Loan Default Predicting

Adele Verron, David Such, Manille Lefort,
Elitsa Stefanova

Motivation

- Many people struggle to get loans due to insufficient or non-existent credit histories.
- We can use machine learning to broaden financial inclusion for the unbanked population by using a variety of alternative data to predict a clients' repayment abilities.
- Easy to implement as this data is readily available in loan applications.

Data

- **loan_status**: Loan status (categorical; Target variable)
- person_home_ownership: Home ownership (categorical)
- loan_intent: Loan intent (categorical)
- loan_grade: Loan grade (categorical)
- cb_person_default_on_file: Historical default (categorical)
- person_age: Age (numerical)
- person_income: Annual Income (numerical)
- person_emp_length: Employment length in years (numerical)
- loan_amnt: Loan amount (numerical)
- loan_int_rate: Loan interest rate (numerical)
- loan_percent_income: Loan amount as a percentage of income (numerical)
- cb_preson_cred_hist_length: Credit history length (numerical)

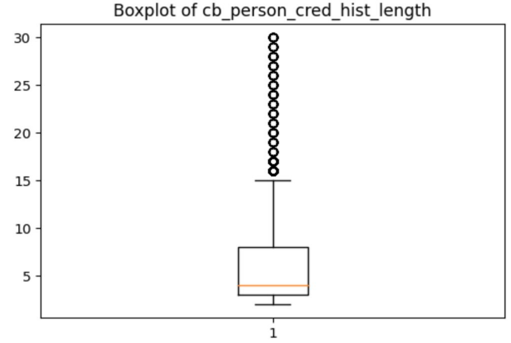
credit_risk_dataset.csv ×

1 to 10 of 32581 entries Filter

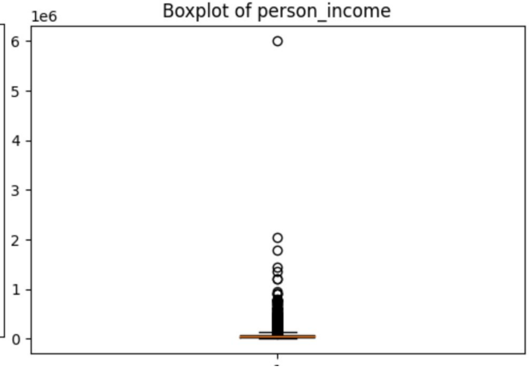
person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_default_on_file	cb_person_cre
22	59000	RENT	123.0	PERSONAL	D	35000	16.02	1	0.59	Y	3
21	9600	OWN	5.0	EDUCATION	B	1000	11.14	0	0.1	N	2
25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	1	0.57	N	3
23	65500	RENT	4.0	MEDICAL	C	35000	15.23	1	0.53	N	2
24	54400	RENT	8.0	MEDICAL	C	35000	14.27	1	0.55	Y	4
21	9900	OWN	2.0	VENTURE	A	2500	7.14	1	0.25	N	2
26	77100	RENT	8.0	EDUCATION	B	35000	12.42	1	0.45	N	3
24	78956	RENT	5.0	MEDICAL	B	35000	11.11	1	0.44	N	4
24	83000	RENT	8.0	PERSONAL	A	35000	8.9	1	0.42	N	2
21	10000	OWN	6.0	VENTURE	D	1600	14.74	1	0.16	N	3

