Intro

Associating risk with clients that have applied for a loan has important implications for banks. Before issuing loans to applicants, a bank might want to check how likely it is for a client to default. There are three questions we sought to answer with this data set. The first, is there a connection between high credit balances and bad risk class label? Second, do clients with similar features have identical class labels? Third, can a client's new credit balance be used as a good predictor of what will be a client's highest credit balance?

IDA

In this phase the raw data set was cleaned in order to facilitate analysis in subsequent phases.

Some of the columns in the raw data set were irrelevant for predicting whether a client was good or bad risk to default, so these columns were discarded.

For some clients, the raw data set contained numerous rows for a single client, whereas other clients only had one row. These rows corresponded to data for different dates. For example, there would be one row containing data for April and another row containing data for May. Both of these rows would correspond to a single client. In order to simplify our analysis, only one of these entries was kept.

For dealing with missing data, the mean or median value of the column was used. Some clients had high values for bank account balance relative to the majority of other clients. Substituting missing values with the column's mean value in this case would not be appropriate, because the mean value is heavily affected by the high balances of a few clients. For columns where this was the case, the missing value was substituted with the column's median value. The median value was also used for replacing missing values in columns with categorical data, such as type of product. For columns with low variance, the missing value was replaced with the column's mean value.

To easily find data belonging to a specific client we decided it would be appropriate to index the data set by client ID.

In order to reduce the dimensionality of the feature space, we aggregated the different type of overdue payments into a single column. This column represents the number of times a client made an overdue payment.

EDA

Our focus in this phase was to find insights that either would help us develop a better understanding of the data, or reveal patterns in the data. The latter helped us select appropriate models to use in the next phase.

The data set initially contained a significant imbalance in the class labels. Out of the 1,125 samples, 900 of these had good risk labels. To not bias our analysis, we made the amount of class labels equal. This left us with 450 samples in our data set (we'll come back to this point in the recommendations section).

Another one of our discoveries in this phase was that clients with high credit amounts owed were about four times more likely to be labeled as a bad risk. We made this discovery by finding the number of clients above an agreed upon threshold of $500,000 for each class label.

We also found an interesting pattern between two of the features. We used seaborn to produce two-dimensional scatter plots of all the features. We noticed that there was a linear relationship between columns 'new_balance' and 'highest_balance.' The 'new_balance' feature represents the current balance a client owes, whereas the 'highest_balance' feature represents the highest amount a client has owed. With this information we can predict what a clients highest balance owed is given their current balance.

Out of the 17 different kinds of credit products contained in the data set, we found 62% of clients had product 10. There was 90% of clients having products 5, 6, and 10. This suggests the kind of credit products is

To get a better understanding of the data contained in each of the columns, we determined the maximum and minimum values, mean, and standard deviation for each column. The max values helped us find outliers in the amount of credit owed. The min values helped us detect negative values, which was useful for detecting anomalies that can perhaps be attributed to mistakes in documenting the data. The mean sometimes gave us representative values of the column, whereas the standard deviation gave us insight into how dispersed the values were from the mean.

Modeling

In this phase we used machine learning to develop models that capture patterns found in the data. I can't overstate how useful performing PCA was for our analysis. After plotting the transformed data set on a two-dimensional plot, we noticed amazing clusters and subclusters. This gave us valuable insight into what models we were going to select.

K-Nearest Neighbors

KNN was an obvious choice because identical class labels tended to be near one another. We used L2 norm to measure similarity between the data points. Several k values were used to see which value would produce the best results. We found that a k value of 7 produced the best results. The accuracy was 89%, the true positive rate was 65%, and the false positive rate was 10%.

K-Means

K-means was used to associate data points with one of three clusters. Three clusters were chosen because it is apparent from the plot of the transformed data that there are three clusters. This is useful because knowing which cluster the data point is in, and where, gives an indication as to its class label. We found that 100% of the tested data points were associated with the correct cluster.

Decision Tree

We chose decision tree in order to have a model that learns decision rules based on the training data. It is a popular classification algorithm, and it can be applied a wide range of problems. It is also simple when compared to other classification algorithms. An interesting aspect of decision trees is that they consider possible traces in a path to assign a class label to a data point. Decision Trees are versatile in that they can be applied to problems in various contexts. The accuracy for our model was 84%, the true positive rate was 62%, and the false positive rate was 12%.

Linear Regression

While performing EDA, I noticed a linear relationship in the scatter plot of highest balance versus new balance. This was useful because it meant we could use information about a client's new credit balance to predict what their highest credit balance will be. We can use this information to associate a client with a class label, because those with high credit balances were found to be more likely to be classified as a bad risk. Our model did a great job of capturing the patterns found in the test data. It had a coefficient of determination value of .87. We noticed that the performance of the model improved significantly as we fed the model more training data. This suggests we can obtain much better predictions of what a client's highest balance will be given their new credit balance.

Logistic Regression

We chose logistic regression because it provides an intuitive probabilistic perspective on making classification predictions. To evaluate the model, we used accuracy, recall, precision, and false positive rate. The accuracy for our model was 86.7%, the recall was 60%, the precision was 95%, and the false positive rate was 3.7%. Of the three classification models we used, this one had the second-best accuracy. One interesting observation is that the false positive rate is only 3.7% (far better than our other classification models).

Instructions for use with different data sets

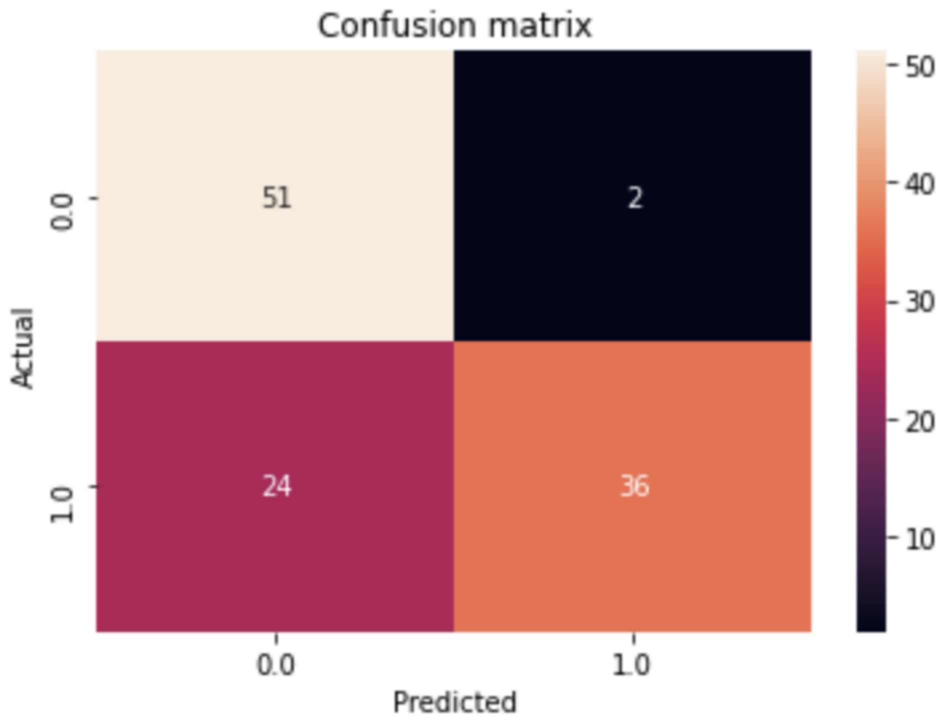
The analysis performed in this report was done within the context of helping banks predict whether a client is a good or bad risk. Therefore, we would advise using the approaches found in this report for a similar data set.

