1 Cover Letter

In this project, we explore data on home loan applicants in an attempt to create a model to approve or deny applicants. We apply two different classification techniques, logistic regression and K-nearest neighbors, and observe the accuracy and run-time of both methods. We implement both methods from scratch and use gradient descent, as discussed in lectures, to minimize the cost function of the logistic regression model by iterating over the cost function until it converges to a local minima. We used these methods on both a real dataset and a simulated dataset to see how using more data would affect the accuracy and the run-time.

After applying our two models to the real data our main findings were:

- Simulation Study: K-Nearest Neighbors takes 1.691 seconds with an accuracy of 67.3%.
- Simulation Study: Logistic Regression takes 0.6101 seconds and achieves an accuracy of 70%.
- Real Dataset: K-Nearest Neighbors takes 0.09 seconds to run and achieves an accuracy of 73% on average.
- Real Dataset: Logistic Regression takes 0.5273 seconds and was able to predict 71% of the false value accurately and 88% of the true value accurately with an overall accuracy of 85%.
- Logistic Regression had better accuracy than K-Nearest Neighbors in both cases.

Despite the logistic regression model having a slightly higher run-time on the real dataset, the difference in accuracy makes the logistic regression model a better choice.

Home Loan Approval

Howie Huang, Trevor Adam, Laura Valenzuela, Marco Oviedo

March 21, 2023

Abstract

Millions of homes are sold each year in the U.S., and the majority of first-time home buyers need a home loan to make their purchase. This could lead to loan officers viewing thousands of applications each year and applicants experiencing extended wait times for loan approval. The goal of this project was to create and compare two classification models to see if an applicant would be approved for a home loan. The two classification models used are K-nearest Neighbors and a Logistic Regression model implemented with the Gradient Descent Algorithm. The average accuracy of the K-nearest Neighbor Model is 73%, with a significant majority of misclassifications being false positives. The accuracy of the Logistic Regression Model is 85%, with a slight majority of misclassifications being false positives. Based on the accuracies of the models, the Logistic Regression model would be the superior model for applicants to receive pre-approvals for home loans .

Contents

2 Introduction

Home loans, also known as mortgages, are a type of loan that enables individuals to purchase or refinance a home. Obtaining a home loan can be a crucial step in achieving one's dream of home ownership. However, the loan approval process can be challenging, and many factors are taken into consideration to determine whether a person is eligible for a home loan. These factors can range from the borrower's credit score to their income and financial history. To aid in the loan approval process, statistical models such as logistic regression can be very useful.

In this project report, we will explore the use of K-nearest neighbor algorithm and logistic regression on a home loan approval data set to create a predictive classifier with given factors to assist lenders in making informed decisions in terms of whether someone is eligible to obtain a home loan. The benefit of this project is for the lenders to know which group of people are likely to return the money they borrowed. In addition, it will automate the process and make it faster for lenders to choose who would get a loan and who wouldn't.

3 Dataset and Data Analysis

The data source that we will be using comes from Kaggle. It consists of 12 independent variables and 1 dependent variable. The training set contains 614 observations, and the testing set contains 367 observations. The variables of the data set are:

Attributes	Description	Data Types
Loan ID	The identification of a loan application	Numeric
Gender	The sex of the applicant	Categorical
Married	The marital status of an applicant with 0 being unmarried and 1 being married	Categorical
Dependents	The number of dependents of an applicant	Categorical
Education	The graduation status of an applicant	Categorical
Self Employed	The self-employment status of an applicant	Categorical
Applicant Income	The monthly income of an applicant	Numeric
Co-Applicant Income	The monthly income of the co-applicant	Numeric
Loan Amount	The total loan amount an applicant requested in thousands	Numeric
Loan Term	The total loan amount an applicant requested in thousands	Numeric
Property Area	The location of the home	Categorical
Credit History	The credit history status of an applicant	Categorical
Loan Status	The loan status of the application	Categorical

The dependent variable is:

- Loan Status (binary): The loan status of the application with 0 being not approved and 1 being approved

In order to make the data set more understandable and easier to process, we converted all the categorical data into binary values. The attributes with multiple categorical values were converted to binary using One-Hot Encoding. This would change Property Area and Dependents from a single column to multiple columns containing binary values. The Dependents attribute was split into four columns one for each categorical value it contained. The names of the new four columns are Dependent_0, Dependent_1, Dependent_2, and Dependent_3+. The Property area attribute was split into 3 columns. The new columns were named Property_Area_Urban, Property_Area_Semiurban, and Property_Area_Rural. An example of how this labeling works is if one applicant is marked down as living in an Urban area. A one would be marked under the Property_Area_Urban column, while a zero is given under the Property_Area_Semiurban and Property_Area_Rural columns. The newly added attributes will be in a table below.

Attributes	Description	Data Types
Dependents_0	The applicant has no dependents	Categorical
Dependents_1	The applicant has one dependent	Categorical
Dependents_2	The applicant has two dependents	Categorical
Dependents_3+	The applicant has three or more dependents	Categorical
Property_Area_Urban	The applicant lives in an urban area	Categorical
Property_Area_Semiurban	The applicant lives in a semi-urban Area	Categorical
Property_Area_Rural	The applicant lives in a rural area	Categorical