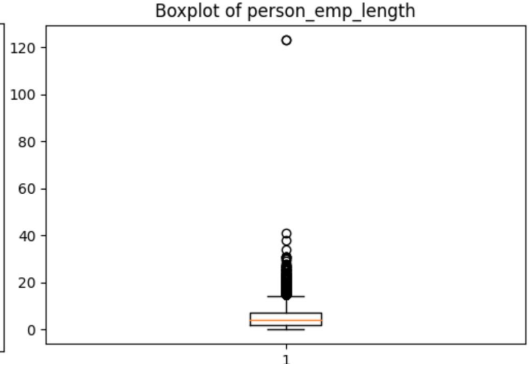
Code

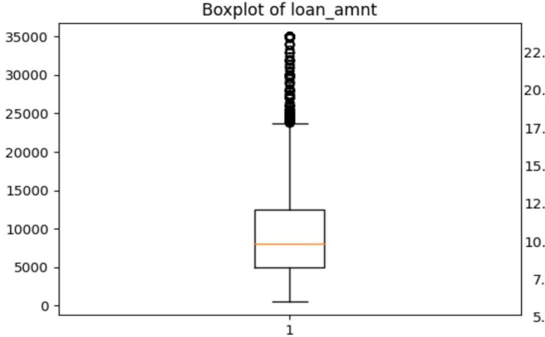
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, roc_auc_score, roc_curve, accuracy_score,
precision_score, recall_score
from sklearn.ensemble import RandomForestClassifier

data = pd.read_csv('/content/credit_risk_dataset.csv')
```

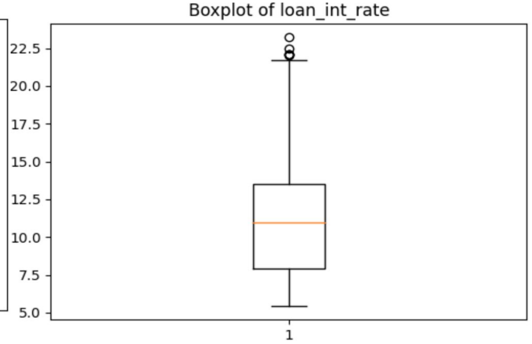


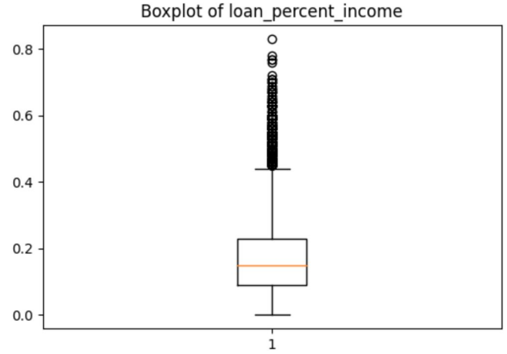
Visualizing the data distributions and removing outliers











Code