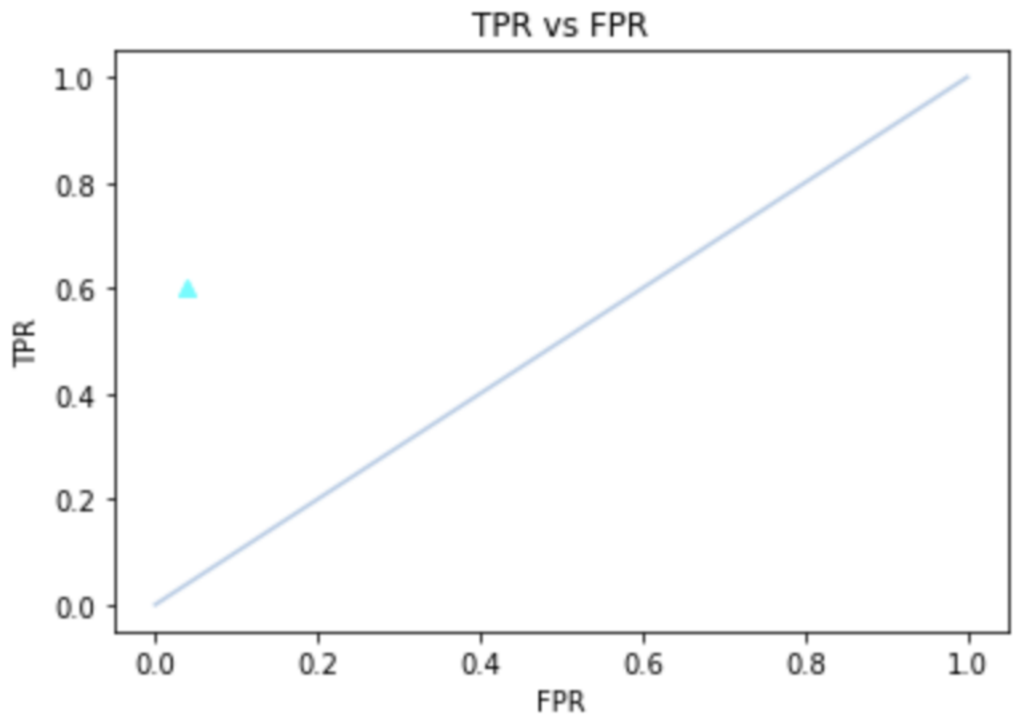
Recommendations

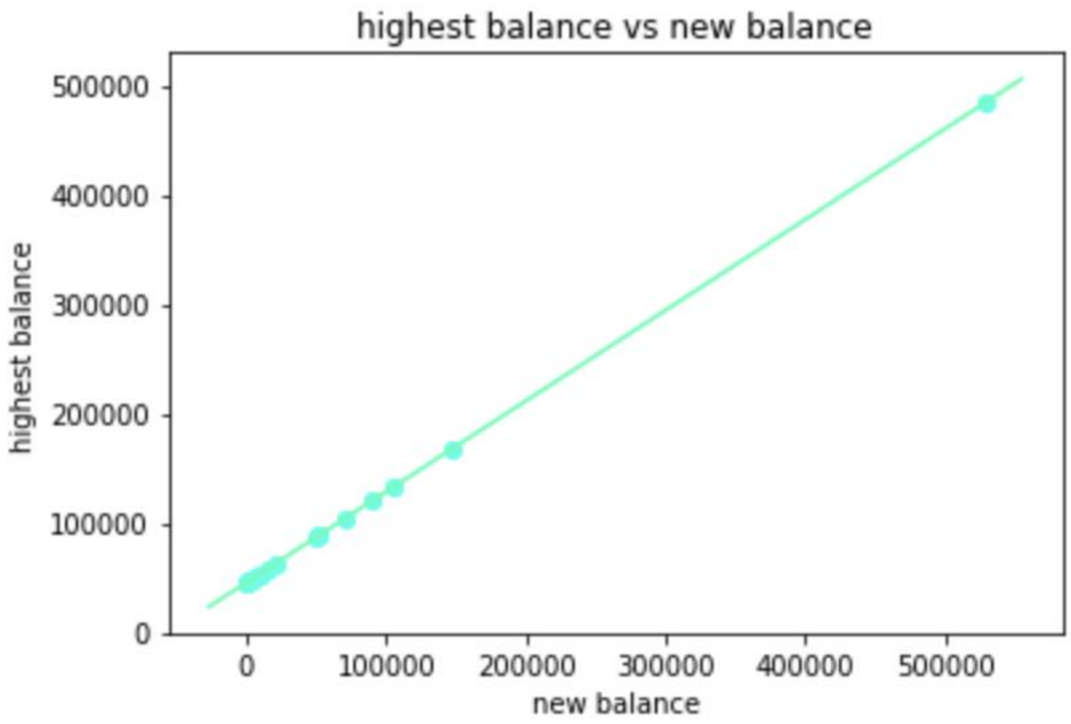
In order to remove the imbalance in class labels, we had to reduce our sample size from 1125 to 450. Models perform better when they have a lot of data to learn from, so, in order to improve performance, acquiring more data is recommended.