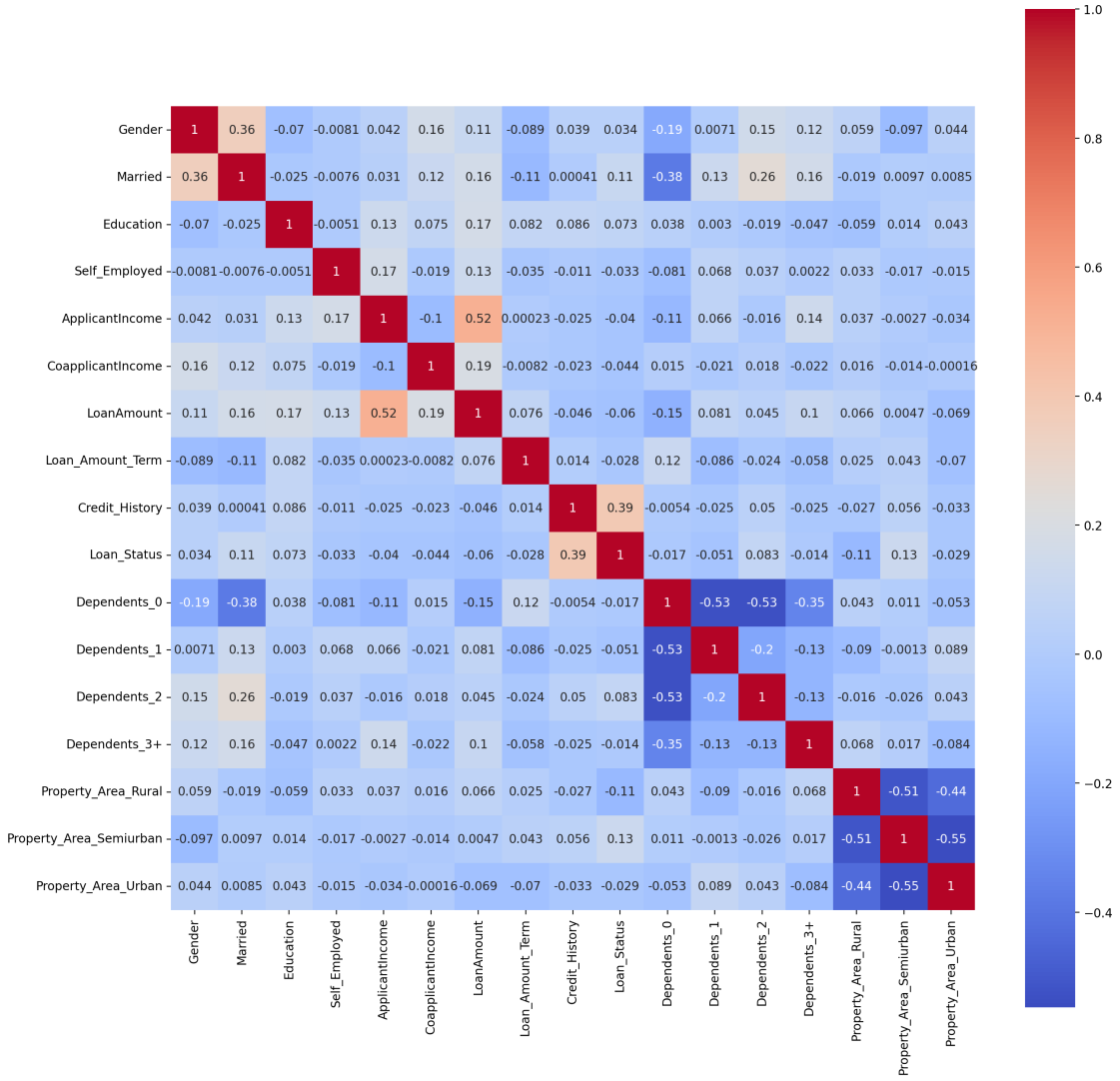

Figure 1: Correlation Heat Map of the Data

Figure 1 is a Heat map of the correlation values in our data set. Looking at the figure, we can see that the majority of variables have almost no correlation between them. Towards the bottom right of the Heat Map, we can see some variables have a slightly moderate negative correlation between them.

When looking into our data, we noticed the Applicant income had the most outliers and variability in its values. This was to be expected since income can vary drastically among individuals.

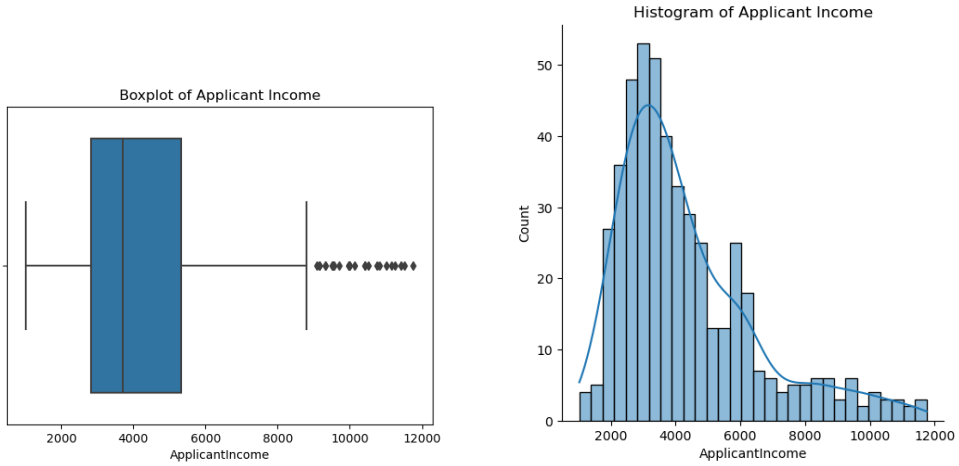

Figure 2: Distribution of an Applicants Income

Figure 2 shows these outliers in the box plot and the histogram shows the distribution of the Applicant income data. Looking at the box plot we can see that the outliers are made up of applicants making more than nine thousand a month. In addition, the histogram shows that the distribution of the data is right-skewed. Instead of removing the outliers in our data set we instead decided to perform a log transformation of the data to see if that would resolve the issues.

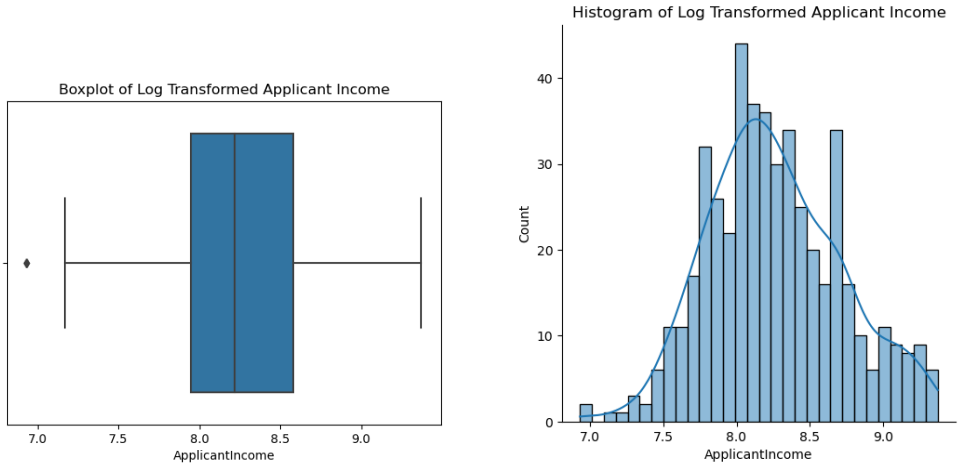

Figure 3: Distribution of an Applicants Log Transformed Income

From the Figure 3 box plot we can see that practically all of the outliers have been accounted for once we log-transformed the data. Also, we can see the distribution of the data has a normal bell-shaped curve. Log Transforming the data is an important step in this report because it allows us to keep the outliers present in the data set. This benefits our loan approval model because it can now account for applicants with a higher monthly income.

