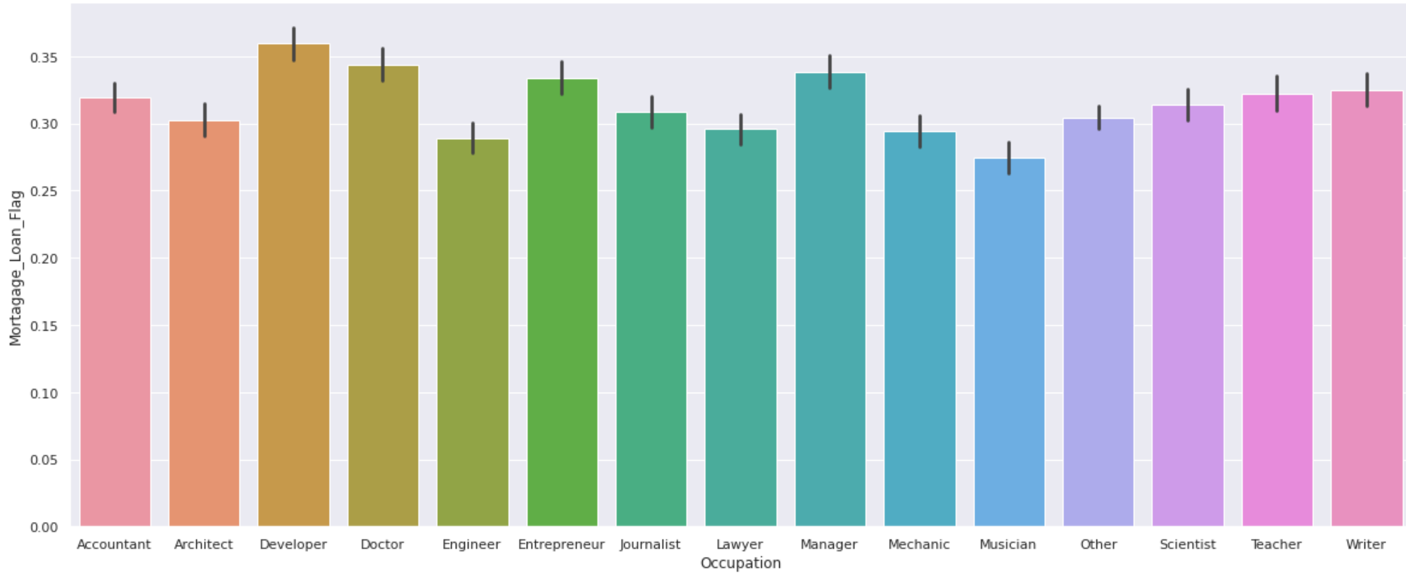
In [ ]:
```
#Correlation matrix for loan types only
corr_matrix_credit = credit_df[['Auto_Loan_Flag','Credit_Builder_Flag','Student_Loan_Flag'
                                 'Home_Equity_Flag','Debt_Consolidation_Flag','PayDay_F
sns.set(rc = {'figure.figsize':(20,8)})
sns.heatmap(corr_matrix_credit, cmap="YlGnBu", annot=True)
plt.title("Loan Correlation Matrix")
plt.show()
```

In [ ]:
```
#create an object of just the diffrent types of loan values
loan_values= credit_df[['Auto_Loan_Flag','Credit_Builder_Flag','Student_Loan_Flag','Mortag
                         'Home_Equity_Flag','Debt_Consolidation_Flag','PayDay_F
#develop histogram of loan types
loan_values.hist(figsize = (17,17));
```



In [ ]:

```
#view the mortgage loan distribution by occupation
ax = sns.barplot(x="Occupation", y="Mortagage_Loan_Flag", data= credit_df)
```



Binary Output Label

In [ ]:

```
# USING ENTIRE DATASET

pan = {'Poor' : 'No','Standard' : 'No',
       'Good': 'Yes'}

valuesCredit_Ad=credit_df['Credit_Ad'].value_counts()
propNo=(valuesCredit_Ad[0]/(valuesCredit_Ad[0]+valuesCredit_Ad[1]))*100
propYes=(valuesCredit_Ad[1]/(valuesCredit_Ad[0]+valuesCredit_Ad[1]))*100

print('proportion of No: ', propNo)
print("\n\nProportion of Yes: ", propYes)
```

```
proportion of No:  82.21806616582666


Proportion of Yes:  17.78193383417335
```

Look for any outliers/data that might not fit. Also, looking if any features that should be numerical, are not showing up due to dirty data:

In [ ]:

```
df_stat = credit_df.describe()
df_stat.loc['iqr'] = df_stat.apply(lambda x: x["75%"]-x["25%"])
df_stat
```

Out[ ]:

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	I
count	86596.000000	9.464100e+04	80454.000000	94641.000000	94641.000000	94641.000000	
mean	34.440771	1.757294e+05	4196.572372	5.404222	5.605446	14.531091	
std	10.150058	1.423242e+06	3186.064195	2.960657	2.976443	8.739028	
min	18.000000	7.005930e+03	303.645417	0.000000	0.000000	1.000000	
25%	26.000000	1.944141e+04	1625.265833	3.000000	4.000000	7.000000	
50%	34.000000	3.758034e+04	3096.066250	5.000000	5.000000	13.000000	
75%	42.000000	7.281486e+04	5961.745000	7.000000	7.000000	20.000000	
max	118.000000	2.419806e+07	15204.633333	99.000000	99.000000	34.000000	

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	I
iqr	16.000000	5.337345e+04	4336.479167	4.000000	3.000000	13.000000	

9 rows × 27 columns

In [ ]:

```
credit_df.isnull().sum()
```

Out[ ]:

```
ID                              0
Customer_ID                     0
Age                          8045
Occupation                      0
Annual_Income                   0
Monthly_Inhand_Salary       14187
Num_Bank_Accounts               0
Num_Credit_Card                 0
Interest_Rate                   0
Num_of_Loan                     0
Type_of_Loan                10783
Delay_from_due_date             0
Num_of_Delayed_Payment       6645
Changed_Credit_Limit         1976
Num_Credit_Inquiries         1846
Credit_Mix                  19129
Outstanding_Debt                0
Credit_Utilization_Ratio        0
Payment_of_Min_Amount           0
Total_EMI_per_month             0
Amount_invested_monthly      8327
Monthly_Balance              1147
Credit_Score                    0
Credit_Age_in_Years          8586
Mode_Behavior_binary        94641
Debt_to_Income_Ratio            0
Credit_Ad                       0
Auto_Loan_Flag                  0
Credit_Builder_Flag             0
Student_Loan_Flag               0
Mortagage_Loan_Flag             0
Home_Equity_Flag                0
Debt_Consolidation_Flag         0
PayDay_Flag                     0
Personal_Loan_Flag              0
dtype: int64
```

In [ ]:

```
# USING ENTIRE DATASET X and Y SPLITS

categorical_features_credit = ['Payment_of_Min_Amount','Auto_Loan_Flag',
                               'Credit_Builder_Flag',
                               'Student_Loan_Flag', 'Mortagage_Loan_Flag',
                               'Home_Equity_Flag','Debt_Consolidation_Flag',
                               'PayDay_Flag', 'Personal_Loan_Flag']
numerical_features_credit = ['Debt_to_Income_Ratio',
                             'Outstanding_Debt','Num_Bank_Accounts',
                             'Num_Credit_Card', 'Interest_Rate',
                             'Credit_Age_in_Years',
                             'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
                             'Num_Credit_Inquiries',
                             'Total_EMI_per_month', 'Amount_invested_monthly',
                             'Monthly_Balance', 'Num_Credit_Card','Num_Bank_Accounts',
                             'Num_of_Loan', 'Age'
                             ]
```

```
# Generate dataframes with the different features

Xcat_credit_df = credit_df[categorical_features_credit]

Xnum_credit_df = credit_df[numerical_features_credit]

# Combine the two in order to maintain indices, used merge function
# although it should be automatic even with the join command, as
# both come from the same dataframe with no modifications.

# Create X and y for splitting and further pre-processing:

X_credit_df = pd.merge(Xcat_credit_df, Xnum_credit_df, left_index=True,
                       right_index=True)

y_credit_df = credit_df[['Credit_Ad']]
```

In [ ]:

```
# Split data:
# Stratify tells us to randomly select samples for each train and test
# set while maintaining the proportions of Credit_Ad classes
# Since the original dataset contains about 20% yes and 80% No
# We should expect to see these same proportions in the ylabel for train
# and test:

X_train_credit, X_test_credit, y_train_credit, y_test_credit = train_test_split(X_credit_c
                                                                      y_credit_df,
                                                                      stratify = y_credi
                                                              test_size = 0.1,
                                                            random_state = 42)
```

In [ ]:

```
X_train_credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 85176 entries, 92663 to 51901
Data columns (total 25 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Payment_of_Min_Amount     85176 non-null  category
 1   Auto_Loan_Flag            85176 non-null  int64
 2   Credit_Builder_Flag       85176 non-null  int64
 3   Student_Loan_Flag         85176 non-null  int64
 4   Mortagage_Loan_Flag       85176 non-null  int64
 5   Home_Equity_Flag          85176 non-null  int64
 6   Debt_Consolidation_Flag   85176 non-null  int64
 7   PayDay_Flag               85176 non-null  int64
 8   Personal_Loan_Flag        85176 non-null  int64
 9   Debt_to_Income_Ratio      85176 non-null  float64
 10  Outstanding_Debt          85176 non-null  float64
 11  Num_Bank_Accounts         85176 non-null  int64
 12  Num_Credit_Card           85176 non-null  int64
 13  Interest_Rate             85176 non-null  int64
 14  Credit_Age_in_Years       77440 non-null  float64
 15  Num_of_Delayed_Payment    79174 non-null  float64
 16  Changed_Credit_Limit      83396 non-null  float64
 17  Num_Credit_Inquiries      83531 non-null  float64
 18  Total_EMI_per_month       85176 non-null  float64
 19  Amount_invested_monthly   77699 non-null  float64
 20  Monthly_Balance           84133 non-null  float64
 21  Num_Credit_Card           85176 non-null  int64
 22  Num_Bank_Accounts         85176 non-null  int64
 23  Num_of_Loan               85176 non-null  float64
 24  Age                       77955 non-null  float64
```

```
dtypes: category(1), float64(11), int64(13)
memory usage: 16.3 MB
```

And indeed, we see the same yes and no class proportions in training and test y label sets:

In [ ]:
```
y_train_credit.value_counts()
```

Out[ ]:
```
Credit_Ad
No         70030
Yes        15146
dtype: int64
```

In [ ]:
```
y_test_credit.value_counts()
```

Out[ ]:
```
Credit_Ad
No         7782
Yes        1683
dtype: int64
```

Null Imputation

In [ ]:
```
# ENTIRE DATASET

#X_train_credit['Monthly_Inhand_Salary'] = X_train_credit.Monthly_Inhand_Salary.fillna(X_t
X_train_credit['Credit_Age_in_Years'] = X_train_credit.Credit_Age_in_Years.fillna(X_train_
X_train_credit['Num_of_Delayed_Payment'] = X_train_credit.Num_of_Delayed_Payment.fillna(X_
X_train_credit['Num_Credit_Inquiries'] = X_train_credit.Num_Credit_Inquiries.fillna(X_trai
X_train_credit['Monthly_Balance'] = X_train_credit.Monthly_Balance.fillna(X_train_credit['
X_train_credit['Amount_invested_monthly'] = X_train_credit.Amount_invested_monthly.fillna
X_train_credit['Changed_Credit_Limit'] = X_train_credit.Changed_Credit_Limit.fillna(X_trai
X_train_credit['Age'] = X_train_credit.Age.fillna(X_train_credit['Age'].median())


X_train_credit_imp_df = X_train_credit.reset_index()

#X_test_credit['Monthly_Inhand_Salary'] = X_test_credit.Monthly_Inhand_Salary.fillna(X_tes
impute_value_CAIY = X_train_credit['Credit_Age_in_Years'].median()
impute_value_TA = X_train_credit['Num_of_Delayed_Payment'].median()
impute_value_TA = X_train_credit['Num_Credit_Inquiries'].median()
impute_value_TA = X_train_credit['Monthly_Balance'].median()
impute_value_TA = X_train_credit['Amount_invested_monthly'].median()
impute_value_TA = X_train_credit['Changed_Credit_Limit'].median()
impute_value_TA = X_train_credit['Age'].median()


X_test_credit['Credit_Age_in_Years'] = X_test_credit.Credit_Age_in_Years.fillna(impute_val
X_test_credit['Num_of_Delayed_Payment'] = X_test_credit.Num_of_Delayed_Payment.fillna(impu
X_test_credit['Num_Credit_Inquiries'] = X_test_credit.Num_Credit_Inquiries.fillna(impute_v
X_test_credit['Monthly_Balance'] = X_test_credit.Monthly_Balance.fillna(impute_value_TA)
X_test_credit['Amount_invested_monthly'] = X_test_credit.Amount_invested_monthly.fillna(in
X_test_credit['Changed_Credit_Limit'] = X_test_credit.Changed_Credit_Limit.fillna(impute_v
X_test_credit['Age'] = X_test_credit.Age.fillna(impute_value_TA)


X_test_credit_imp_df = X_test_credit.reset_index()
```

In [ ]:
```
X_train_credit_imp_df.head(8)
```

Out[ ]:

	index	Payment_of_Min_Amount	Auto_Loan_Flag	Credit_Builder_Flag	Student_Loan_Flag	Mortagage_Loan_Flag	H
0	92663	NM	1	0	0	0	
1	12975	Yes	0	0	1	1	
2	36816	Yes	0	0	0	1	
3	22205	No	0	0	0	0	
4	77679	No	0	0	1	1	
5	44420	Yes	1	0	0	0	
6	86269	Yes	0	1	1	0	
7	10576	No	0	0	1	0	

8 rows × 26 columns

In [ ]:

```
#check for NULL values after imputation
X_train_credit_imp_df.isnull().sum()
```

Out[ ]:

```
index                      0
Payment_of_Min_Amount      0
Auto_Loan_Flag             0
Credit_Builder_Flag        0
Student_Loan_Flag          0
Mortagage_Loan_Flag        0
Home_Equity_Flag           0
Debt_Consolidation_Flag    0
PayDay_Flag                0
Personal_Loan_Flag         0
Debt_to_Income_Ratio       0
Outstanding_Debt           0
Num_Bank_Accounts          0
Num_Credit_Card            0
Interest_Rate              0
Credit_Age_in_Years        0
Num_of_Delayed_Payment     0
Changed_Credit_Limit       0
Num_Credit_Inquiries       0
Total_EMI_per_month        0
Amount_invested_monthly    0
Monthly_Balance            0
Num_Credit_Card            0
Num_Bank_Accounts          0
Num_of_Loan                0
Age                        0
dtype: int64
```

Scaling and Encoding Feature Set

In [ ]:

```
# Pre-Process Training data categorical variables and
# scale/normalize numericals by column

X_train_credit_num = X_train_credit_imp_df[numerical_features_credit]
X_test_credit_num =  X_test_credit_imp_df[numerical_features_credit]

sc = preprocessing.StandardScaler()
sc_fitted = sc.fit(X_train_credit_num)

X_train_credit_sc = pd.DataFrame(sc_fitted.transform(X_train_credit_num),
                                columns=X_train_credit_num.columns)
```

```
X_test_credit_sc = pd.DataFrame(sc_fitted.transform(X_test_credit_num),
                                columns=X_test_credit_num.columns)
```

In [ ]:

```
X_train_credit_imp_df[categorical_features_credit].head()
```

Out[ ]:

	Payment_of_Min_Amount	Auto_Loan_Flag	Credit_Builder_Flag	Student_Loan_Flag	Mortagage_Loan_Flag	Home_Eq
0	NM	1	0	0	0	
1	Yes	0	0	1	1	
2	Yes	0	0	0	1	
3	No	0	0	0	0	
4	No	0	0	1	1	

In [ ]:

```
# Hot encode categorical features:


X_train_credit_cat = X_train_credit_imp_df[categorical_features_credit]
X_test_credit_cat =  X_test_credit_imp_df[categorical_features_credit]


enc_hot = OneHotEncoder(categories = 'auto', sparse=False)

cat_feat = enc_hot.fit(X_train_credit_cat)



X_train_credit_enc = pd.DataFrame(cat_feat.transform(X_train_credit_cat),
                                  columns=cat_feat.get_feature_names(categorical_features_cre
X_test_credit_enc = pd.DataFrame(cat_feat.transform(X_test_credit_cat),
                                  columns=cat_feat.get_feature_names(categorical_features_cre
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Fun
ction get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be
removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Fun
ction get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be
removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

In [ ]:

```
#check the unique values from the Occupation attribute
#X_train_credit_imp_df['Occupation'].unique()
```

In [ ]:

```
X_train_credit_enc.head()
```

Out[ ]:

	Payment_of_Min_Amount_NM	Payment_of_Min_Amount_No	Payment_of_Min_Amount_Yes	Auto_Loan_Flag_0	Auto_
0	1.0	0.0	0.0	0.0	
1	0.0	0.0	1.0	1.0	
2	0.0	0.0	1.0	1.0	
3	0.0	1.0	0.0	1.0	
4	0.0	1.0	0.0	1.0	

In [ ]:
```
# Combine the two dataframes for scaled and encoded
# numerical and categorical features:

X_train_credit_pre = pd.merge(X_train_credit_sc, X_train_credit_enc, left_index=True,
                       right_index=True)

X_test_credit_pre = pd.merge(X_test_credit_sc, X_test_credit_enc, left_index=True,
                       right_index=True)
```

In [ ]:
```
# change y_train shape
y_train_credit = np.ravel(y_train_credit)
y_test_credit = np.ravel(y_test_credit)
```

Modeling

Balanced Perceptron model

Model Creation

In [ ]:
```
balanced_model = make_pipeline(Perceptron(class_weight='balanced'))
unbalanced_model = make_pipeline(Perceptron())
```

Model Fitting

In [ ]:
```
balanced_mod = balanced_model.fit(X_train_credit_pre, y_train_credit)
balanced_mod_pred_train = balanced_mod.predict(X_train_credit_pre)
balanced_mod_pred = balanced_mod.predict(X_test_credit_pre)
```

Model Validation

Cross Validation

In [ ]:
```
bal_percep_cv = cross_val_score(balanced_mod, X_train_credit_pre, y_train_credit, cv=5)
```

In [ ]:
```
print(bal_percep_cv)

print('Average', bal_percep_cv.mean().round(2))
```

```
[0.64768725 0.66521867 0.70589962 0.63891987 0.67061931]
Average 0.67
```

Confusion Matrix

In [ ]:
```
cm_mod_train_bal = confusion_matrix(y_train_credit,
                      balanced_mod_pred_train)

cm_mod_test_bal = confusion_matrix(y_test_credit,
                      balanced_mod_pred)
```

In [ ]:
```
labels_mod = ['No', 'Yes']
```

```
train_results_bal = pd.DataFrame(cm_mod_train_bal, index = labels_mod,
                                  columns = labels_mod)

test_results_bal = pd.DataFrame(cm_mod_test_bal, index = labels_mod,
                                 columns = labels_mod)
```

In [ ]:

```
# create df for test accuracy scores
accuracy_score_table = pd.DataFrame(columns=['Model', 'Accuracy Score'])
```

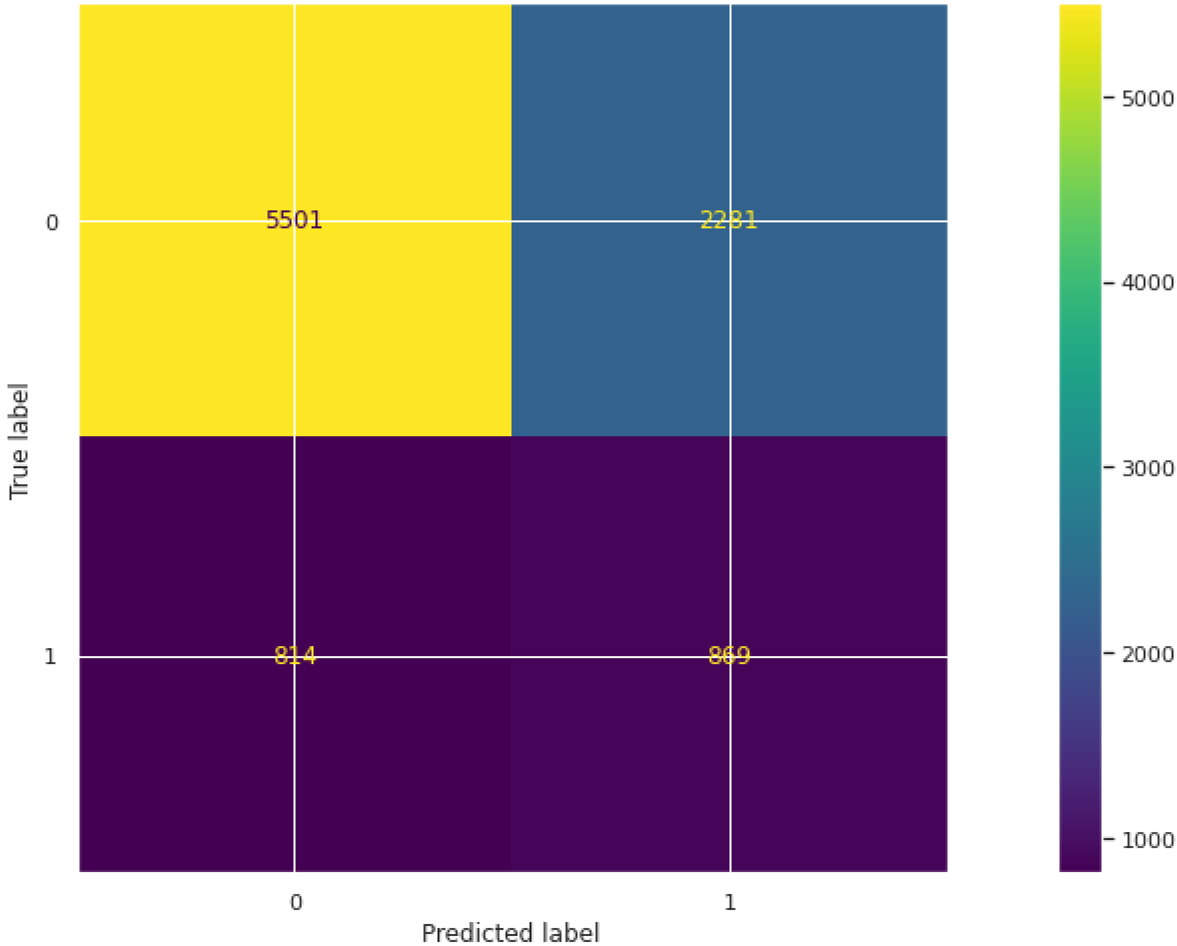
In [ ]:

```
print('Train Data Confusion Matrix for Credit Data on Balanced Classifier')
print(train_results_bal)
print('\nTrain Accuracy for Credit Data on Balanced Classifier:\n')
print('\t\t\t\t', accuracy_score(y_train_credit, balanced_mod_pred_train))
print('\n')

print('Test Data Confusion Matrix for Credit Data on Balanced Classifier')
print(test_results_bal)
print('\nTest Accuracy for Credit data on Balanced Classifier:\n')
print('\t\t\t\t', accuracy_score(y_test_credit, balanced_mod_pred))
print('\n')

# save accuracy score
bal_acc = accuracy_score(y_test_credit, balanced_mod_pred)
accuracy_score_table.loc[len(accuracy_score_table.index)] = ['Balanced Perceptron', bal_ac
```

```
Train Data Confusion Matrix for Credit Data on Balanced Classifier
        No    Yes
No   49000  21030
Yes   7132   8014

Train Accuracy for Credit Data on Balanced Classifier:

                                   0.6693669578284963


Test Data Confusion Matrix for Credit Data on Balanced Classifier
       No   Yes
No   5501  2281
Yes   814   869

Test Accuracy for Credit data on Balanced Classifier:

                                   0.6730058108821976
```

In [ ]:

```
#create a confusion matrix for display of the balanced perceptron
confusion_matrix_plot = metrics.confusion_matrix(y_test_credit,balanced_mod_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix_plot)
cm_display.plot()
plt.show()
```



Classification Report

In [ ]:
```
print('\nClassification Report for Credit Data on Balanced Classifier:\n')
print(classification_report(y_test_credit, balanced_mod_pred))
print('\n')
```

```
Classification Report for Credit Data on Balanced Classifier:

              precision    recall  f1-score   support

          No       0.87      0.71      0.78      7782
         Yes       0.28      0.52      0.36      1683

    accuracy                           0.67      9465
   macro avg       0.57      0.61      0.57      9465
weighted avg       0.77      0.67      0.71      9465
```

Unbalanced Perceptron Model

Model Fitting

In [ ]:
```
unbalanced_mod = unbalanced_model.fit(X_train_credit_pre, y_train_credit)
unbalanced_mod_pred_train = unbalanced_mod.predict(X_train_credit_pre)
unbalanced_mod_pred = unbalanced_mod.predict(X_test_credit_pre)
```

In [ ]:
```
unbalanced_pred_train = pd.DataFrame(unbalanced_mod_pred_train)
y_train_credit_df = pd.DataFrame(y_train_credit)
```

Model Validation

Cross Validation

In [ ]:
```
unbal_percep_cv = cross_val_score(unbalanced_mod, X_train_credit_pre, y_train_credit, cv=5
```

In [ ]:
```
print(unbal_percep_cv)

print('Average', unbal_percep_cv.mean().round(2))
```

```
[0.78234327 0.70032286 0.81702377 0.73595539 0.79236865]
Average 0.77
```

Confusion Matrix

In [ ]:
```
cm_mod_train_unbal = confusion_matrix(y_train_credit, unbalanced_mod_pred_train)

cm_mod_test_unbal = confusion_matrix(y_test_credit, unbalanced_mod_pred)
```

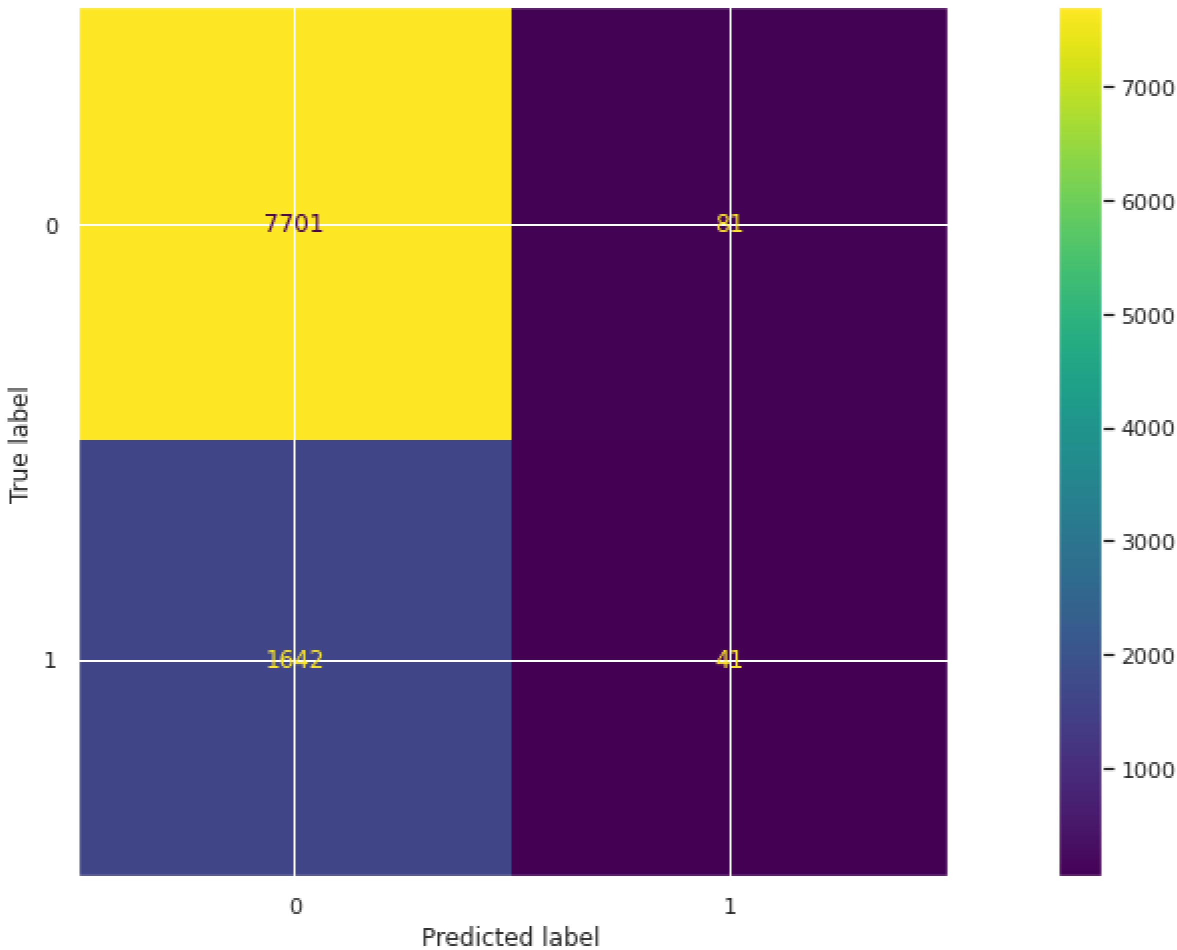
In [ ]:
```
train_results_unbal = pd.DataFrame(cm_mod_train_unbal, index = labels_mod,
                                  columns = labels_mod)

test_results_unbal = pd.DataFrame(cm_mod_test_unbal, index = labels_mod,
                                  columns = labels_mod)
```

In [ ]:
```
print('Train Data Confusion Matrix for Credit Data on Unbalanced Classifier')
print(train_results_unbal)
print('\nTrain Accuracy for Credit Data on Unbalanced Classifier:\n')
print('\t\t\t\t', accuracy_score(y_train_credit, unbalanced_mod_pred_train))
print('\n')

print('Test Data Confusion Matrix for Credit Data on Unbalanced Classifier')
print(test_results_unbal)
print('\nTest Accuracy for Credit data on Unbalanced Classifier:\n')
print('\t\t\t\t', accuracy_score(y_test_credit, unbalanced_mod_pred))
print('\n')

unbal_acc = accuracy_score(y_test_credit, unbalanced_mod_pred)
accuracy_score_table.loc[len(accuracy_score_table.index)] = ['Unbalanced Perceptron', unba
```

```
Train Data Confusion Matrix for Credit Data on Unbalanced Classifier
        No  Yes
No   69238  792
Yes  14745  401

Train Accuracy for Credit Data on Unbalanced Classifier:

                                 0.8175894618202311


Test Data Confusion Matrix for Credit Data on Unbalanced Classifier
       No  Yes
No   7701   81
Yes  1642   41

Test Accuracy for Credit data on Unbalanced Classifier:

                                 0.817960908610671
```

In [ ]:

```
#create a confusion matrix for display of the unbalanced perceptron
confusion_matrix_plot = metrics.confusion_matrix(y_test_credit,unbalanced_mod_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix_plot)
cm_display.plot()
plt.show()
```



Classification Report

In [ ]:

```
print('\nClassification Report for Credit Data on Unbalanced Classifier:\n')
print(classification_report(y_test_credit, unbalanced_mod_pred))
print('\n')
```

```
Classification Report for Credit Data on Unbalanced Classifier:

              precision    recall  f1-score   support

          No       0.82      0.99      0.90      7782
         Yes       0.34      0.02      0.05      1683

    accuracy                           0.82      9465
   macro avg       0.58      0.51      0.47      9465
weighted avg       0.74      0.82      0.75      9465
```

Neural Network

Model Fitting

In [ ]:

```
# # Loop through different alphas and hidden_layer
# # tuples and find best cross validated accuracy value:

# alpha22 = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1]
# activation=['relu', 'tanh']
# layers=[1,2,5,10,15,20]
# units=[2,4,6,8,10]
# randomsize=[1,5,42]
# NNscores_list = []
# NNavg_list = []

# for i in range(0,6):
#   for t in range(0,6):
#     for n in range(0,5):
#       nn = MLPClassifier(solver='sgd', alpha=alpha22[i],
#                          hidden_layer_sizes=(layers[t], units[n]), random_state=42)
#       cv = np.mean(cross_val_score(nn, X_train_credit_pre, y_train_credit, cv=5, scoring
#       NNscores_list.append({'alpha': alpha22[i], 'layers': layers[t],
#                             'units': units[n], 'cv': cv})
```

In [ ]:

```
#NNscores=pd.DataFrame(NNscores_list, columns=['alpha','layers','units', 'cv'])
```

In [ ]:

```
#NNscores.max()
# maximum cv accuracy was 0.8151
```

In [ ]:

```
#NNscores.idxmax()
# maximum cv values is at index 51
```

In [ ]:

```
#NNscores.head(60)
```

In [ ]:

```
nn = MLPClassifier(solver='lbfgs', alpha=1e-5,
                     hidden_layer_sizes=(5, 2), random_state=42)

nn_model = nn.fit(X_train_credit_pre, y_train_credit)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:54
9: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

In [ ]:

```
nn_train_pred = nn.predict(X_train_credit_pre)
nn_test_pred = nn.predict(X_test_credit_pre)
```

Model Validation

Cross Validation

In [ ]:

```
nn_cv = cross_val_score(nn_model, X_train_credit_pre, y_train_credit, cv=5)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:54
9: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:54
9: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:54
9: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:54
9: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:54
9: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

In [ ]:

```
print(nn_cv)

print('Average', nn_cv.mean().round(2))
```

```
[0.84526884 0.84602289 0.83933079 0.8395656  0.84572938]
Average 0.84
```

Confusion Matrix

In [ ]:

```
cm_mod_train_nn = confusion_matrix(y_train_credit,
                      nn_train_pred)

cm_mod_test_nn = confusion_matrix(y_test_credit,
                      nn_test_pred)
```

In [ ]:

```
labels_mod = ['Yes', 'No']

train_results_nn = pd.DataFrame(cm_mod_train_nn, index = labels_mod,
                                columns = labels_mod)

test_results_nn = pd.DataFrame(cm_mod_test_nn, index = labels_mod,
                                columns = labels_mod)
```

In [ ]:

```
print('Train Data Confusion Matrix for Credit Data on NN Classifier')
print(train_results_nn)
print('\nTrain Accuracy for Credit Data on NN Classifier:\n')
```

```
print('\t\t\t\t', accuracy_score(y_train_credit, nn_train_pred))
print('\n')

print('Test Data Confusion Matrix for Credit Data on NN Classifier')
print(test_results_nn)
print('\nTest Accuracy for Credit data on NN Classifier:\n')
print('\t\t\t\t', accuracy_score(y_test_credit, nn_test_pred))
print('\n')

nn_acc = accuracy_score(y_test_credit, nn_test_pred)
accuracy_score_table.loc[len(accuracy_score_table.index)] = ['Neural Network', nn_acc]
```

```
Train Data Confusion Matrix for Credit Data on NN Classifier
       Yes    No
Yes  66191  3839
No    9835  5311

Train Accuracy for Credit Data on NN Classifier:

                                  0.839461820231051


Test Data Confusion Matrix for Credit Data on NN Classifier
      Yes   No
Yes  7359  423
No   1118  565

Test Accuracy for Credit data on NN Classifier:

                                  0.837189646064448
```

Classification Report

In [ ]:

```
print('\nClassification Report for Credit Data on NN Classifier:\n')
print(classification_report(y_test_credit, nn_test_pred))
print('\n')
```

```
Classification Report for Credit Data on NN Classifier:

              precision    recall  f1-score   support

          No       0.87      0.95      0.91      7782
         Yes       0.57      0.34      0.42      1683

    accuracy                           0.84      9465
   macro avg       0.72      0.64      0.66      9465
weighted avg       0.82      0.84      0.82      9465
```

Decision Tree Classifier

Model Fitting

In [ ]:

```
#put the from at top of data
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf_model = clf.fit(X_train_credit_pre, y_train_credit)
y_pred_dt_train = clf_model.predict(X_train_credit_pre)
```

```
y_pred_dt = clf_model.predict(X_test_credit_pre)
print('Test accuracy %2.2f ' % accuracy_score(y_test_credit,y_pred_dt))
```

```
Test accuracy 0.86
```

Model Validation

Cross Validation

In [ ]:
```
clf_cv = cross_val_score(clf_model, X_train_credit_pre, y_train_credit, cv=5)
```

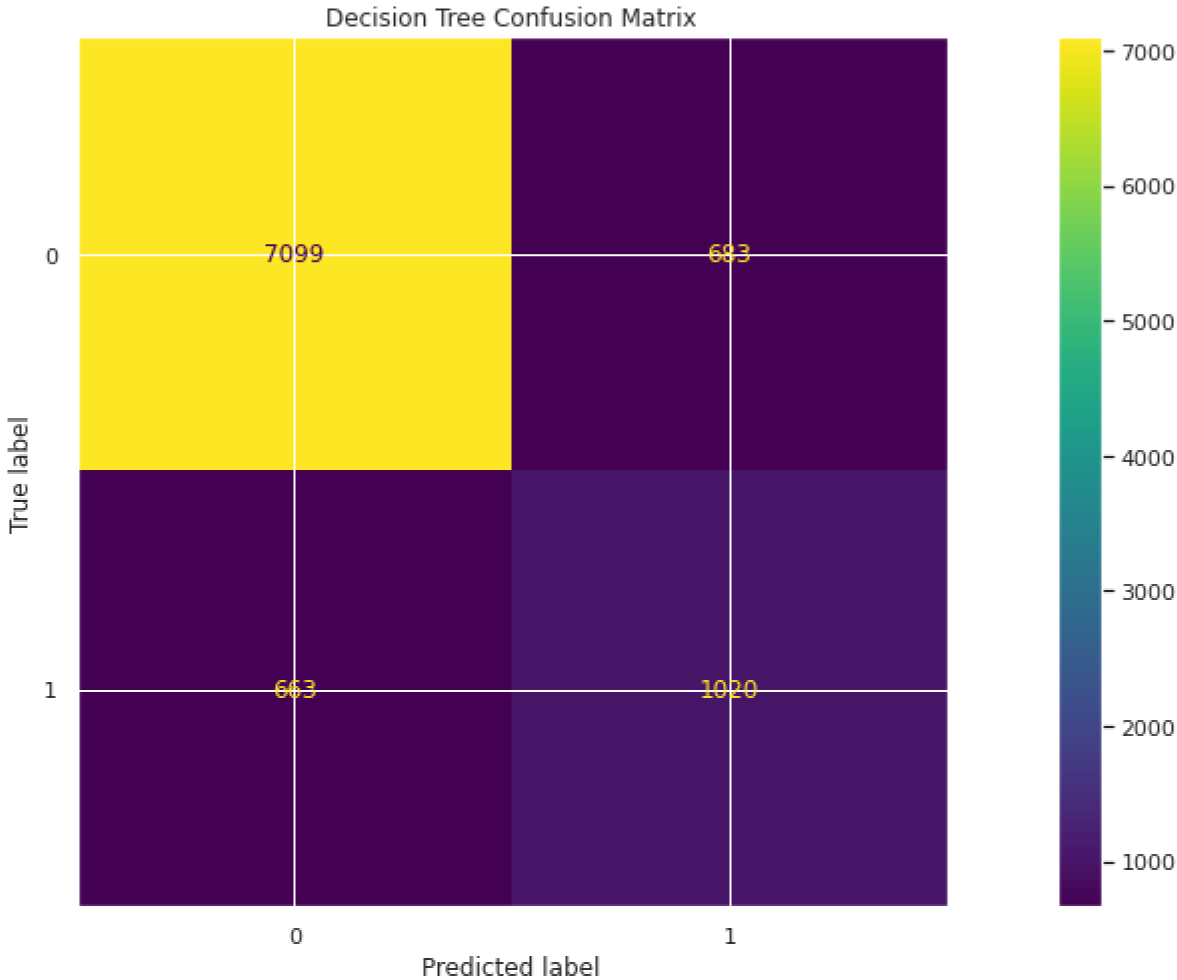
In [ ]:
```
print(clf_cv)

print('Average', clf_cv.mean().round(2))
```

```
[0.863583   0.86140299 0.85999413 0.86322278 0.8636337 ]
Average 0.86
```

Confusion Matrix

In [ ]:
```
cm_mod_train_dt = confusion_matrix(y_train_credit, y_pred_dt_train)

cm_mod_test_dt = confusion_matrix(y_test_credit,y_pred_dt)

labels_mod = ['Deny', 'Approve']

train_results_lr = pd.DataFrame(cm_mod_train_dt, index = labels_mod,
                                columns = labels_mod)

test_results_dt = pd.DataFrame(cm_mod_test_dt, index = labels_mod,
                                 columns = labels_mod)
```

In [ ]:
```
#create a confusion matrix for display of the decision tree model
confusion_matrix_plot = metrics.confusion_matrix(y_test_credit,y_pred_dt)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix_plot)
cm_display.plot()
plt.title("Decision Tree Confusion Matrix")
plt.show()
```



In [ ]:

```
print('Train Data Confusion Matrix for Credit Data on Balanced Classifier')
print(train_results_bal)
print('\nTrain Accuracy for Credit Data on Balanced Classifier:\n')
print('\t\t\t\t', accuracy_score(y_train_credit, balanced_mod_pred_train))
print('\n')

print('Test Data Confusion Matrix for Credit Data on Decision Tree Classifier')
print(test_results_dt)
print('\nTest Accuracy for Credit data on Decision Tree Classifier:\n')
print('\t\t\t\t', accuracy_score(y_test_credit, y_pred_dt))
print('\n')

DT_acc = accuracy_score(y_test_credit, y_pred_dt)
accuracy_score_table.loc[len(accuracy_score_table.index)] = ['Decision Tree', DT_acc]
```

```
Train Data Confusion Matrix for Credit Data on Balanced Classifier
        No    Yes
No   49000  21030
Yes   7132   8014

Train Accuracy for Credit Data on Balanced Classifier:

                                  0.6693669578284963


Test Data Confusion Matrix for Credit Data on Decision Tree Classifier
         Deny  Approve
Deny     7099      683
Approve   663     1020

Test Accuracy for Credit data on Decision Tree Classifier:
```

0.8577918647649234

Classification Report

In [ ]:
```
print('\nClassification Report for Credit Data on Decision Tree Classifier:\n')
print(classification_report(y_test_credit, y_pred_dt))
print('\n')
```

```
Classification Report for Credit Data on Decision Tree Classifier:

              precision    recall  f1-score   support

          No       0.91      0.91      0.91      7782
         Yes       0.60      0.61      0.60      1683

    accuracy                           0.86      9465
   macro avg       0.76      0.76      0.76      9465
weighted avg       0.86      0.86      0.86      9465
```

Random Forest Model

Model Fitting

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

# Initial model
rf = RandomForestClassifier(n_estimators = 500, n_jobs = -1, random_state = 42)
rf_model = rf.fit(X_train_credit_pre, y_train_credit)
rf_train_pred = rf.predict(X_train_credit_pre)
rf_test_pred = rf.predict(X_test_credit_pre)
```

Model Validation

Cross Validation

In [ ]:
```
rf_cv = cross_val_score(rf_model, X_train_credit_pre, y_train_credit, cv=5)
```