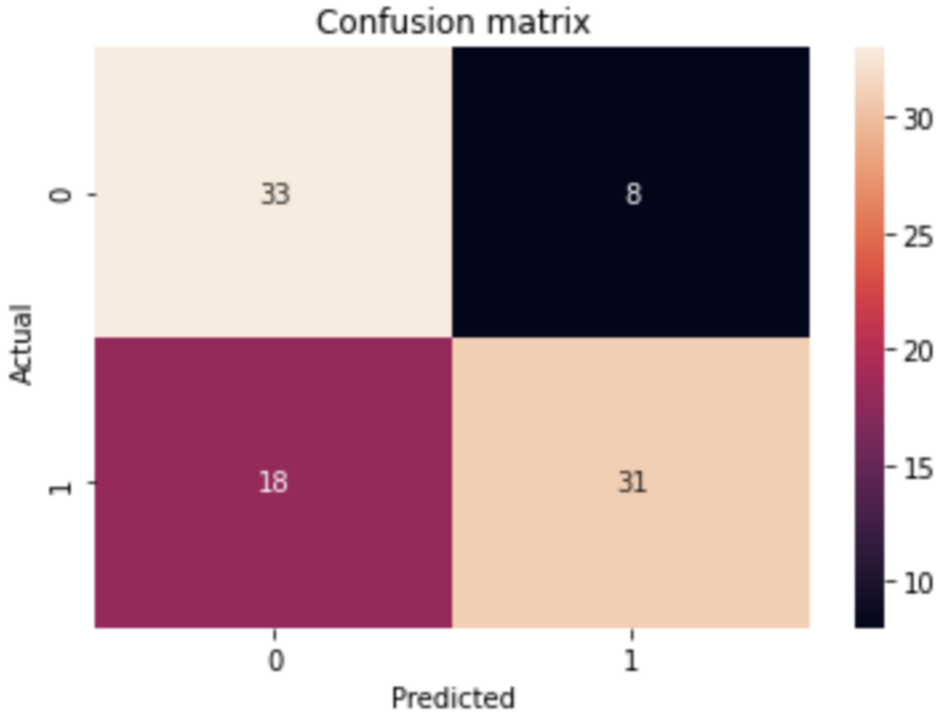
As mentioned previously, we found that clients with high credit amounts owed were about four times more likely to be labeled as a bad risk. We would advise granting loans to clients that have the financial resources to pay it back.