4 Proposed Methods

4.1 K-Nearest Neighbors

K-Nearest Neighbors is a classification algorithm that classes a new observation based on the class of the K nearest points from the training data. We approach this method under the assumption that points that are close to each other share similarities and are more likely to be from the same class. To implement our algorithm, we first normalize our data to avoid some features dominating the distance calculation. Then, for each observation in the test data, we calculate the euclidean distance between the test observation and every observation in the training data. We use the distance formula

$$d(x_{test}, x_{train}) = \sqrt{\sum_{i=1}^{p} (x_{test_i} - x_{train_i})}$$

where x_{test} and $x_{train} \in \mathbb{R}^p$ are observations and p is the amount of features. After calculating the distance between the test observation and all training observations, we find the class of the K nearest neighbors and classify our test observation based on a majority vote.

4.2 Logistic Regression

Logistic Regression is a statistical technique used to predict the relationship between our independent variables and our dependent variable. In this case, our dependent variable is either 0 or 1. Which makes this classification method be called Binary Logistic Regression. Before applying this model, we need to test the logistic regression assumptions, and see if they hold or not. These assumptions are the absence of multicollinearity, which means there is no high inter correlation among the predictors. No outliers, which have been removed in the data cleaning process. And the sample size is adequate with not a lot of variables compared to the observations. After doing some tests with the data and doing the partial dependence plots we can say that the assumptions are true.

For this project, we used the sigmoid function. It works by squishing any value to fit between 0 and 1:

$$S(x) = \frac{1}{1 + e^{-x}}$$

Its output is interpreted as the probability of a person getting a loan. And at the same time we need to use a cost function to see how good our model is at predicting who will get a loan, and how big or small the error is. The goal is to minimize this said error and we do that by using gradient descent. Gradient descent is going to iterate the function until it converges to the local minima and then it will stop.

5 Simulation Study

Our two classification models K-Nearest Neighbors and Logistic Regression with Gradient Descent, were tested on a simulation data set to assess their accuracy and computation speed. The simulation data set is modeled after the original home loan approval data set by taking into account the proportion of binary values in each of the attributes to try to get as possible to the original data set's distributions. The simulated data set has 17 attributes as well as 5000 simulated observations.

5.1 K-Nearest Neighbors

For K-Nearest neighbors, we ran through the simulated data set over two different iterations. The first iteration was running through different values of k and seeing the computational performance and accuracy of the model. After running the model through k values of 1-15, we got the following plot

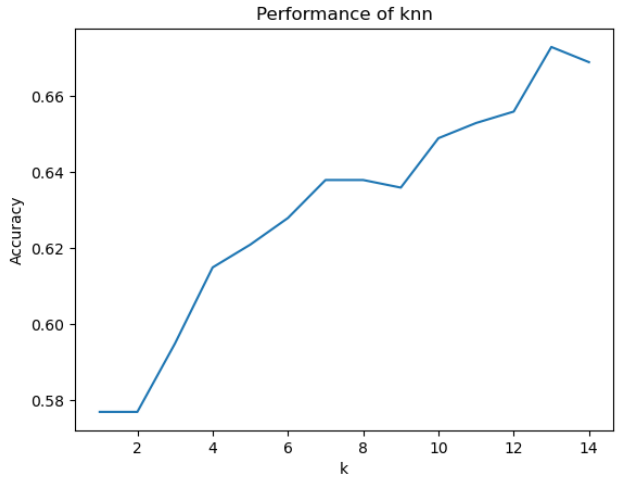

Figure 4: Accuracy of the model with Different Values of K

Figure 4 shows that the accuracy of the K-Nearest Neighbors model is not that high and that the accuracy of the model peaks when $k = 13$. The computational time to run through all 15 values of k was 23.123 seconds.

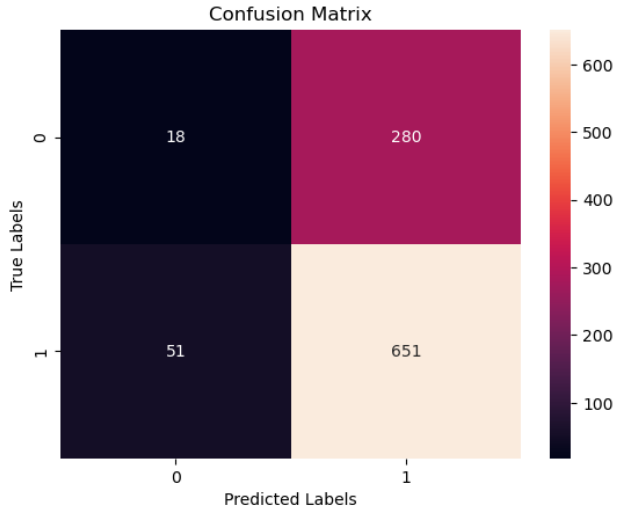

Figure 5: Confusion Matrix when k = 13

Figure 5 is the prediction from the simulated data when $k = 13$. We can see that the majority of errors from the K-Nearest Neighbors model are false positives. The accuracy of the model when $k = 13$ is 67.3% and the computational time it took to run with the single value of k is 1.691 seconds. When comparing it to the original data there is a decrease in the accuracy of the model. This could be due to significantly increasing the size of the data set as well as slight differences in the distribution of the simulated data.

5.2 Logistic Regression

In our logistic regression model, the first thing we did was to test the model assumptions. There are several model assumptions we need to ensure:

- Linearity: the relationship between the independent variables and the log-odds of the dependent variables is linear
- Independence: observations in the data are independent from each other and there is no multi-collinearity among the independent variable
- Sample Size: the sample size cannot be too small
- Outliers: there is no potential outliers that will affect the outcome of our model

To test the linearity in the data, we used partial dependence plots to show the linear relationship between each independent variables and the log-odds of the dependent variable. Note: from the x-axis we can tell that the data have been transformed, but this will not affect the linearity test since the relationships are consistent.