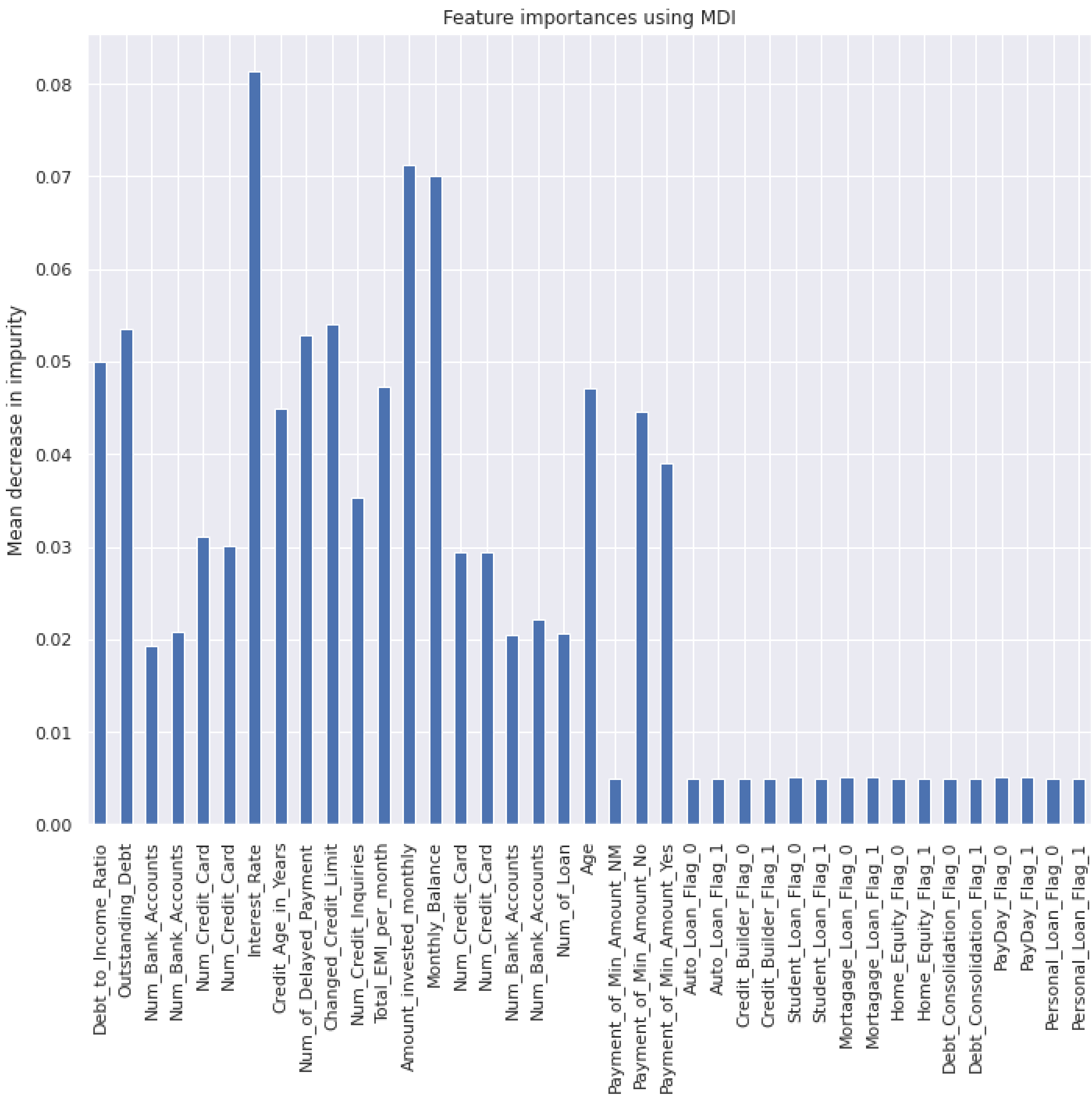
In [ ]:
```
print(rf_cv)

print('Average', rf_cv.mean().round(2))
```

```
[0.91142287 0.91147637 0.907602   0.91077194 0.91077194]
Average 0.91
```

RF Importance Plot

In [ ]:
```
importances = rf_model.feature_importances_
feature_names = X_train_credit_pre.columns
forest_importances = pd.Series(importances, index=feature_names)

fig, ax = plt.subplots(figsize=(10,10))
forest_importances.plot.bar()
ax.set_title("Feature importances using MDI")
```

```
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```



Confusion Matrix

In [ ]:

```
cm_mod_train_rf = confusion_matrix(y_train_credit,
                                   rf_train_pred)

cm_mod_test_rf = confusion_matrix(y_test_credit,
                                  rf_test_pred)
```

In [ ]:

```
train_results_rf = pd.DataFrame(cm_mod_train_rf, index = labels_mod,
                                columns = labels_mod)

test_results_rf = pd.DataFrame(cm_mod_test_rf, index = labels_mod,
                               columns = labels_mod)
```

In [ ]:
```
print('Train Data Confusion Matrix for Credit Data on RF Classifier')
print(train_results_rf)
print('\nTrain Accuracy for Credit Data on RF Classifier:\n')
print('\t\t\t\t', accuracy_score(y_train_credit, rf_train_pred))
print('\n')

print('Test Data Confusion Matrix for Credit Data on RF Classifier')
print(test_results_rf)
print('\nTest Accuracy for Credit data on RF Classifier:\n')
print('\t\t\t\t', accuracy_score(y_test_credit, rf_test_pred))
print('\n')

RF_acc = accuracy_score(y_test_credit, rf_test_pred)
accuracy_score_table.loc[len(accuracy_score_table.index)] = ['Random Forests', RF_acc]
```

```
Train Data Confusion Matrix for Credit Data on RF Classifier
          Deny  Approve
Deny     70030        0
Approve      0    15146

Train Accuracy for Credit Data on RF Classifier:

                                 1.0


Test Data Confusion Matrix for Credit Data on RF Classifier
         Deny  Approve
Deny     7435      347
Approve   558     1125

Test Accuracy for Credit data on RF Classifier:

                                 0.9043845747490755
```

Classification Report

In [ ]:
```
print('\nClassification Report for Credit Data on RF Classifier:\n')
print(classification_report(y_test_credit, rf_test_pred))
print('\n')
```

```
Classification Report for Credit Data on RF Classifier:

              precision    recall  f1-score   support

          No       0.93      0.96      0.94      7782
         Yes       0.76      0.67      0.71      1683

    accuracy                           0.90      9465
   macro avg       0.85      0.81      0.83      9465
weighted avg       0.90      0.90      0.90      9465
```

KNN Model

Euclidean Distance

Model Fitting

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
```

```
k_values = range(1,20,2)
metric = "euclidean"
knn_accuracy = []
clfs =[]

def k_neighbors(X_train_credit_pre, y_train_credit_enc, X_test_credit_pre,
                y_test_credit_enc, kvalues, metric):
  for i in kvalues:
    clf = KNeighborsClassifier(metric=metric,p=2, n_neighbors=i).fit(X_train_credit_pre,
                                                                     y_train_credit)
    clf_train_pred = clf.predict(X_train_credit_pre)
    clf_test_pred = clf.predict(X_test_credit_pre)
    clfs.append(clf)
    print(i)
    knn_accuracy.append({'k values': i,
    'Training Accuracy':accuracy_score(clf_train_pred, y_train_credit_enc),
    'Test Accuracy': accuracy_score(clf_test_pred, y_test_credit_enc)})

  return pd.DataFrame(knn_accuracy,clfs)

knn_df = k_neighbors(X_train_credit_pre, y_train_credit, X_test_credit_pre,
                     y_test_credit, k_values, metric)
knn_df
```

```
1
3
5
7
9
11
13
15
17
19
```

Out[ ]:

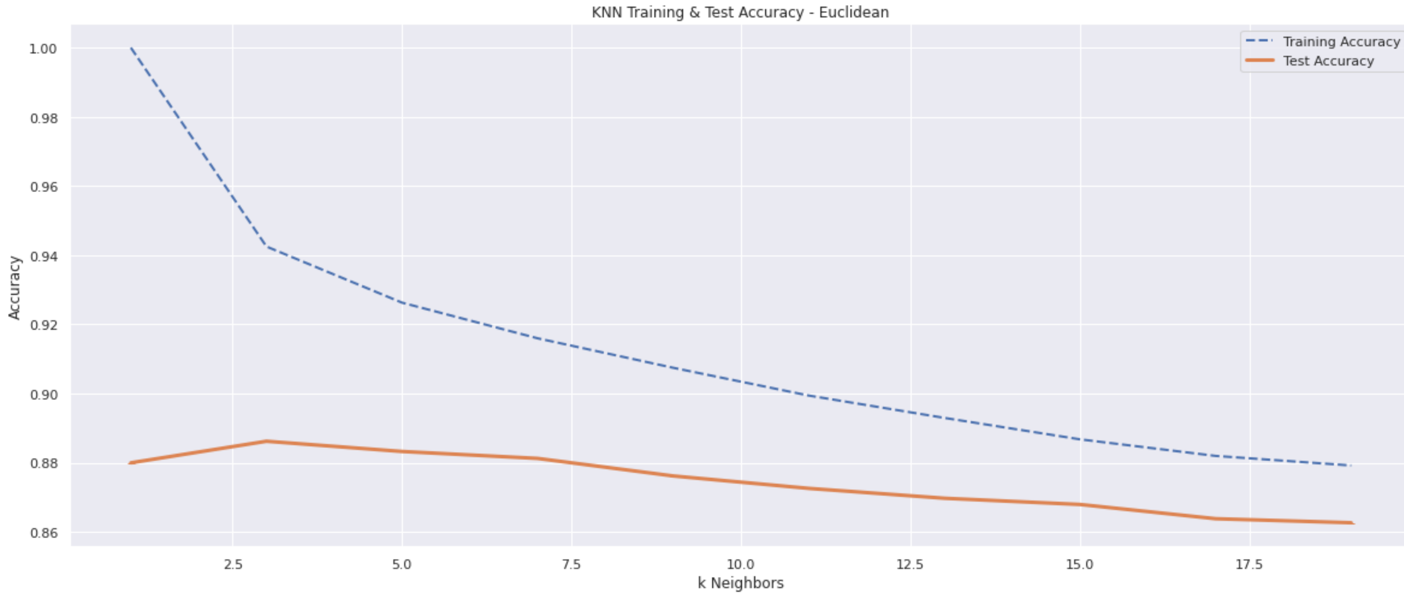
	k values	Training Accuracy	Test Accuracy
KNeighborsClassifier(metric='euclidean', n_neighbors=1)	1	1.000000	0.879979
KNeighborsClassifier(metric='euclidean', n_neighbors=3)	3	0.942507	0.886212
KNeighborsClassifier(metric='euclidean')	5	0.926282	0.883254
KNeighborsClassifier(metric='euclidean', n_neighbors=7)	7	0.915974	0.881247
KNeighborsClassifier(metric='euclidean', n_neighbors=9)	9	0.907450	0.876175
KNeighborsClassifier(metric='euclidean', n_neighbors=11)	11	0.899420	0.872583
KNeighborsClassifier(metric='euclidean', n_neighbors=13)	13	0.892963	0.869731
KNeighborsClassifier(metric='euclidean', n_neighbors=15)	15	0.886764	0.867934
KNeighborsClassifier(metric='euclidean', n_neighbors=17)	17	0.881974	0.863814
KNeighborsClassifier(metric='euclidean', n_neighbors=19)	19	0.879203	0.862652

Model Validation

In [ ]:

```
plt.plot(knn_df['k values'], knn_df['Training Accuracy'], '--',linewidth=2, label='Trainir
plt.plot(knn_df['k values'], knn_df['Test Accuracy'], '-_',linewidth=3, label='Test Accura

plt.xlabel('k Neighbors')
plt.ylabel('Accuracy')
plt.title('KNN Training & Test Accuracy - Euclidean')
```

```
plt.legend()
plt.show()
```