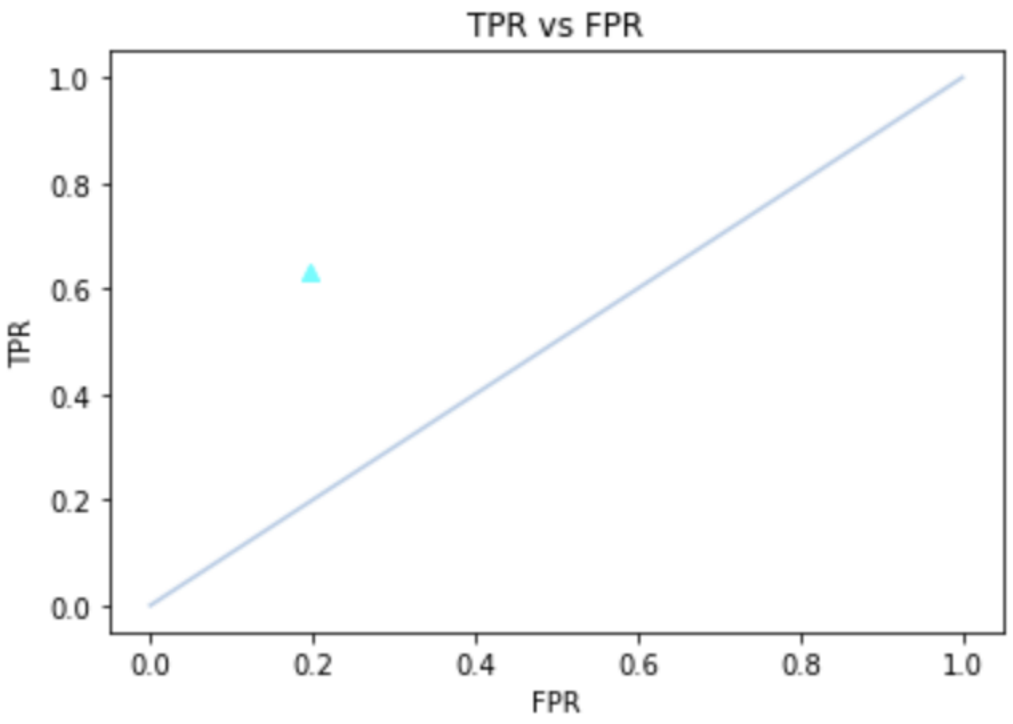
Figures

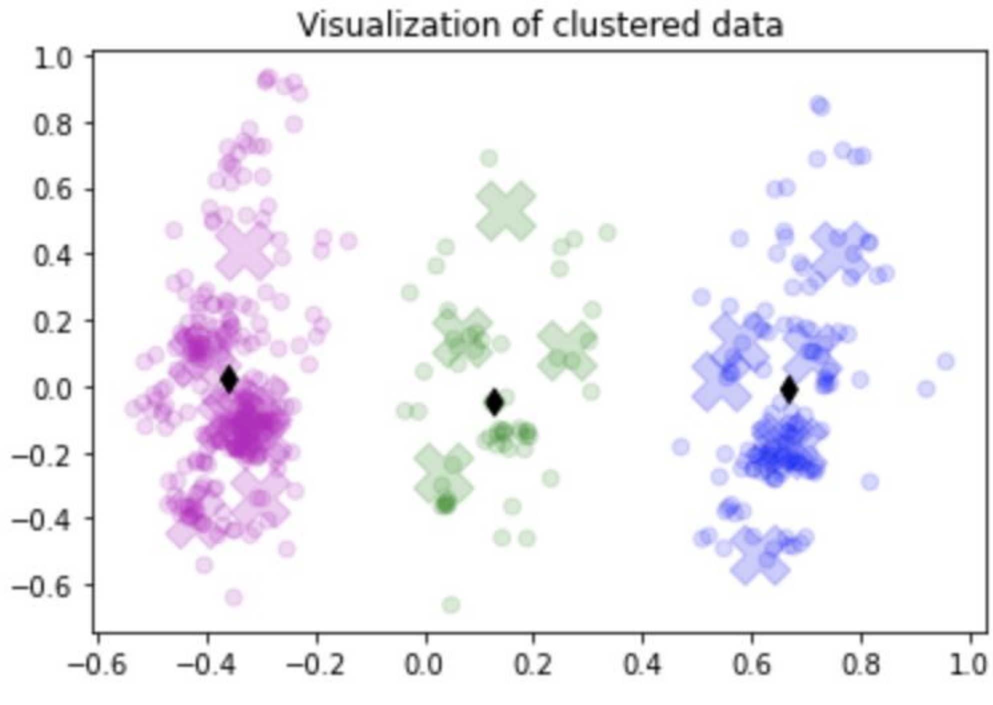

Confusion matrix
Actual
Predicted
0.0
1.0
0.0
1.0
51
2
24
36
50
40
30
20
10


TPR vs FPR
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

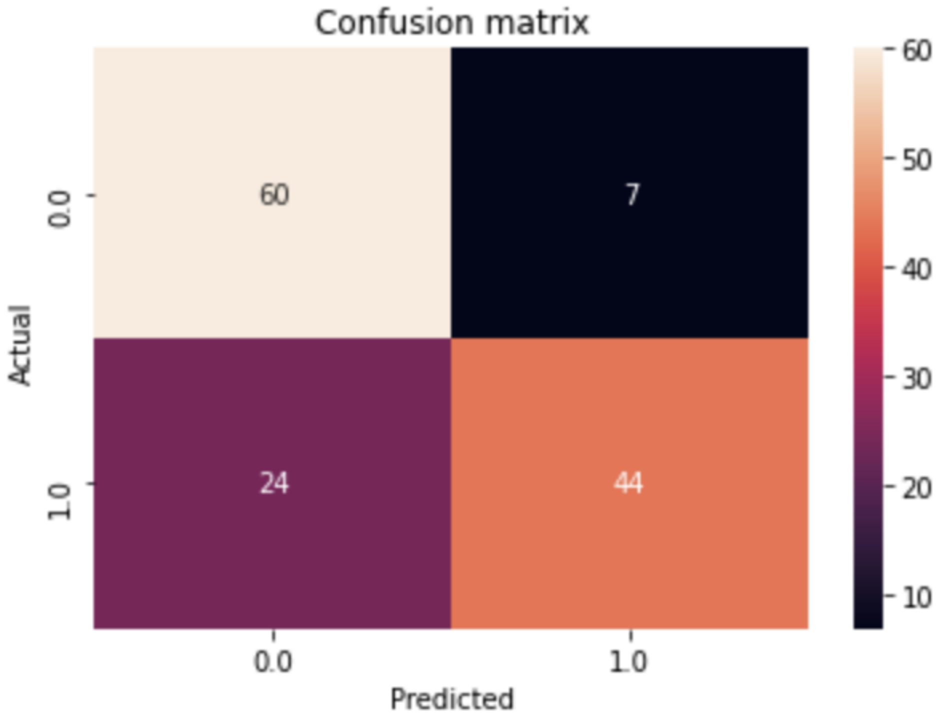

highest balance vs new balance
highest balance
new balance
500000
400000
300000
200000
100000
0
0
100000
200000
300000
400000
500000


Confusion matrix
Actual
Predicted
33
8
18
31
0
1
0
1
30
25
20
15
10

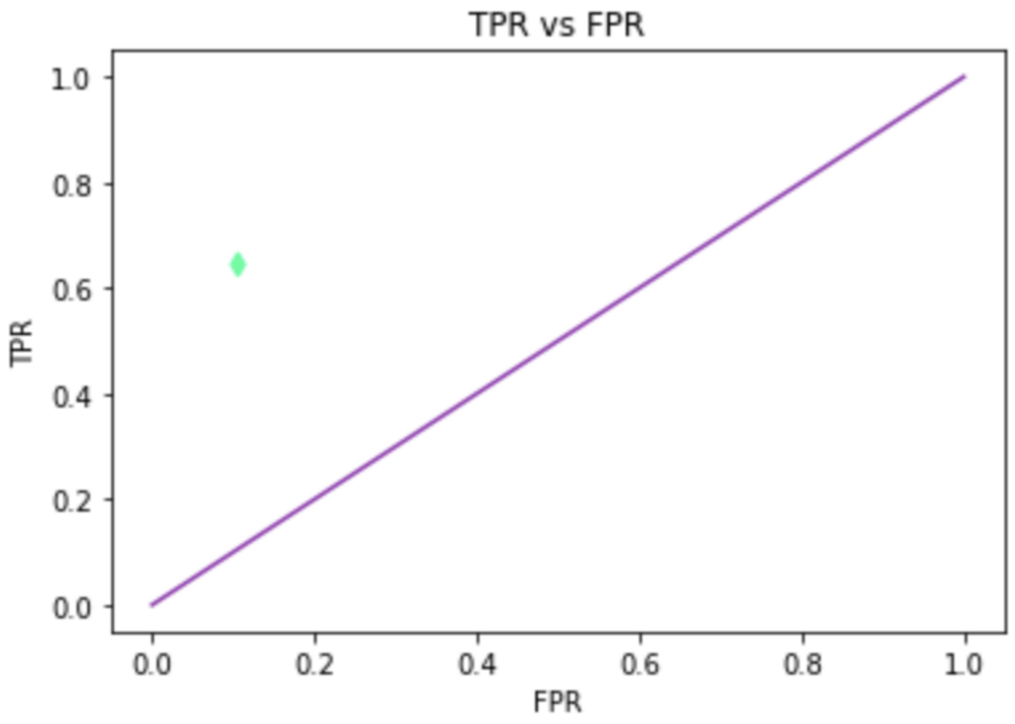

TPR vs FPR
TPR
FPR
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Visualization of clustered data
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