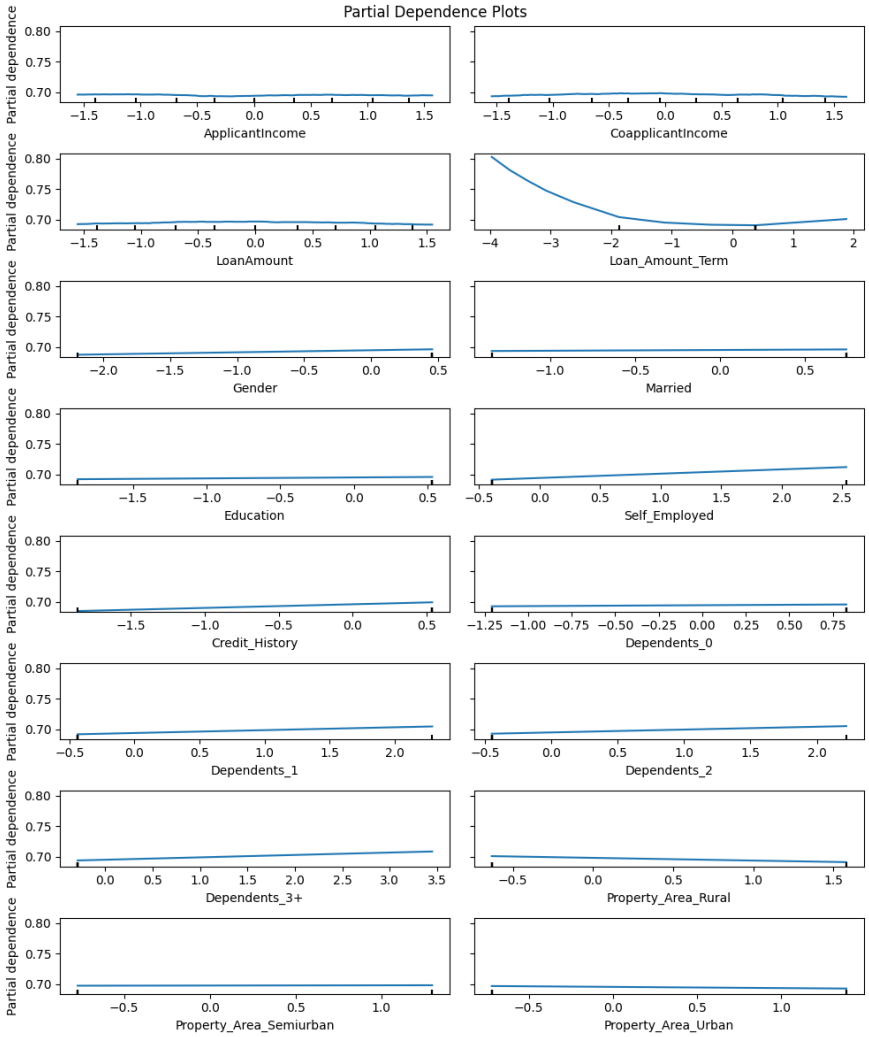

Figure 6: Partial Dependence Plots in Logistic Regression

Figure 6 proves that there exists a linear relationship between most of the independent variables and the log-odds of the dependent variable. Although the Loan_Amount_Terms has a semi-quadratic regression line, the relationship is still approximately linear.

To test whether the data are independent from each other and its collinearity, we can use a correlation matrix.

	ApplicantIncome	CoapplicantIncome	LoanAmount
ApplicantIncome	1.000000	-0.007017	-0.009517
CoapplicantIncome	-0.007017	1.000000	-0.022831
LoanAmount	-0.009517	-0.022831	1.000000
Loan_Amount_Term	-0.004339	-0.004278	0.004953
Gender	-0.011935	0.028321	-0.020562
Married	-0.006588	-0.016816	0.025820
Education	-0.003022	0.007452	0.006484
Self_Employed	-0.014685	-0.000452	-0.017852
Credit_History	0.003983	-0.001378	-0.015770
Dependents_0	0.011769	0.014491	0.011112
Dependents_1	-0.012112	-0.001184	0.000459
Dependents_2	-0.007592	-0.002499	-0.003183
Dependents_3+	0.006129	-0.021665	-0.016797
Property_Area_Rural	-0.020682	-0.014184	0.001830
Property_Area_Semiurban	0.015008	-0.007923	0.000475
Property_Area_Urban	0.004507	0.021903	-0.002261

Figure 7: Correlation Matrix in Logistic Regression

Figure 7 is a part of the correlation matrix, the rest of the matrix is consistent. As the correlation values in the non-diagonal entries approach to 1 or -1, it is an indication that the data are highly correlated; as they approach to 0, the data are independent. As we can see, all the correlation value are small which indicates the data are independent and there is no collinearity among the independent variables.

In terms of sample size and outliers, since the simulated data has 5000 observations and we made sure to account for numeric outliers when transforming the data, two assumptions are satisfied.

Our logistic regression has a learning rate of 0.1. We constructed a forward-propagation method so that when an input data is given, the result of the predicted probability from the sigmoid function will be output. To make prediction, we classify any probability bigger than 0.6 to 1 and any result less or equal to 0.6 to 0. Additionally, we will split 50-50 between the training data and the testing data in this simulation data. After fitting the model, we achieve 70% accuracy with a run time of 0.6101 seconds. The following figure is a confusion matrix of our results:

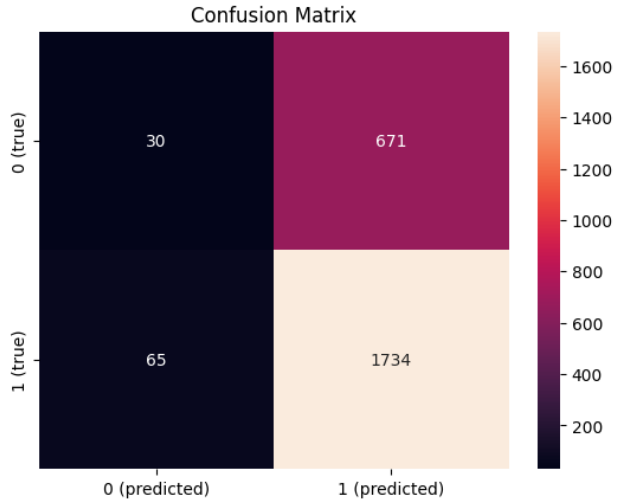

Figure 8: Confusion Matrix in Logistic Regression

From figure 8 we can see that the majority of predicted value is true positive. 671 predicted values are false positive, 65 predicted values are false negative, and 30 predicted values are true negative. Although

we achieve a decent accuracy with the model, it seems like we can make improvement on better detecting negative(0) values.