```
data = data.dropna()
# List of variables for boxplots
variables = ['person_age', 'person_income', 'person_emp_length', 'loan_amnt', 'loan_int_rate', 'loan_percent_income', 'cb_person_cred_hist_length]
# Create separate boxplots for each variable
for variable in variables:
    plt.figure(figsize=(6, 4))
    plt.boxplot(data[variable])
    plt.title(f'Boxplot of {variable}')
    plt.show()
numerical_cols = ['person_age', 'person_income', 'person_emp_length', 'loan_int_rate', 'loan_amnt', 'loan_percent_income',
'cb_person_cred_hist_length]
def outliers(data, numerical_cols):
    Q1 = data[numerical_cols].quantile(0.25)
    Q3 = data[numerical_cols].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 -1.5 * IQR
    upper_bound = Q3 +1.5 * IQR
    # Identify the outliers
    outliers_id = ((data[numerical_cols] < lower_bound) | (data[numerical_cols] > upper_bound)).any(axis=1)
    # Create a new DataFrame including only outliers
    outliers_df = data[outliers_id]
    # Create a new DataFrame excluding outliers
    data_clean = data[~outliers_id]
    return outliers_df, data_clean
# Identify and print outliers
outliers_df, data_clean = outliers(data, numerical_cols)
print("\nDataFrame after removing outliers:")
```

Cleaned and Recoded Data

	person_age	person_income	person_home_ownership	person_emp_length
1	21	9600	1	5.0
5	21	9900	1	2.0
9	21	10000	1	6.0
19	24	10800	2	8.0
23	24	10980	1	0.0

	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status
1	1	1	1000	11.14	0
5	3	3	2500	7.14	1
9	3	0	1600	14.74	1
19	1	1	1750	10.99	1
23	0	3	1500	7.29	0

	loan_percent_income	cb_person_default_on_file	cb_person_cred_hist_length
1	0.10	0	2
5	0.25	0	2
9	0.16	0	3
19	0.16	0	2
23	0.14	0	3

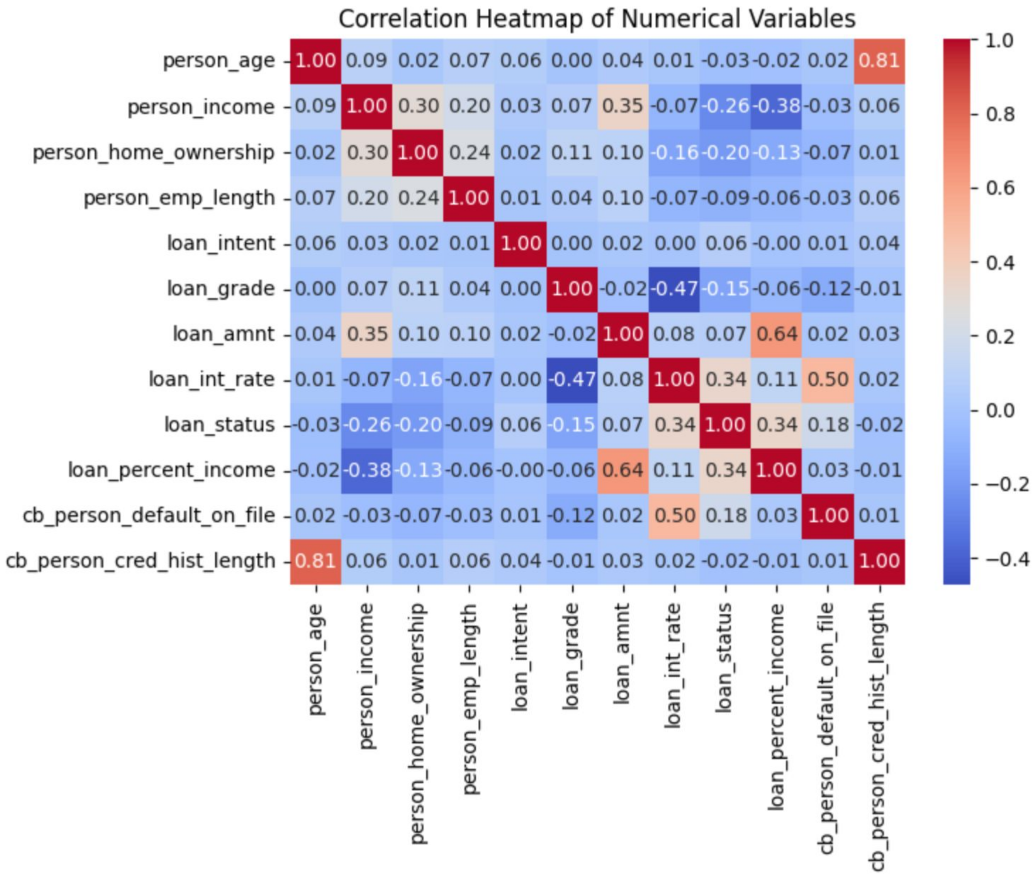
[23750 rows x 12 columns]

Code

```
# Recoding the non-numerical columns

print(data_clean['person_home_ownership'].unique())
print(data_clean['loan_intent'].unique())
print(data_clean['loan_grade'].unique())
print(data_clean['cb_person_default_on_file'].unique())

ownership_mapping = {'RENT': 0, 'OWN': 1, 'MORTGAGE': 2, 'OTHER': 3}
data_clean['person_home_ownership'].replace(ownership_mapping, inplace=True)

intent_mapping = {'PERSONAL':0, 'EDUCATION':1, 'MEDICAL':2, 'VENTURE':3, 'HOMEIMPROVEMENT':4, 'DEBTCONSOLIDATION':5}
data_clean['loan_intent'].replace(intent_mapping, inplace=True)