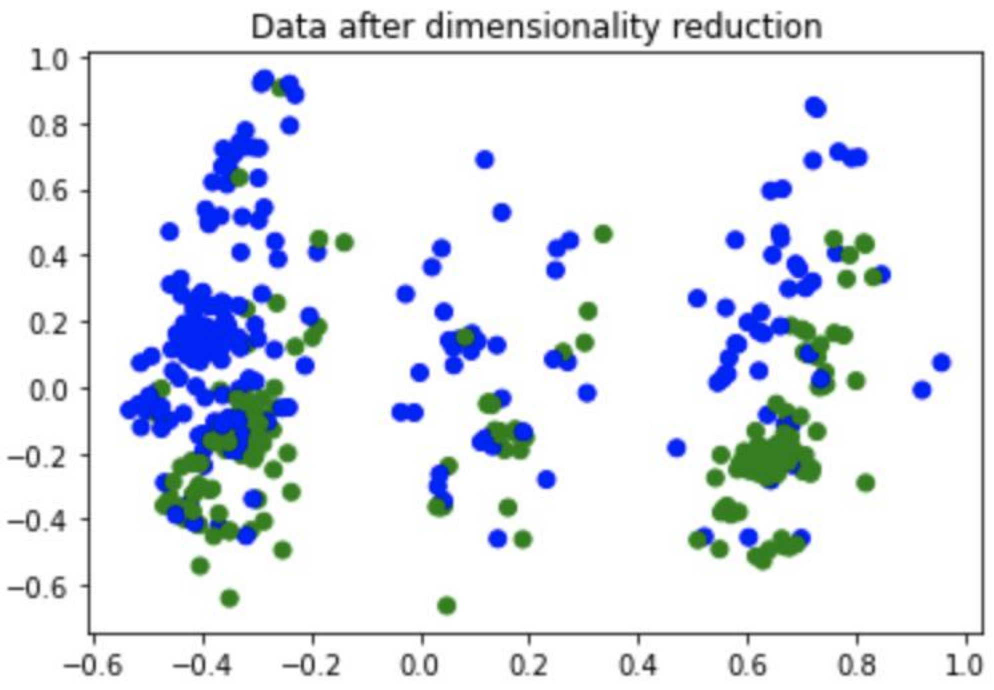
FPR

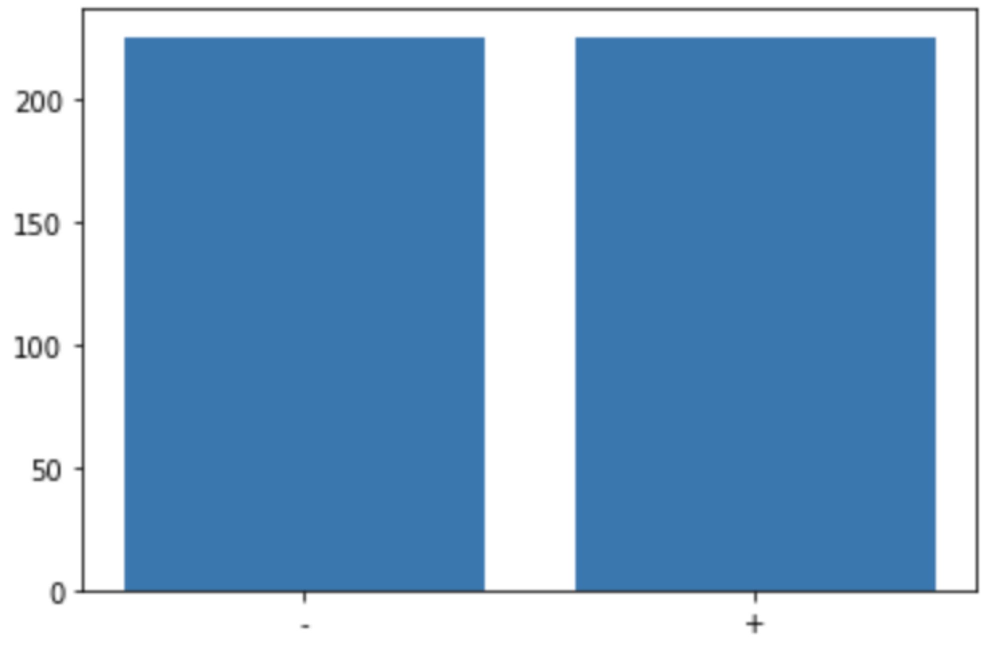


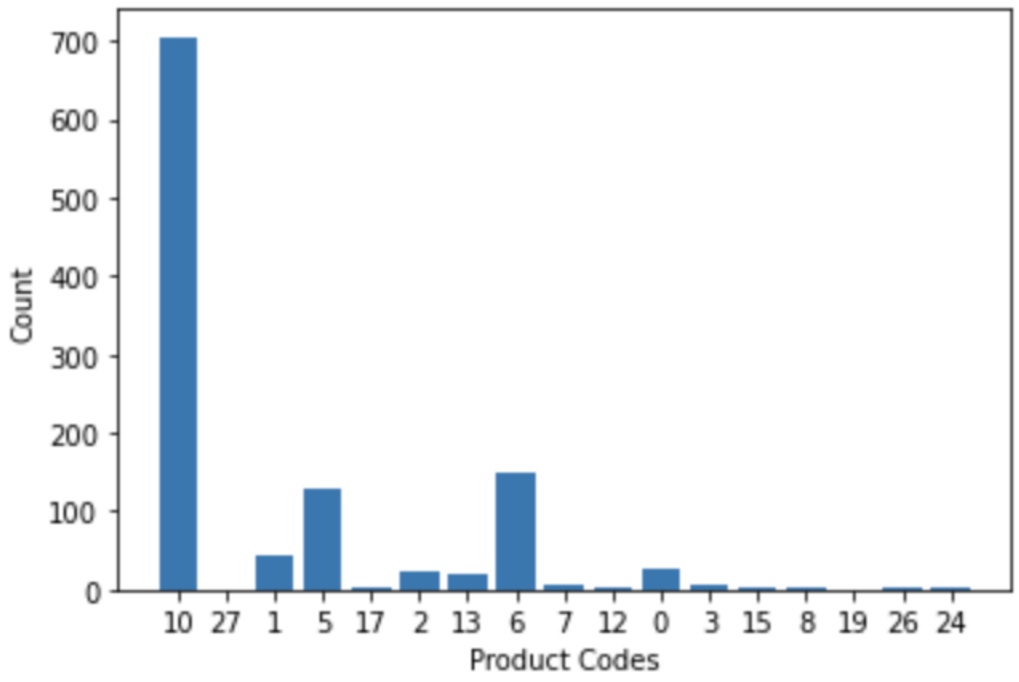
Confusion matrix
Actual
Predicted
0.0
1.0
0.0
1.0
60
7
24
44
60
50
40
30
20
10