6 Real Data

6.1 K-Nearest Neighbors

To choose a value for K, we created multiple training and test splits and iterated over multiple values of K, and calculated the accuracy using the algorithm discussed earlier. After running one hundred train and test splits on different values of K, we found that $K = 13$ provided the most consistent results. We also ran 5-fold cross-validation for different values of K and reached the same conclusion. The K-Nearest Neighbors on the real dataset takes 0.09 seconds to run and achieves an accuracy of 73% on average. When comparing our KNN function to the one in the scikit-learn package in python, we achieve similar accuracy but the sk-learn package was faster with a 0.005 second runtime to perform the same task. Our KNN algorithm performs slightly better on the real data compared to the simulated data but by

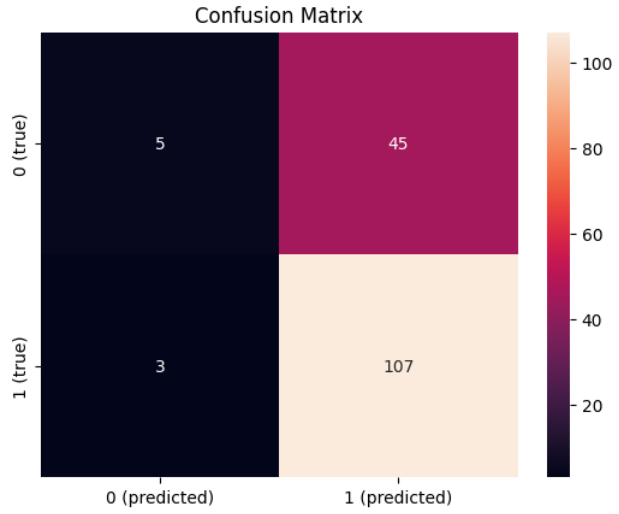

Figure 9: Real data confusion matrix when k = 13

viewing the confusion matrix we can see that there are some issues with our predictions. We have 5 true negatives and 107 true positives but, the majority of misclassifications are false positives which means our model approves someone for a loan when they are not qualified which is not ideal. It is possible that the imbalance in predictions is caused by the data being imbalanced.

6.2 Logistic Regression

First we made sure all model assumptions are satisfied. Since the real data is smaller than the simulation data, we had to cut down the binary classification threshold from 0.6 to 0.5 in order to achieve a better accuracy. This time, we assign 70% of the data as training data and 30% of the data as testing data. When the threshold was 0.6, we were able to predict 50% of the false value(0) accurately and 88% of the true value(1) accurately, with a overall accuracy of 77%. Now that the threshold is 0.5, we are able to predict 71% of the false value accurately and 88% of the true value accurately with a overall accuracy of 85%, which is a higher accuracy than the logistic regression algorithm in sk-learn package(79%). The run-time is approximately 0.5273 seconds and the confusion matrix is the following:

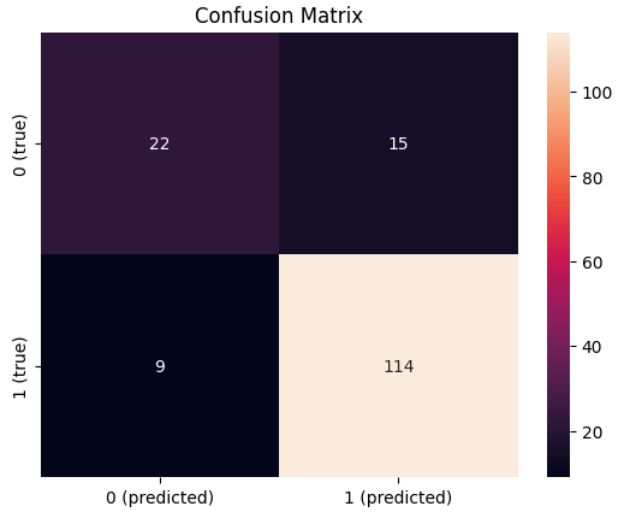

Figure 10: Real Data Confusion Matrix in Logistic Regression

Comparing both confusion matrix using simulation data and real data, we can see that it performs a lot better in the real data.

7 Results

Between false positive and false negative values, both dataset in both models suggests that the models tend to predict false positive values. This may likely due to outside variables that are not included in the dataset affecting the true results, which further suggests an improvement in future data acquisition. This may also be the result of an imbalance in the data set. The logistic regression model yields much greater accuracy than the KNN model and the classification error are more balanced meaning that it would be the better model for home loan predictions.