Identifying the best performing model

In [ ]:

```
#return the results of the most accurate KNN model - Euclidean
knn_accuracy = KNeighborsClassifier(metric=metric,p=2, n_neighbors=3).fit(X_train_credit_p
                                                                  y_train_credit)
knn_model = knn_accuracy.fit(X_train_credit_pre, y_train_credit)
y_pred_knn_euc = knn_model.predict(X_test_credit_pre)
y_pred_knn_euc_train = knn_model.predict(X_train_credit_pre)

print('accuracy %2.2f ' % accuracy_score(y_test_credit,y_pred_knn_euc))

euc_acc = accuracy_score(y_test_credit, y_pred_knn_euc)
accuracy_score_table.loc[len(accuracy_score_table.index)] = ['KNN Euc', euc_acc]
```

```
accuracy 0.89
```

Cross Validation

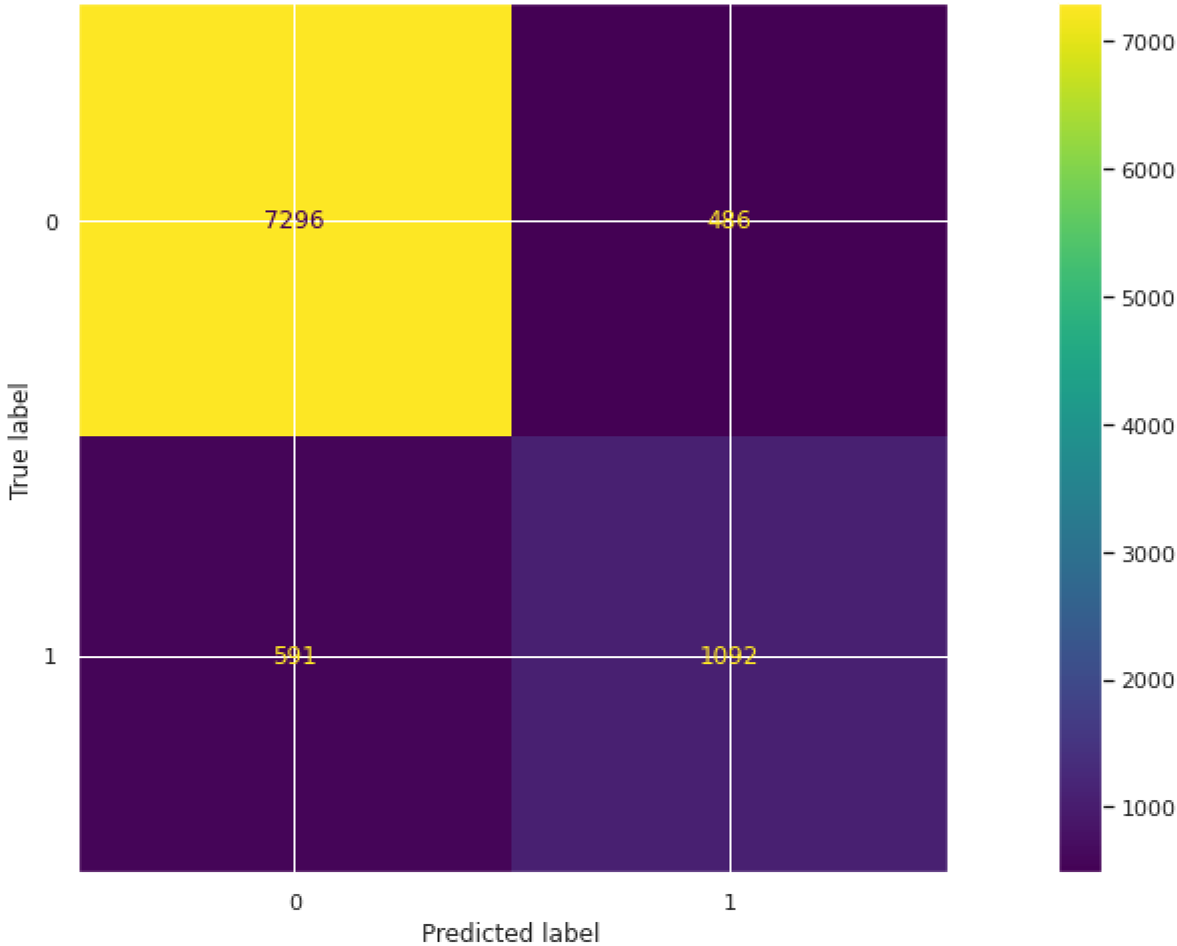
Confusion Matrix

In [ ]:

```
cm_mod_train_lr = confusion_matrix(y_train_credit, y_pred_knn_euc_train)
cm_mod_test_knn = confusion_matrix(y_test_credit,y_pred_knn_euc)
labels_mod = ['Yes','No']
train_results_lr = pd.DataFrame(cm_mod_train_lr, index = labels_mod,
                                  columns = labels_mod)
test_results_knn = pd.DataFrame(cm_mod_test_knn, index = labels_mod,
                                  columns = labels_mod)
```

In [ ]:

```
confusion_matrix_plot = metrics.confusion_matrix(y_test_credit,y_pred_knn_euc)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix_plot)
cm_display.plot()
plt.show()
```



Classification Report

In [ ]:
```
print('\nClassification Report for Credit Data on KNN - Euclidean Classifier:\n')
print(classification_report(y_test_credit, y_pred_knn_euc))
print('\n')
```

```
Classification Report for Credit Data on KNN - Euclidean Classifier:

              precision    recall  f1-score   support

          No       0.93      0.94      0.93      7782
         Yes       0.69      0.65      0.67      1683

    accuracy                           0.89      9465
   macro avg       0.81      0.79      0.80      9465
weighted avg       0.88      0.89      0.88      9465
```

Manhattan Distance

Model Fitting

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
k_values = range(1,20,2)
metric = "manhattan"
knn_accuracy = []
clfs =[]

def k_neighbors(X_train_credit_pre, y_train_credit, X_test_credit_pre,
                y_test_credit, kvalues, metric):
```

```
    for i in kvalues:
      clf = KNeighborsClassifier(metric=metric,p=2, n_neighbors=i).fit(X_train_credit_pre,
                                                                      y_train_credit)
      clf_train_pred = clf.predict(X_train_credit_pre)
      clf_test_pred = clf.predict(X_test_credit_pre)
      clfs.append(clf)
      knn_accuracy.append({'k values': i,
      'Training Accuracy':accuracy_score(clf_train_pred, y_train_credit),
      'Test Accuracy': accuracy_score(clf_test_pred, y_test_credit)})
    return pd.DataFrame(knn_accuracy,clfs)

knn_df_man = k_neighbors(X_train_credit_pre, y_train_credit, X_test_credit_pre,
                          y_test_credit, k_values, metric)
knn_df_man
```

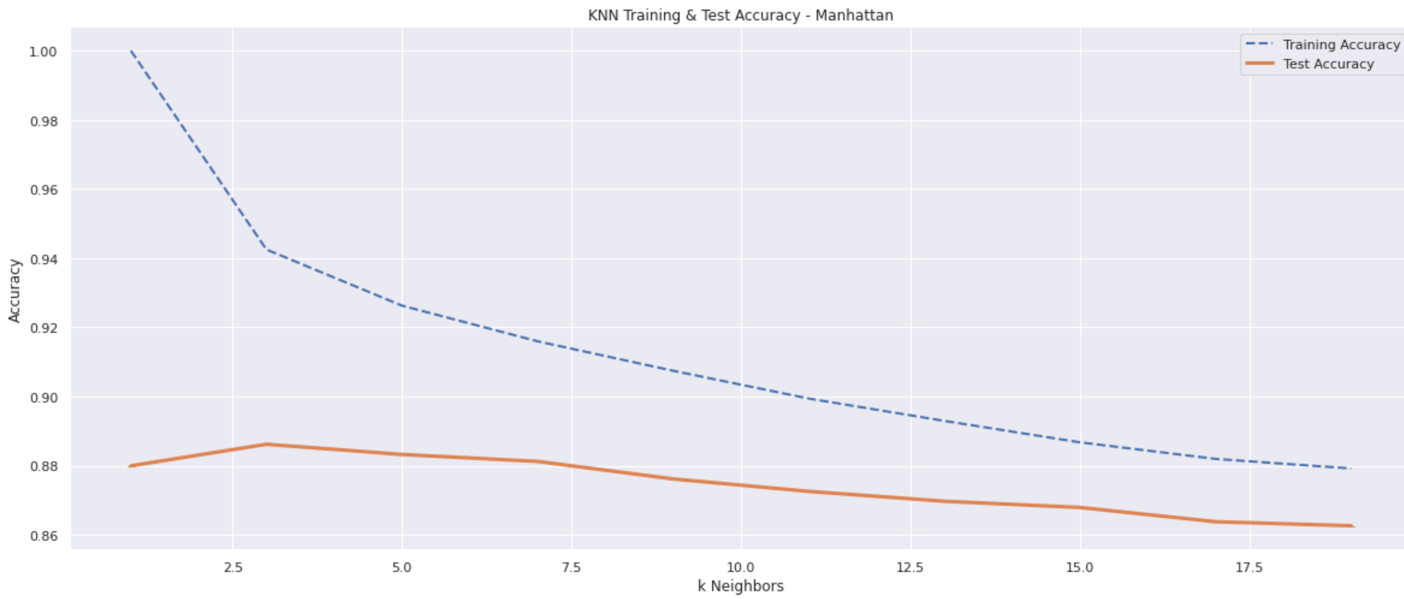
Out[ ]:

	k values	Training Accuracy	Test Accuracy
KNeighborsClassifier(metric='manhattan', n_neighbors=1)	1	1.000000	0.889805
KNeighborsClassifier(metric='manhattan', n_neighbors=3)	3	0.949375	0.900475
KNeighborsClassifier(metric='manhattan')	5	0.937025	0.906392
KNeighborsClassifier(metric='manhattan', n_neighbors=7)	7	0.930227	0.902905
KNeighborsClassifier(metric='manhattan', n_neighbors=9)	9	0.922255	0.896778
KNeighborsClassifier(metric='manhattan', n_neighbors=11)	11	0.911865	0.889699
KNeighborsClassifier(metric='manhattan', n_neighbors=13)	13	0.904281	0.881986
KNeighborsClassifier(metric='manhattan', n_neighbors=15)	15	0.896297	0.877443
KNeighborsClassifier(metric='manhattan', n_neighbors=17)	17	0.891378	0.876704
KNeighborsClassifier(metric='manhattan', n_neighbors=19)	19	0.887457	0.872266

Model Validation

In [ ]:

```
plt.plot(knn_df_man['k values'], knn_df['Training Accuracy'], '--',linewidth=2, label='Tra
plt.plot(knn_df_man['k values'], knn_df['Test Accuracy'], '-_',linewidth=3, label='Test Ac

plt.xlabel('k Neighbors')
plt.ylabel('Accuracy')
plt.title('KNN Training & Test Accuracy - Manhattan')
plt.legend()
plt.show()
```



Identifying the Best Performing Model

In [ ]:

```
#return the results of the most accurate KNN model - Manhattan
knn_accuracy = KNeighborsClassifier(metric='manhattan',p=2, n_neighbors=5).fit(X_train_cre
                                                                 y_train_credit)
knn_model = knn_accuracy.fit(X_train_credit_pre, y_train_credit)

y_pred_knn_man = knn_model.predict(X_test_credit_pre)
y_pred_knn_man_train = knn_model.predict(X_train_credit_pre)

print('accuracy %2.2f ' % accuracy_score(y_test_credit,y_pred_knn_man))

man_acc = accuracy_score(y_test_credit, y_pred_knn_man)
accuracy_score_table.loc[len(accuracy_score_table.index)] = ['KNN Manhattan', man_acc]
```

```
accuracy 0.91
```

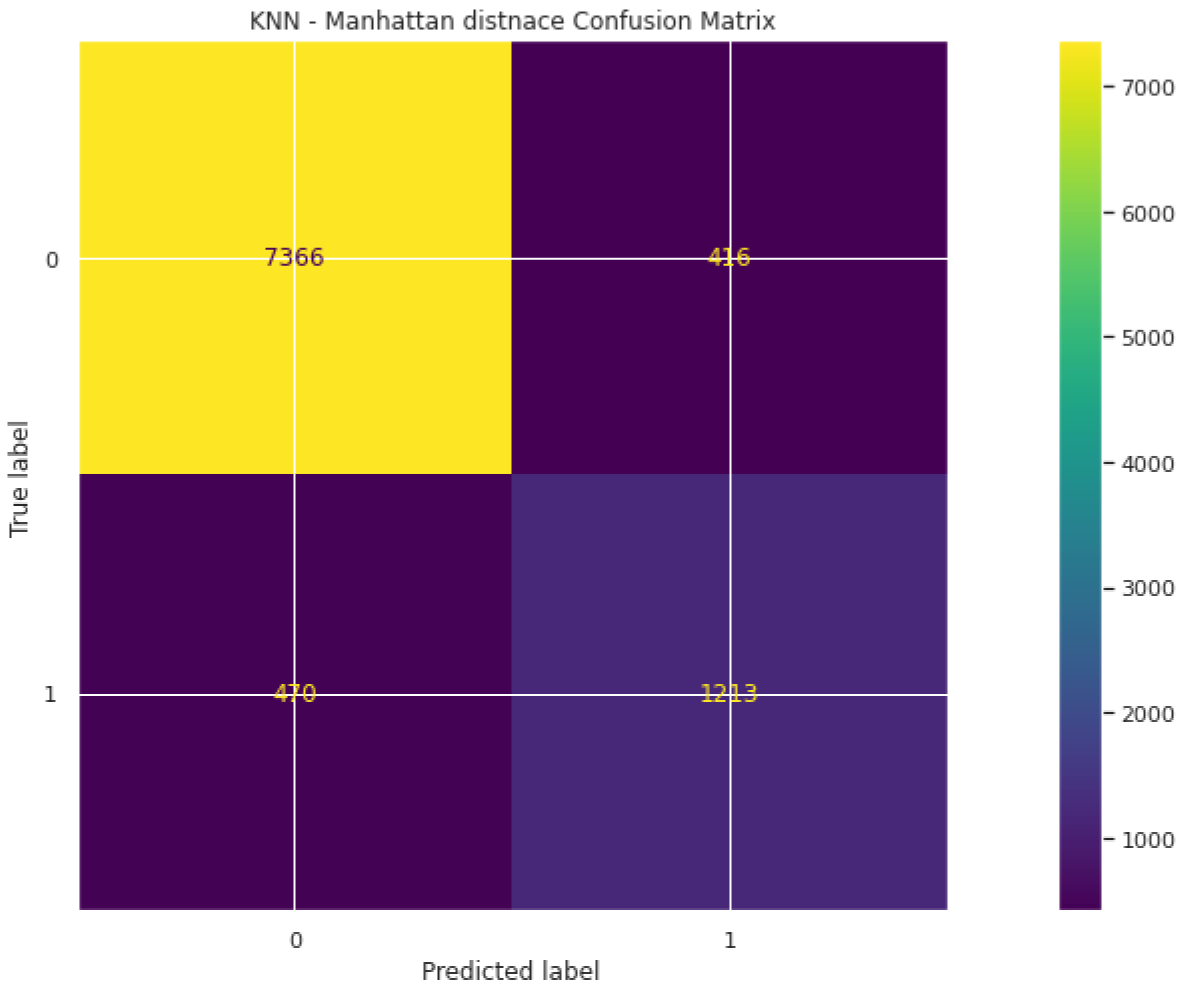
Confusion Matrix

In [ ]:

```
cm_mod_train_lr = confusion_matrix(y_train_credit, y_pred_knn_man_train)
cm_mod_test_knn_man = confusion_matrix(y_test_credit,y_pred_knn_man)
labels_mod = ['Yes','No']
train_results_lr = pd.DataFrame(cm_mod_train_lr, index = labels_mod,
                                  columns = labels_mod)
test_results_knn_man = pd.DataFrame(cm_mod_test_knn_man, index = labels_mod,
                                  columns = labels_mod)
```

In [ ]:

```
confusion_matrix_plot = metrics.confusion_matrix(y_test_credit,y_pred_knn_man)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix_plot)
cm_display.plot()
plt.title("KNN - Manhattan distnace Confusion Matrix")
plt.show()
```



Classification Report

In [ ]:

```
print('\nClassification Report for Credit Data on KNN - Manhattan Classifier:\n')
print(classification_report(y_test_credit, y_pred_knn_man))
print('\n')
```

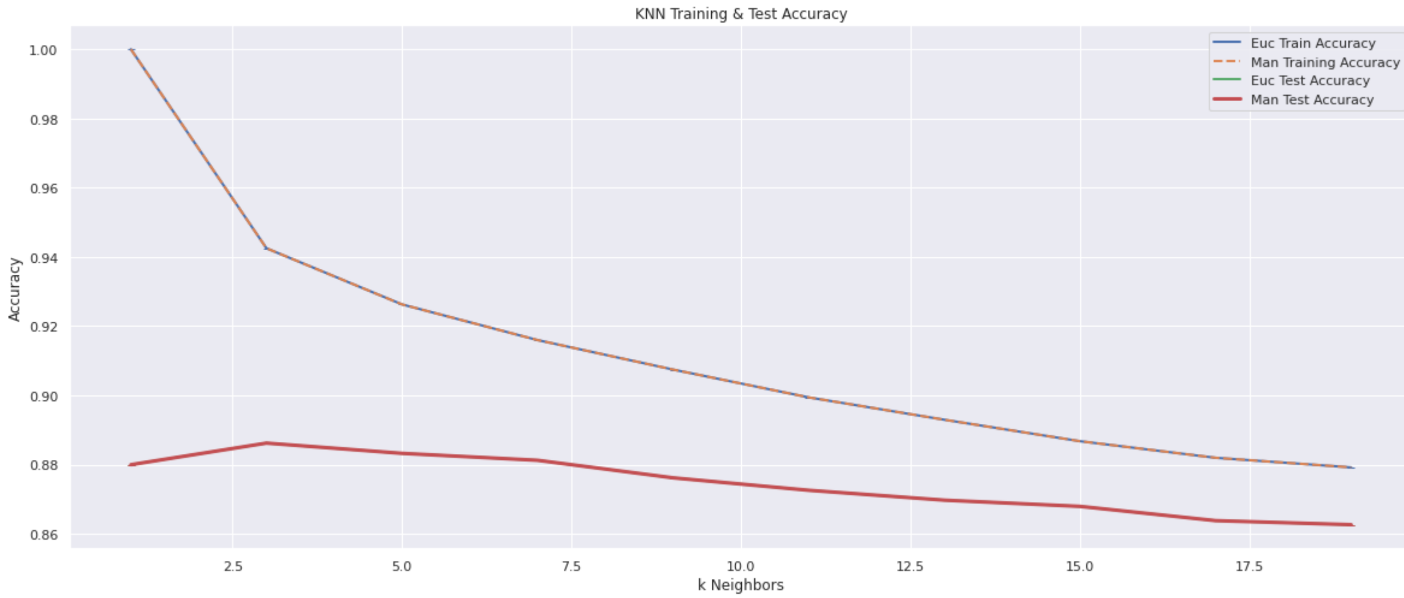
```
Classification Report for Credit Data on KNN - Manhattan Classifier:

              precision    recall  f1-score   support

          No       0.94      0.95      0.94      7782
         Yes       0.74      0.72      0.73      1683

    accuracy                           0.91      9465
   macro avg       0.84      0.83      0.84      9465
weighted avg       0.91      0.91      0.91      9465
```

Comparing Euclidean vs Manhattan KNN Performance

In [ ]:

```
# plot of KNN model Performance

plt.plot(knn_df['k values'], knn_df['Training Accuracy'], '-_',linewidth=2, label='Euc Tra
plt.plot(knn_df_man['k values'], knn_df['Training Accuracy'], '--',linewidth=2, label='Man
plt.plot(knn_df['k values'], knn_df['Test Accuracy'], '-_',linewidth=2, label='Euc Test Ac
plt.plot(knn_df_man['k values'], knn_df['Test Accuracy'], '-_',linewidth=3, label='Man Tes

plt.xlabel('k Neighbors')
plt.ylabel('Accuracy')
plt.title('KNN Training & Test Accuracy')
```

```
plt.legend()
plt.show()
```



SGDClassifier Model

Model Fitting

In [ ]:

```
#loop to create the model

loss_f = ['log', 'hinge', 'perceptron']
scores_list = []
avg_list = []

for i, loss in enumerate(loss_f):
  models = SGDClassifier(loss = loss).fit(X_train_credit_pre,y_train_credit)
  cv = cross_val_score(models, X_train_credit_pre, y_train_credit, cv=5, scoring='accuracy
  scores_list.append({'Model': models, 'cv': cv})
  avg_list.append({'Model': models, 'Avg CrossVal': cv.mean()})
```

In [ ]:

```
pd.DataFrame(avg_list)
```

Out[ ]:

	Model	Avg CrossVal
0	SGDClassifier(loss='log')	0.831044
1	SGDClassifier()	0.825679
2	SGDClassifier(loss='perceptron')	0.741548

Model Validation

Cross Validation

In [ ]:

```
from sklearn.linear_model import SGDClassifier

alphas = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
results = []
for a in alphas:
  l1clf = SGDClassifier(loss ='log', penalty= 'l1', alpha = a).fit(X_train_credit_pre,y_tr
```

```
    print('Finished training, alpha=%f' % a)
    l1cv = cross_val_score(l1clf, X_train_credit_pre, y_train_credit, scoring = 'accuracy',
    #Create cross_val_score with l1clf for Xtrain and  ytrain with cv=5 'accuracy' as scori
    print('Finished 5-fold CV, alpha=%f' % a)
    results.append({'alpha': a, 'log L1': l1cv.mean()})
    results # append 'alpha' and 'log L1'
l1_acc = pd.DataFrame(results)
```