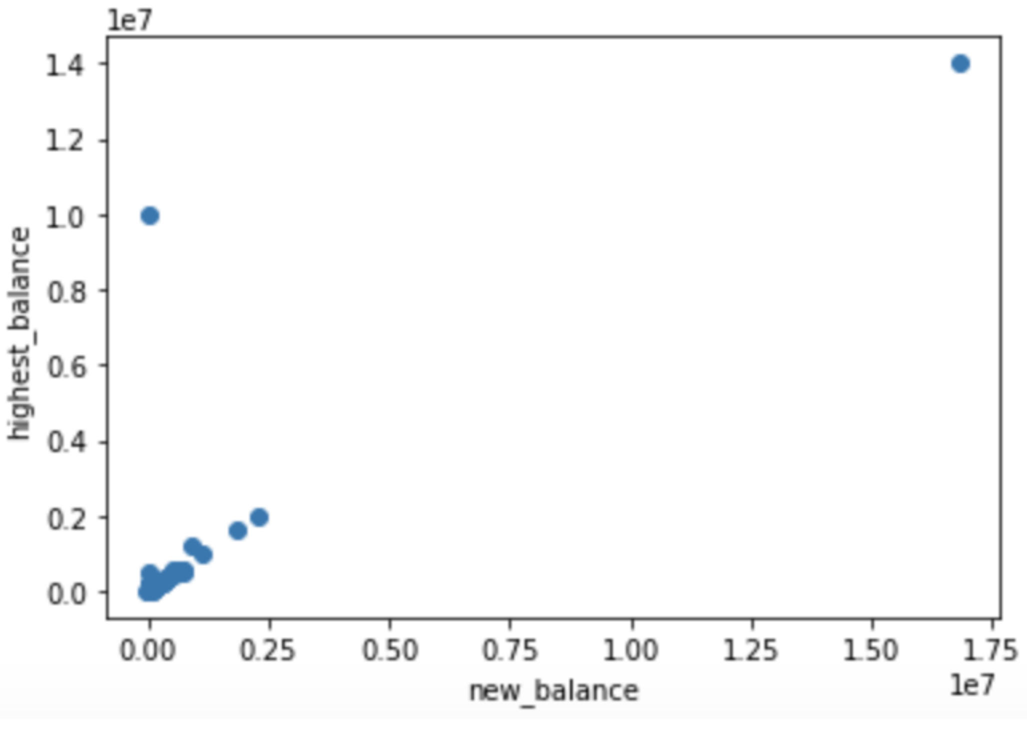


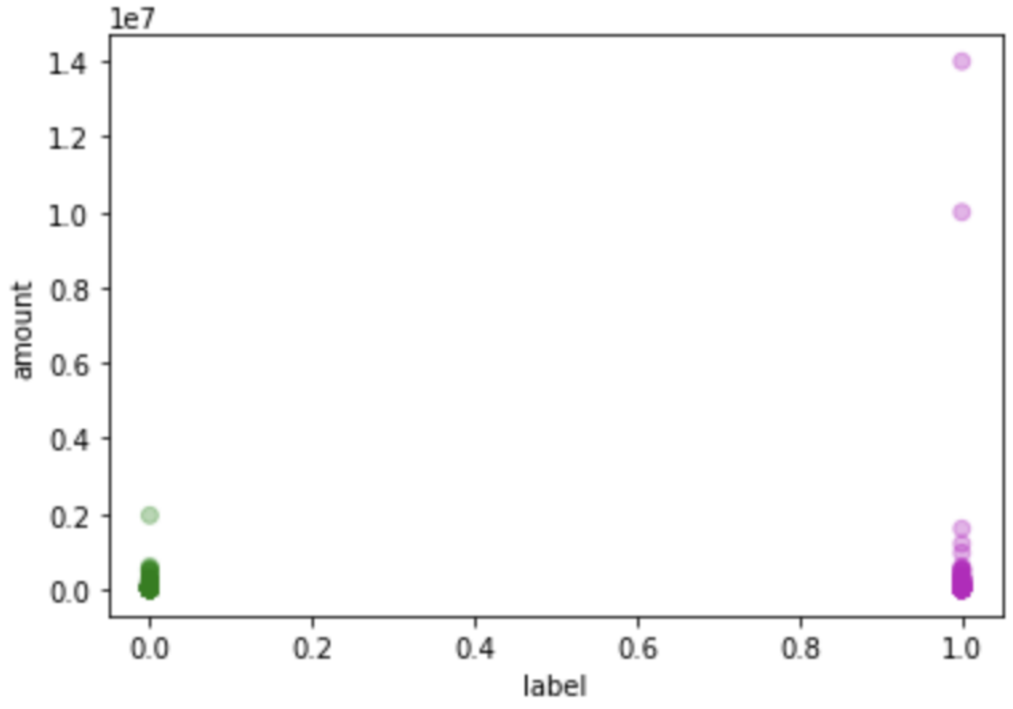
TPR vs FPR
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


Data after dimensionality reduction
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0


200
150
100
50
0
-
+


Count
700
600
500
400
300
200
100
0
10 27 1 5 17 2 13 6 7 12 0 3 15 8 19 26 24
Product Codes


1e7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
highest_balance
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75
1e7
new_balance



Code + Documentation

```
#Phase 1
import pandas as pd
#read data sets
data_set1=pd.read_csv('/Users/josereyes/Documents/Data_Phase1/data/customer_data.csv')
data_set2=pd.read_csv('/Users/josereyes/Documents/Data_Phase1/data/payment_data.csv')
# change type of id Series elements in order to perform fancy indexing
data_set1['id']=data_set1['id'].astype(str)
data_set2['id']=data_set2['id'].astype(str)
ids=list(data_set1['id'])
# have id Series be the index
data_set1=data_set1.set_index('id')
data_set2=data_set2.set_index('id')
#perform fancy indexing
df=data_set2.loc[ids]
df=df[~df.index.duplicated(keep='first')]
#change index type and sort
df.index=df.index.astype('int64')
df= df.sort_index()
data_set1.index=data_set1.index.astype('int64')
data_set1= data_set1.sort_index()
#join the two data frames
df=data_set1.join(df)
```

```
# Operation - aggregate number of times overdue for each type into one column to reduce the
dimensionality of the data
df['OVD']=df['OVD_t1']+df['OVD_t2']+df['OVD_t3']
# Operation - One-hot encode OVD to indicate the presence of any overdue payments in order
to make this column more pertinent to problem
for i in range(len(df['OVD'])):
  if df.iloc[i,-1]!=0:
    df.iloc[i,-1]=1
# remove columns prod_limit,update_date, and report_date, since the majority of values are
NaN
# remove OVD_t1, OVD_t2, OVD_t3 since this information is already captured in OVD
df.drop('report_date', inplace=True, axis=1)
df.drop('update_date', inplace=True, axis=1)
df.drop('prod_limit', inplace=True, axis=1)
df.drop('OVD_t1', inplace=True, axis=1)
df.drop('OVD_t2', inplace=True, axis=1)
df.drop('OVD_t3', inplace=True, axis=1)
# Operation - fill NaN values to allow machine learning algorithm to make better predictions
#I chose mean value because there was low variance in fea_2 feature
df['fea_2']=df['fea_2'].fillna(df['fea_2'].mean())
#I chose median value because there was high variance in highest_balance feature
df['highest_balance']=df['highest_balance'].fillna(df['highest_balance'].median())
```