8 Code

We optimized our code by making the models by scratch and by implementing the gradient descent algorithm.

```
'''Importing the needed packages'''
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import time

'''Cleaning the Data'''
df_train
```

```
df_train.info()
df_train.isnull().sum()

#fills the missing values with 0
df_train['Credit_History'] = df_train['Credit_History'].fillna(0)
df_train['Dependents'] = df_train['Dependents'].fillna(0)

df_train.isnull().sum()
df_train = df_train.dropna()

df_train = df_train[df_train['ApplicantIncome'] < 14000]  #removes extreme values
df_train = df_train[df_train['CoapplicantIncome'] < 7000]  #removes extreme values
df_train
df_train = df_train.drop(['Loan_ID'], axis=1)        #drops the loan column of the data frame
df_train.Dependents.unique()

#categorical variable to binary values
df_train = df_train.replace({'Loan_Status': {'Y': 1, 'N': 0}})
df_train = df_train.replace({'Gender': {'Male': 1, 'Female': 0}})
df_train = df_train.replace({'Married': {'Yes': 1, 'No': 0}})
df_train = df_train.replace({'Education': {'Graduate': 1, 'Not Graduate': 0}})
df_train = df_train.replace({'Self_Employed': {'Yes': 1, 'No': 0}})
df_train = df_train.replace({'Dependents': { 0: '0'}})
df_train.info()

df_train = pd.get_dummies(df_train)    #changes all categorical variable to binary values

df_train.to_csv(r'C:\Users\ovied\Downloads\df_train.csv')

'''Data Analysis'''
df = pd.read_csv(r"C:\Users\ovied\Downloads\df_train.csv")
df = df.drop('Unnamed: 0', axis=1)

corr = df.corr()
plt.subplots(figsize=(15,15),dpi=200)
sns.heatmap(corr, cmap='coolwarm', annot=True, square=True)

sns.boxplot(x='ApplicantIncome',data=df)
plt.title('Boxplot of Applicant Income')

sns.displot(df['ApplicantIncome'], kde=True, bins=30)
plt.title('Histogram of Applicant Income')
plt.show()

df['ApplicantIncome'] = np.log(df['ApplicantIncome'])
sns.boxplot(x='ApplicantIncome',data=df)
plt.title('Boxplot of Log Transformed Applicant Income')
plt.show()

sns.displot(df['ApplicantIncome'], kde=True, bins=30)
plt.title('Histogram of Log Transformed Applicant Income')
plt.show()

'''KNN Model Real Data'''
df = pd.read_csv(r"C:\Users\ovied\Downloads\df_train.csv")
df = df.drop('Unnamed: 0', axis=1)
```

```
'''This function gets the euclidean distance'''
def euclidean_distance(point1, point2):
    return np.sqrt(np.sum(np.square(point1 - point2),axis=1))

'''This is our KNN model. It takes the testing and training data sets as well as the
parameter you wish to set k to'''

def KNN(X_test, X_train, y_train, k):
    neighbors = []
    for x in X_test:
        distances = euclidean_distance(x, X_train)          #gets the distances between the points
        dist = pd.DataFrame({'distance': distances, 'label': y_train})
        dist = dist.sort_values('distance', axis=0)         #sorts the distances and labels
        y_sorted = list(dist['label'][:k])
        neighbors.append(y_sorted[:k])                      #appends them to a list
        y_pred = []
        for neighbor in neighbors:
            counts = Counter(neighbor)                      #counts the labels
            label = counts.most_common(1)[0][0]             #takes the majority count of the label
            y_pred.append(label)                            #appends the majority count to the list
    return y_pred

'''This function is used the to the accuracy of our knn model'''

def KNN_accuracy(X_test, y_test, X_train, y_train, k):
    y_pred = KNN(X_test, X_train, y_train, k)           #gets the pred. values from the knn function
    accuracy = sum(y_pred == y_test) / len(y_test)      #gets the accuracy
    return accuracy, y_pred


X = df.drop(['Loan_Status'], axis = 1)
y = df['Loan_Status']
scaler = StandardScaler()
X = scaler.fit_transform(X)    #Standardize the data in X
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

start = time.time()
accuracies = []
ks = range(1, 15)
for k in ks:
    accuracy, y_pred = KNN_accuracy(X_test, y_test, X_train, y_train, k=k)
    accuracies.append(accuracy)
end = time.time()
elapsed_time = end - start
print(f'Elapsed time {elapsed_time}')


fig, ax = plt.subplots()
ax.plot(ks, accuracies)
ax.set(xlabel="k",
       ylabel="Accuracy",
       title="Performance of knn")
plt.show()
```

```
cm = confusion_matrix(y_test, y_pred)     #gets the confusion matrix
sns.heatmap(cm, annot=True,  fmt='d')     #makes a plot of the matrix

plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

'''Getting the average accuracy from the model'''
import statistics
ks = range(1, 15)
avg_scores = []
for k in ks: #iterate over ks
    for i in range(1,100): #100 iterations for each K
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
        accuracies = []
        accuracy, y_pred = KNN_accuracy(X_test, y_test, X_train, y_train, k=k)
        accuracies.append(accuracy)
    avg_scores.append(accuracy) #average accuracies over 100 models for each k

statistics.mean(avg_scores)

"""Testing the Prebuilt model from sklearn"""

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
X = df.drop('Loan_Status', axis = 1)
y = df['Loan_Status']
scores = []
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state = 80) #Create data split
knn = KNeighborsClassifier(n_neighbors= 20 , algorithm='kd_tree')
#KD Tree algorithm, time complexity of O(nlog(d)), K = sqrt(N)

knn.fit(X_train.values,y_train)#fit knn

scores.append(knn.score(X_test, y_test))#Get accuracy

scaler = StandardScaler()  #Scale data
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#KD Tree algorithm, time complexity of O(nlog(d)), K = sqrt(N)
knn2 = KNeighborsClassifier(n_neighbors= 13 , algorithm='kd_tree')
#this time the input data is scaled
knn2.fit(X_train,y_train)#fit knn
knn2.score(X_test,y_test)

knn_pred_scaled = knn2.predict(X_test)

from sklearn.model_selection import cross_val_score
knn_cv = KNeighborsClassifier(n_neighbors= 13 , algorithm='kd_tree')
X = df.drop('Loan_Status', axis = 1)
y = df['Loan_Status']
```

```
cv_scores = cross_val_score(knn_cv,X,y,cv=7)

cv_scores

'''KNN Model Sim Data'''

df2 = pd.read_csv(r"C:\Users\ovied\Downloads\Simulation_data.csv")