```
Finished training, alpha=0.000010
Finished 5-fold CV, alpha=0.000010
Finished training, alpha=0.000100
Finished 5-fold CV, alpha=0.000100
Finished training, alpha=0.001000
Finished 5-fold CV, alpha=0.001000
Finished training, alpha=0.010000
Finished 5-fold CV, alpha=0.010000
Finished training, alpha=0.100000
Finished 5-fold CV, alpha=0.100000
Finished training, alpha=1.000000
Finished 5-fold CV, alpha=1.000000
Finished training, alpha=10.000000
Finished 5-fold CV, alpha=10.000000
Finished training, alpha=100.000000
Finished 5-fold CV, alpha=100.000000
Finished training, alpha=1000.000000
Finished 5-fold CV, alpha=1000.000000
```

In [ ]:

```
print(l1_acc)
```

```
        alpha    log L1
0     0.00001  0.829365
1     0.00010  0.831255
2     0.00100  0.837642
3     0.01000  0.832911
4     0.10000  0.822180
5     1.00000  0.822180
6    10.00000  0.822180
7   100.00000  0.822180
8  1000.00000  0.822180
```

In [ ]:

```
#use the best Stochastic Gradient Descent classifier to build specific model
sgcd = SGDClassifier(loss ='log', penalty= 'l1', alpha = 0.001).fit(X_train_credit_pre,y_t
sgcd_pred = sgcd.predict(X_test_credit_pre)
print('accuracy %2.2f ' % accuracy_score(y_test_credit,sgcd_pred))

sgd_acc = accuracy_score(y_test_credit, sgcd_pred)
accuracy_score_table.loc[len(accuracy_score_table.index)] = ['SGD', sgd_acc]
```

```
accuracy 0.84
```

Confusion Matrix

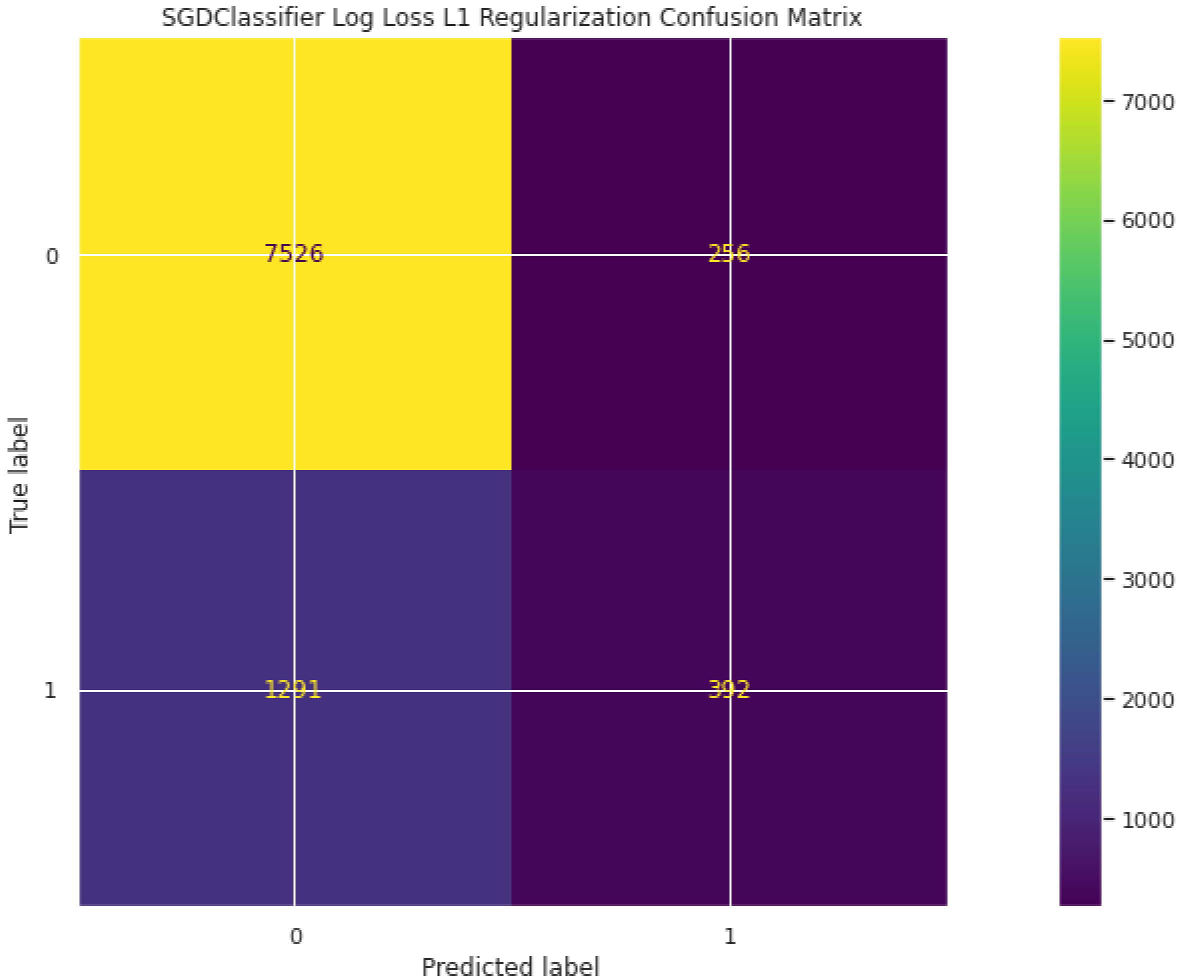
In [ ]:

```
cm_mod_test_sgcd = confusion_matrix(y_test_credit, sgcd_pred)

test_results_sgcd = pd.DataFrame(cm_mod_test_sgcd, index = labels_mod,
                                 columns = labels_mod)
```

In [ ]:

```
from sklearn import metrics
confusion_matrix_plot = metrics.confusion_matrix(y_test_credit,sgcd_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix_plot)
cm_display.plot()
```

```
plt.title("SGDClassifier Log Loss L1 Regularization Confusion Matrix")
plt.show()
```



Classification Report

In [ ]:

```
from sklearn.metrics import confusion_matrix, accuracy_score, plot_confusion_matrix, class
print('\nClassification Report for Credit Data on SGDClassifier - Log Loss - L1:\n')
print(classification_report(y_test_credit, sgcd_pred))
print('\n')
```

```
Classification Report for Credit Data on SGDClassifier - Log Loss - L1:

              precision    recall  f1-score   support

          No       0.85      0.97      0.91      7782
         Yes       0.60      0.23      0.34      1683

    accuracy                           0.84      9465
   macro avg       0.73      0.60      0.62      9465
weighted avg       0.81      0.84      0.81      9465
```

Performance Metrics and Modeling Results

Accuracy Scores

In [ ]:

```
accuracy_score_table = accuracy_score_table.sort_values('Accuracy Score', ascending=False)
```

```
print(accuracy_score_table)
```

```
                   Model  Accuracy Score
0          KNN Manhattan        0.906392
1         Random Forests        0.904385
2                KNN Euc        0.886212
3          Decision Tree        0.857792
4         Neural Network        0.837190
5                    SGD        0.836556
6  Unbalanced Perceptron        0.817961
7    Balanced Perceptron        0.673006
```

Classification Report

In [ ]:

```
print('\nClassification Report for Credit Data on Balanced Classifier:\n')
print(classification_report(y_test_credit, balanced_mod_pred))
print('\n')
```

```
Classification Report for Credit Data on Balanced Classifier:

              precision    recall  f1-score   support

          No       0.87      0.71      0.78      7782
         Yes       0.28      0.52      0.36      1683

    accuracy                           0.67      9465
   macro avg       0.57      0.61      0.57      9465
weighted avg       0.77      0.67      0.71      9465
```

In [ ]:

```
print('\nClassification Report for Credit Data on Unbalanced Classifier:\n')
print(classification_report(y_test_credit, unbalanced_mod_pred))
print('\n')
```

```
Classification Report for Credit Data on Unbalanced Classifier:

              precision    recall  f1-score   support

          No       0.82      0.99      0.90      7782
         Yes       0.34      0.02      0.05      1683

    accuracy                           0.82      9465
   macro avg       0.58      0.51      0.47      9465
weighted avg       0.74      0.82      0.75      9465
```

In [ ]:

```
print('\nClassification Report for Credit Data on RF Classifier:\n')
print(classification_report(y_test_credit, rf_test_pred))
print('\n')
```

```
Classification Report for Credit Data on RF Classifier:

              precision    recall  f1-score   support

          No       0.93      0.96      0.94      7782
         Yes       0.76      0.67      0.71      1683

    accuracy                           0.90      9465
```

```
   macro avg       0.85      0.81      0.83      9465
weighted avg       0.90      0.90      0.90      9465
```

In [ ]:

```
print('\nClassification Report for Credit Data on Decision Tree Classifier:\n')
print(classification_report(y_test_credit, y_pred_dt))
print('\n')
```

```
Classification Report for Credit Data on Decision Tree Classifier:

              precision    recall  f1-score   support

          No       0.91      0.91      0.91      7782
         Yes       0.60      0.61      0.60      1683

    accuracy                           0.86      9465
   macro avg       0.76      0.76      0.76      9465
weighted avg       0.86      0.86      0.86      9465
```

In [ ]:

```
print('\nClassification Report for Credit Data on KNN - Euclidean Classifier:\n')
print(classification_report(y_test_credit, y_pred_knn_euc))
print('\n')
```

```
Classification Report for Credit Data on KNN - Euclidean Classifier:

              precision    recall  f1-score   support

          No       0.93      0.94      0.93      7782
         Yes       0.69      0.65      0.67      1683

    accuracy                           0.89      9465
   macro avg       0.81      0.79      0.80      9465
weighted avg       0.88      0.89      0.88      9465
```

In [ ]:

```
print('\nClassification Report for Credit Data on KNN - Manhattan Classifier:\n')
print(classification_report(y_test_credit, y_pred_knn_man))
print('\n')
```

```
Classification Report for Credit Data on KNN - Manhattan Classifier:

              precision    recall  f1-score   support

          No       0.94      0.95      0.94      7782
         Yes       0.74      0.72      0.73      1683

    accuracy                           0.91      9465
   macro avg       0.84      0.83      0.84      9465
weighted avg       0.91      0.91      0.91      9465
```

In [ ]:

```
print('\nClassification Report for Credit Data on SGDClassifier - Log Loss - L1:\n')
print(classification_report(y_test_credit, sgcd_pred))
print('\n')
```

```
Classification Report for Credit Data on SGDClassifier - Log Loss - L1:

              precision    recall  f1-score   support

          No       0.85      0.97      0.91      7782
         Yes       0.60      0.23      0.34      1683

    accuracy                           0.84      9465
   macro avg       0.73      0.60      0.62      9465
weighted avg       0.81      0.84      0.81      9465
```

In [ ]:

```
print('\nClassification Report for Credit Data on NN Classifier:\n')
print(classification_report(y_test_credit, nn_test_pred))
print('\n')
```

```
Classification Report for Credit Data on NN Classifier:

              precision    recall  f1-score   support

          No       0.87      0.95      0.91      7782
         Yes       0.57      0.34      0.42      1683

    accuracy                           0.84      9465
   macro avg       0.72      0.64      0.66      9465
weighted avg       0.82      0.84      0.82      9465
```