#Phase 2

```
# import cleaned dataset and display
import pandas as pd
df=pd.read_csv('data/DataFrame.csv')
df.head(5)
# display the first five rows data.head(5)
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#find which values in highest_balance are greater than 550000 and have bad risk
mask=(df['highest_balance']>550000) & (df['label']==1)
#find sum
highrisk=mask.sum()
#find which values in highest_balance are greater than 550000 and have good risk
mask=(df['highest_balance']>550000) & (df['label']==0)
#take sum
lowrisk=mask.sum()
#initialize list
colors = []
#replace 0 and 1 with the magenta or green for scatter plot
for e in df['label']:
    if e ==1:
        colors.append('m')
    else:
        colors.append('g')
#produce scatter plot
plt.scatter(df['label'], df['highest_balance'], c=colors, alpha=.3)
#label x axis
plt.xlabel('label')
#label y axis
plt.ylabel('amount')
#show plots
plt.show()
#drop new_balance
df.drop('new_balance', inplace=True, axis=1)
#slice data frame
data=df.loc[:,'OVD_sum':'OVD'].copy()
# make two-dimensional plot of each feature
h=sns.PairGrid(data,palette='RdBu_r')
h.map(plt.scatter, alpha=0.8)
#Initialize dictionary
DiversityofProducts={}
#count how many of each product is contained in the data
for element in df['prod_code']:
    if element in DiversityofProducts:
        DiversityofProducts[element]+=1
    else:
        DiversityofProducts[element]=1
#convert to series
s=pd.Series(DiversityofProducts)
# plot
plt.bar(x=s.index.to_numpy(), height=s.values)
#label x axis
plt.xlabel('Product Codes')
#label y axis
plt.ylabel('Count')
# remove imbalance in class labels
j=0
while df.shape[0]>450:
    if df['label'].iloc[j]==0:
        df.drop(df.index[j], inplace=True, axis=0)
    else:
        j+=1
#get numpy array of label values
lab_values=df['label'].values
#count number of times 0 appears
```

```
count0=np.equal(lab_values,0).sum()
#count number of times 1 appears
count1=np.equal(lab_values,1).sum()
#plot
plt.bar([1,2],height=[count0,count1])
#set tick
plt.xticks([1,2],['-','+'])
```