df2 = df2.drop('Unnamed: 0', axis=1)
X2 = df2.drop(['Loan_Status'], axis = 1)
y2 = df2['Loan_Status']
scaler = StandardScaler()
X2 = scaler.fit_transform(X2)    #Standardize the data in X
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2)

start = time.time()
accuracies = []
ks = range(1, 15)
for k in ks:
    accuracy, y_pred = KNN_accuracy(X_test, y_test, X_train, y_train, k=k)
    accuracies.append(accuracy)
end = time.time()
elapsed_time = end - start
print(f'Elapsed time {elapsed_time}')


fig, ax = plt.subplots()
ax.plot(ks, accuracies)
ax.set(xlabel="k",
       ylabel="Accuracy",
       title="Performance of knn")
plt.show()

cm = confusion_matrix(y_test, y_pred)                    #makes the confusion matrix
sns.heatmap(cm, annot=True,  fmt='d')

plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()


'''Logistic Regression Model Real Data'''

df_test = pd.read_csv(r"/Users/howie/Desktop/STA 141C/Final Project/df_test.csv",index_col=0)
df_train = pd.read_csv(r"/Users/howie/Desktop/STA 141C/Final Project/df_train.csv",index_col=0)

X = df_train.drop(columns = "Loan_Status") #drop the column 'Loan_Status'
Y = df_train["Loan_Status"]

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state= 0)

# normalize the independent variables
X_train = (X_train - np.mean(X_train, axis=0)) / np.std(X_train, axis = 0)
X_test = (X_test - np.mean(X_test, axis=0)) / np.std(X_test, axis= 0 )

from sklearn.base import BaseEstimator, ClassifierMixin
```

```
class LogisticRegression(BaseEstimator, ClassifierMixin):

        def __init__(self, learning_rate = 0.1, num_iterations = 1000):
                self.learning_rate = learning_rate
                self.num_iterations = num_iterations


        def sigmoid(self, z):
            return 1 / (1 + np.exp(-z))

        def initialize_weights(self, n_features):
              self.weights = np.zeros((n_features,))
              self.bias = 0

        def forward_propagation(self,X):
              linear_output = np.dot(X, self.weights) + self.bias
              y_pred = self.sigmoid(linear_output)
              return y_pred

        def backward_propagation(self, X, y_pred, y_true):
              n_samples = X.shape[0]
              dw = (1 / n_samples) * np.dot(X.T, (y_pred - y_true))
              db = (1 / n_samples) * np.sum(y_pred - y_true)
              return dw, db

        def update_weights(self, dw, db):
              self.weights = self.weights - self.learning_rate * dw
              self.bias = self.bias - self.learning_rate * db

        def fit(self,X,y):
            n_samples, n_features = X.shape
            self.initialize_weights(n_features)

            for i  in range(self.num_iterations):
                y_pred = self.forward_propagation(X)
                dw, db = self.backward_propagation(X, y_pred, y)
                self.update_weights(dw, db)

            return self

        def predict(self,X):
              y_pred = self.forward_propagation(X)
              y_pred_class = np.where(y_pred > 0.5, 1, 0)
              return y_pred_class

        def predict1(self,X):
              y_pred= self.forward_propagation(X)
              return y_pred

        def predict_proba(self, X):
              y_pred = self.forward_propagation(X)
              return np.column_stack((1 - y_pred, y_pred))

log_reg = LogisticRegression(learning_rate=0.1, num_iterations=1000)
log_reg.fit(X = X_train, y = y_train)
```

```
log_fitted = log_reg.fit(X = X_train, y = y_train)

from sklearn import metrics
from sklearn.metrics import classification_report

y_pred = log_reg.predict(X = X_test)
y_pred_prob = log_reg.predict1(X = X_test)
log_odds = np.log(y_pred_prob / (1 - y_pred_prob))
print("Accuracy:" ,metrics.accuracy_score(y_pred, y_test))

print(classification_report(y_test, y_pred))

#Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
ax = plt.subplot()
sns.heatmap(cm, annot = True, fmt= 'g', ax=ax)
ax.set_title("Confusion Matrix")
ax.xaxis.set_ticklabels(['0 (predicted)', '1 (predicted)'])
ax.yaxis.set_ticklabels(['0 (true)', '1 (true)'])
plt.show()

# From the confusion matrix, we can see there are 114 true postive, 22 true negative, 9 false positive, 15 fal

'''Logistic Regression Assumptions Check'''

import warnings
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.compose import TransformedTargetRegressor
from sklearn.inspection import PartialDependenceDisplay

# turn off warnings
warnings.filterwarnings('ignore') #ignore the warnings


# ensure the model output is an estimator
estimator = make_pipeline(
    StandardScaler(),
    PolynomialFeatures(degree=2),
    TransformedTargetRegressor(regressor=LogisticRegression(), transformer=StandardScaler())
)

estimator.fit(X = X_train, y = y_train)

# dis-normalize the training data (does not affect anything, but the numbers will look nice)
# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state= 0)

# Create partial residual plots for each independent variable
fig, ax = plt.subplots(8, 2, figsize=(10, 12))
PartialDependenceDisplay.from_estimator(estimator, X_train, X_train.columns, ax = ax)

# Displat the partial residual plots
fig.suptitle('Partial Dependence Plots')
fig.tight_layout()

# variables shown significant linearity
```

```
# Create a correlation matrix of the independent variables
corr_matrix = X.corr()

# Display the correlation matrix
print(corr_matrix)

# there is no significant pair-wise correlation found

'''Comparing Prebuilt Sklearn model'''

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report

df_train = pd.read_csv(r"C:\\Users\\Laura\\Documents\\UC DAVIS\\STA 141C\\df_train.csv")
X = df_train.drop(columns = "Loan_Status") #drop the column 'Loan_Status'
Y = df_train["Loan_Status"]


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state= 0)
log_reg = LogisticRegression() #We fit the Logistic Regression model
log_reg.fit(X_train, y_train)

log_reg_pred = log_reg.predict(X_test)
print("Accuracy:" ,metrics.accuracy_score(log_reg_pred, y_test))
print(classification_report(y_test, log_reg_pred))

#Confusion Matrix
cm = confusion_matrix(y_test, log_reg_pred)
ax = plt.subplot()
sns.heatmap(cm, annot = True, fmt= 'g', ax=ax)
ax.set_title("Confusion Matrix")
plt.show()
#From the confusion matrix we can tell that 13 and 117 were predicted correctly. And 6 and 24 were predicted i

'''Logistic Regression model Sim. Data'''

simu_data = pd.read_csv(r"/Users/howie/Desktop/STA 141C/Final Project/Simulation_data.csv",index_col=0)
X_simu = simu_data.drop(columns = "Loan_Status") #drop the column 'Loan_Status'
Y_simu = simu_data["Loan_Status"]

X_train_simu, X_test_simu, y_train_simu, y_test_simu = train_test_split(X_simu, Y_simu, test_size=0.5, random_

# normalize the independent variables
X_train_simu = (X_train_simu - np.mean(X_train_simu, axis=0)) / np.std(X_train_simu, axis = 0)
X_test_simu = (X_test_simu - np.mean(X_test_simu, axis=0)) / np.std(X_test_simu, axis= 0 )

log_reg = LogisticRegression(learning_rate=0.1, num_iterations=1000)
log_reg.fit(X = X_train_simu, y = y_train_simu)
log_fitted = log_reg.fit(X = X_train_simu, y = y_train_simu)

y_pred = log_reg.predict(X = X_test_simu)
y_pred_prob = log_reg.predict1(X = X_test_simu)
log_odds = np.log(y_pred_prob / (1 - y_pred_prob))
```

```
print("Accuracy:" ,metrics.accuracy_score(y_pred, y_test_simu))

print(classification_report(y_test_simu, y_pred))

#Confusion Matrix
cm = confusion_matrix(y_test_simu, y_pred)
ax = plt.subplot()
sns.heatmap(cm, annot = True, fmt= 'g', ax=ax)
ax.set_title("Confusion Matrix")
ax.xaxis.set_ticklabels(['0 (predicted)', '1 (predicted)'])
ax.yaxis.set_ticklabels(['0 (true)', '1 (true)'])
plt.show()

# ensure the model output is an estimator
estimator = make_pipeline(
    StandardScaler(),
    PolynomialFeatures(degree=2),
    TransformedTargetRegressor(regressor=LogisticRegression(), transformer=StandardScaler())
)

estimator.fit(X = X_train_simu, y = y_train_simu)

# dis-normalize the training data (does not affect anything, but the numbers will look nice)
# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state= 0)

# Create partial residual plots for each independent variable
fig, ax = plt.subplots(8, 2, figsize=(10, 12))
PartialDependenceDisplay.from_estimator(estimator, X_train_simu, X_train_simu.columns, ax = ax)

# Displat the partial residual plots
fig.suptitle('Partial Dependence Plots')
fig.tight_layout()

# Create a correlation matrix of the independent variables
corr_matrix = X_simu.corr()

# Display the correlation matrix
print(corr_matrix)

# there is no significant pair-wise correlation found

'''Generating the Sim. Data'''

df = pd.read_csv(r"C:\Users\ovied\OneDrive\Desktop\loan_sanction_train.csv")
df['Credit_History'] = df['Credit_History'].fillna(0)          #fills the missing credit history with 0
df['Dependents'] = df['Dependents'].fillna(0)                #fills the missing dependent count with 0
df = df.dropna()
df = df.drop(['Loan_ID'], axis=1)
df = df.replace({'Dependents': { 0: '0'}})
df = df[df['ApplicantIncome'] < 14000]  #removes extreme values
df = df[df['CoapplicantIncome'] < 7000]

#categorical variable to binary values
df = df.replace({'Loan_Status': {'Y': 1, 'N': 0}})
df = df.replace({'Gender': {'Male': 1, 'Female': 0}})
df = df.replace({'Married': {'Yes': 1, 'No': 0}})
```

```
df = df.replace({'Education': {'Graduate': 1, 'Not Graduate': 0}})
df = df.replace({'Self_Employed': {'Yes': 1, 'No': 0}})

df.iloc[:, :].mean()

df['Dependents'].value_counts(normalize=True)

df['Property_Area'].value_counts(normalize=True)

np.random.seed(42)

'''Creates the numerical variables based on the minimum and maximum of the original data set'''

ApplicantIncome = np.random.uniform(low=df['ApplicantIncome'].min(),
high=df['ApplicantIncome'].max(), size=5000)

CoapplicantIncome = np.random.uniform(low=df['CoapplicantIncome'].min(),
high=df['CoapplicantIncome'].max(), size=5000)

LoanAmount = np.random.uniform(low=df['LoanAmount'].min(), high=df['LoanAmount'].max(), size=5000)

Loan_Amount_Term = np.random.choice(df['Loan_Amount_Term'].unique(), 5000,
p = df['Loan_Amount_Term'].value_counts(normalize=True).values)

'''Creates the categorical variables based on the true value proportions of the original data set'''

Gender = np.random.choice([0, 1], 5000, p=[1-0.816206, 0.816206])
Married = np.random.choice([0, 1], 5000, p=[1-0.650198, 0.650198])
Dependents = np.random.choice(['0', '1', '2', '3+'], 5000, p=[0.583004, 0.171937, 0.166008, 0.079051])
Education = np.random.choice([0, 1], 5000, p=[1-0.778656, 0.778656])
Self_Employed = np.random.choice([0, 1], 5000, p=[1-0.122530, 0.122530])
Credit_History = np.random.choice([0, 1], 5000, p=[1-0.778656, 0.778656])
Property_Area = np.random.choice(['Semiurban', 'Urban', 'Rural'], 5000, p=[0.389328, 0.320158, 0.290514])
Loan_Status = np.random.choice([0, 1], 5000, p=[1-0.695652, 0.695652])


'''creates pandas dataframe'''

df = pd.DataFrame({
    'ApplicantIncome': ApplicantIncome,
    'CoapplicantIncome': CoapplicantIncome,
    'LoanAmount': LoanAmount,
    'Loan_Amount_Term': Loan_Amount_Term,
    'Gender': Gender,
    'Married': Married,
    'Dependents': Dependents,
    'Education': Education,
    'Self_Employed': Self_Employed,
    'Credit_History': Credit_History,
    'Property_Area': Property_Area,
    'Loan_Status': Loan_Status
})

#changes remaining categorical variables to binary values
df = pd.get_dummies(df)
```

```
df.to_csv(r'C:\Users\ovied\Downloads\Simulation_data.csv')
```

9 References

- Approval, Home Loan. Kaggle. n.d. https://www.kaggle.com/datasets/rishikeshkonapure/home-loan approval. 01 March 2023.
- Prasad, Ashwin. Medium. 14 June 2021. https://medium.com/analytics-vidhya/logistic-regression-with-gradient-descent-explained-machine-learning-a9a12b38d710. 22 March 2023.
- Solutions, Statistics. Complete Dissertation. 2023. https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/assumptions-of-logistic-regression/. 21 March 2023.
- I-King-Of-Ml. (2019, October 31). KNN(K-nearest neighbour) algorithm, maths behind it and how to find the best value for K. Medium. Retrieved March 23, 2023, from https://medium.com/@rdhawan201455/knn-k-nearest-neighbour-algorithm-maths-behind-it-and-how-to-find-the-best-value-for-k-6ff5b0955e3d