#Phase 3

```
# import your cleaned dataset
from google.colab import files
uploaded = files.upload()
import pandas as pd
#read data
df = pd.read_csv('Data.csv')
# display the first five rows data.head(5)
df.head(5)
#import relevant items
import matplotlib.pyplot as plt
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
import sklearn.model_selection
from sklearn.preprocessing import MinMaxScaler
# copy data frame
Z=df.copy().sample(frac=1,random_state=0)
#normalization
norm = MinMaxScaler().fit(Z)
#transform the data
Z = norm.transform(Z)
#convert to data frame
Z=pd.DataFrame(Z,index=df.index,columns=df.columns)
#assign subset of data to X
X=Z.loc[:,'fea_1':'fea_11']
#assign labels to y
y=Z['label']
#import PCA
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
#fit PCA
pca.fit(X)
#perform dimensionality reduction
X=pca.transform(X)
#split the data
Xtrain, Xtest, ytrain, ytest = sklearn.model_selection.train_test_split(X,
y, train_size=0.7,random_state=0)
#Instantiate knn classifier
```

```
knn=KNeighborsClassifier(n_neighbors=7)
#determine best accuracy
Accuracy=np.round(100*np.max(cross_val_score(knn, X, y, cv=10)),1)
print(Accuracy)
#fit knn model
knn=knn.fit(Xtrain,ytrain)
# predict y values
ypred=knn.predict(Xtest)
#initialize number of true positives
TP=0
#initialize number of false positives
FP=0
#initialize number of true negatives
TN=0
#initialize number of false negatives
FN=0
# count number true positves, true negatives, false positives, and false
negatives
for i in range(len(ypred)):
    if ypred[i]==1 and ytest.iloc[i]==1:
        TP+=1
    elif ypred[i]==1 and ytest.iloc[i]==0:
        FP+=1
    elif ypred[i]==0 and ytest.iloc[i]==1:
        FN+=1
    elif ypred[i]==0 and ytest.iloc[i]==0:
        TN+=1
#Calculate true positive rate
TPR=TP/(TP+FN)
#Calculate precision
Precision=TP/(TP+FP)
#calculate false positive rate
FPR=FP/(TN+FP)
#calculate true negative rate
TNR=TN/(TN+FP)
#calculate false negative rate
FNR=FN/(TP+FN)
#calculate prevalence
Prevalence=(TP+FN)/len(ytest)
#intialize list
c=[]
#replace 1 and 0 with ‘b’ and ‘g’
for e in y:
    if e==1:
        c.append('b')
    else:
        c.append('g')
#create figure
plt.figure(1)
#display scatter plot
plt.scatter(X[:,0],X[:,1],c=c)
#display title
plt.title('Data after dimensionality reduction')
#create figure
plt.figure(2)
#create points for comparison with (TNR,FPR)
x=np.linspace(0,1,100)
#make plot
plt.plot(x,x,color=(.6,.2,.7))
#make scatter plot
plt.scatter(FPR,TPR,marker='d',color=(0,1,.6))
#label y axis
plt.ylabel('TPR')
#label x axis
plt.xlabel('FPR')
#title plot
plt.title('TPR vs FPR')
#make plot
plt.figure(3)
#display confusion matrix
confusion_matrix = pd.crosstab(ytest, ypred, rownames=['Actual'],
colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
#titlw confusion matrix
plt.title('Confusion matrix')
```
# import relevant items
```
import sklearn.model_selection
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import PCA
#make copy of data frame
Z=df.copy().sample(frac=1,random_state=0)
#fit MaxMinScaler
norm = MinMaxScaler().fit(Z)
#transform the data
Z = norm.transform(Z)
#create new data frame
Z=pd.DataFrame(Z,index=df.index,columns=df.columns)
```

```
#same
X=Z.loc[:,'fea_1':'fea_11']
#same
y=Z['label']
#initialze pca object
pca=PCA(n_components=2)
#fit to data
pca.fit(X)
#perform dimensionality reduction on data
X=pca.transform(X)
#create data split
Xtrain, Xtest, ytrain, ytest = sklearn.model_selection.train_test_split(X,
y, train_size=0.95,random_state=1)
#initialize cluster centers
init=[[-.4,.2],[.2,.2],[.7,.2]]
#initialize kmeans
kmeans = KMeans(n_clusters=3, init=np.array(init))
#fit it to data
kmeans=kmeans.fit(Xtrain)
#same
colors=[]
for e in kmeans.labels_:
    if e ==0:
        colors.append('m')
    elif e==1:
        colors.append('g')
    elif e==2:
        colors.append('b')

#Predict cluster
ypred=kmeans.predict(Xtest)
#same
ctest=[]
for e in ypred:
    if e ==0:
        ctest.append('m')
    elif e==1:
        ctest.append('g')
    elif e==2:
        ctest.append('b')

#Fit
plt.scatter(Xtrain[:,0], Xtrain[:,1], c=colors, alpha=0.15)
#Test
plt.scatter(Xtest[:,0],Xtest[:,1],c=ctest,marker='X',s=500,alpha=.2)
```

```
#Cluster centers
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=[5
0,50,50],marker='d',c='k')
#same
plt.title('Visualization of clustered data')
#import relevant items
import sklearn
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
#same
Z=df.copy().sample(frac=1,random_state=0)
#same
X=Z.loc[:,'fea_1':'fea_11']
#same
y=Z['label']
#initialize  decision tree
dt = DecisionTreeClassifier(random_state=0)
#same
Xtrain, Xtest, ytrain, ytest = sklearn.model_selection.train_test_split(X,
y, train_size=0.8,random_state=0)
#same
Accuracy=np.round(100*np.max(cross_val_score(dt, X, y, cv=10)),1)
#same
model=dt.fit(Xtrain,ytrain)
#same
ypred=model.predict(Xtest)
#same
TP=0
FP=0
TN=0
FN=0
#same
for i in range(len(ypred)):
    if ypred[i]==1 and ytest.iloc[i]==1:
        TP+=1
    elif ypred[i]==1 and ytest.iloc[i]==0:
        FP+=1
    elif ypred[i]==0 and ytest.iloc[i]==1:
        FN+=1
    elif ypred[i]==0 and ytest.iloc[i]==0:
        TN+=1
```

```
#same
TPR=TP/(TP+FN)
Precision=TP/(TP+FP)
FPR=FP/(TN+FP)
TNR=TN/(TN+FP)
FNR=FN/(TP+FN)
Prevalance=(TP+FN)/len(ytest)
#same
x=np.linspace(0,1,100)
#same
plt.figure(1)
plt.plot(x,x,color=(.7,.8,.9))
plt.scatter(FPR,TPR,marker='^',color=(.1,1,1))
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.title('TPR vs FPR')
plt.figure(2)
#same
confusion_matrix = pd.crosstab(ytest, ypred, rownames=['Actual'],
colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
plt.title('Confusion matrix')
#import relevant items
import numpy as np
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt  # To visualize
import pandas as pd  # To read data
import sklearn.metrics
from sklearn.linear_model import LinearRegression
#same
Z=df.copy().sample(frac=1,random_state=0)
#assign X to new balance column
X = Z['new_balance'] # values converts it into a numpy array
#assign y to highest balance column
y = Z['highest_balance'] # -1 means that calculate the dimension of rows,
but have 1 column

linear_regressor = LinearRegression()  # create object for the class
Xtrain, Xtest, ytrain, ytest = sklearn.model_selection.train_test_split(X,
y, train_size=0.95,random_state=0)
model=linear_regressor.fit(Xtrain.values.reshape(-1, 1),
ytrain.values.reshape(-1, 1))  # perform linear regression
# get coefficient
param=model.coef_[0]
#get y-intercept
```

```
intercept=model.intercept_
#predict
ypred = model.predict(Xtest.values.reshape(-1, 1))  # make predictions
#determine R² value
r2=r2_score(ytest,ypred)
#same
plt.scatter(Xtest, ypred,color=(0,1,.9))
#create function that creates a line representing the model on plot
def line(s, i):
    axes = plt.gca()
    x = np.array(axes.get_xlim())
    y = i + s * x
    plt.plot(x, y, '-',color=(.2,1,.7))
#call function
line(param,intercept)
#same
plt.xlabel('new balance')
#same
plt.ylabel('highest balance')
#same
plt.title('highest balance vs new balance')
#import relevant items
import sklearn.linear_model
import seaborn as sn
#create copy of data
Z=df.copy().sample(frac=1,random_state=0)
#normalize
norm = MinMaxScaler().fit(Z)
#transform data
Z = norm.transform(Z)
#make data frame
Z=pd.DataFrame(Z,index=df.index,columns=df.columns)
#assign x to slice of data frame
X=Z.loc[:,'fea_1':'fea_11']
#assign labels to y
y=Z['label']
# create data split
Xtrain, Xtest, ytrain, ytest = sklearn.model_selection.train_test_split(X,
y, train_size=0.75, random_state=0)
#initialize logistic regresiion classifier
logistic_regression = sklearn.linear_model.LogisticRegression()
#calculate accuracy
Accuracy=np.round(100*np.max(cross_val_score(logistic_regression, X, y,
cv=10)),1)
#fit the model
```

```
model=logistic_regression.fit(Xtrain, ytrain)
#make predictions
ypred = model.predict(Xtest)
#same as before
TP=0
FP=0
TN=0
FN=0
#same as before
for i in range(len(ypred)):
    if ypred[i]==1 and ytest.iloc[i]==1:
        TP+=1
    elif ypred[i]==1 and ytest.iloc[i]==0:
        FP+=1
    elif ypred[i]==0 and ytest.iloc[i]==1:
        FN+=1
    elif ypred[i]==0 and ytest.iloc[i]==0:
        TN+=1
#same as before
TPR=TP/(TP+FN)
Precision=TP/(TP+FP)
FPR=FP/(TN+FP)
TNR=TN/(TN+FP)
FNR=FN/(TP+FN)
Prevalance=(TP+FN)/len(ytest)

#same as before
x=np.linspace(0,1,100)
#same as before
plt.figure(1)
#same as before
plt.plot(x,x,color=(.7,.8,.9))
#same as before
plt.scatter(FPR,TPR,marker='^',color=(.1,1,1))
#same as before
plt.ylabel('TPR')
plt.xlabel('FPR')
#same as before
plt.title('TPR vs FPR')
plt.figure(2)
confusion_matrix = pd.crosstab(ytest, ypred, rownames=['Actual'],
colnames=['Predicted'])
#same as before
sn.heatmap(confusion_matrix, annot=True)
plt.title('Confusion matrix')
```