grade_mapping = {'D':0, 'B':1, 'C':2, 'A':3, 'E':4, 'F':5, 'G':6}
data_clean['loan_grade'].replace(grade_mapping, inplace=True)

default_mapping = {'Y':1, 'N':0}
data_clean['cb_person_default_on_file'].replace(default_mapping, inplace=True)
```

Heatmap

- Loan as an amount and as a percentage of income are highly correlated, as well as a person's age and credit history length.
- We choose only one of these pairs of variables proceeding with our analysis.

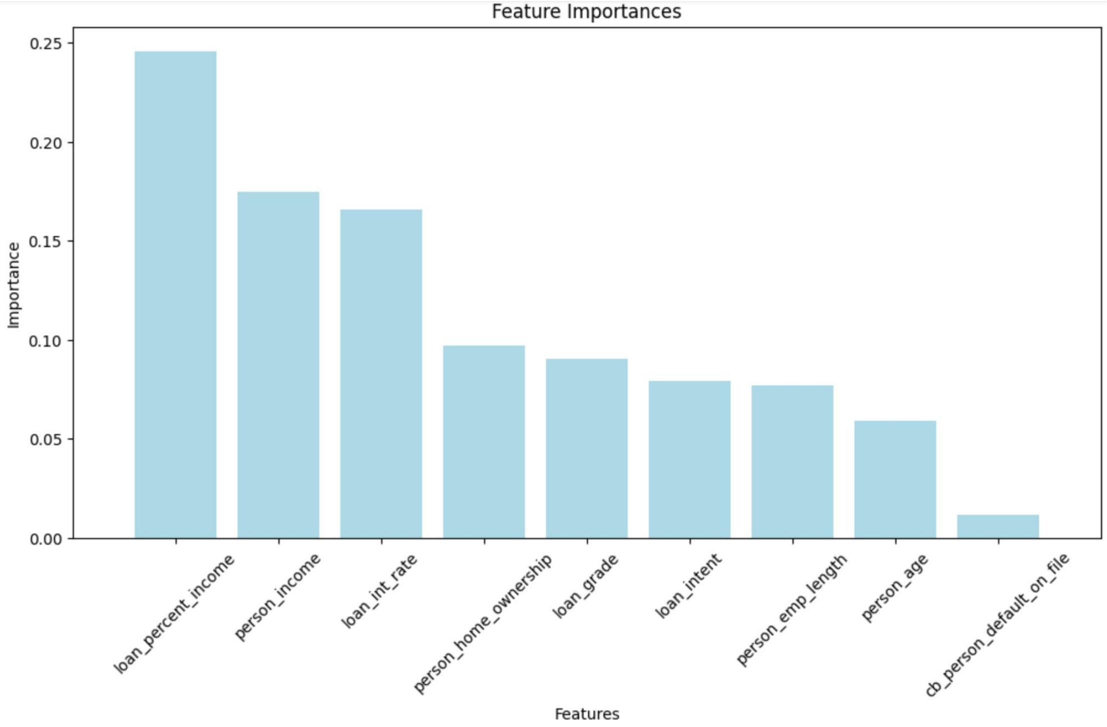


Code

```
# Create a heatmap with numerical variables
numerical_df = data_clean.select_dtypes (include='number')
corr_matrix = numerical_df.corr ()
plt.figure(figsize=(7, 5))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numerical Variables' )
plt.show()

data_clean = data_clean.drop (['loan_amnt', 'cb_person_cred_hist_length' ], axis=1)
```

Random Forest Feature Selection

- Based on this graph we can make a model excluding the least relevant variables: default history, history length, age, employment length, and intent that would only bring more noise to the model.



Code

```
# Chosing variables
X = data_clean.drop('loan_status', axis=1)
Y = data_clean['loan_status']
# Feature Selection using Random Forest
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X, Y)
# Getting feature importances
importances = rf.feature_importances_
# Sorting the feature importances in descending order
indices = np.argsort(importances)[::-1]
# Displaying the feature importances
print("Feature ranking:")
for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
# Plotting feature importances
plt.figure(figsize=(12, 6))
plt.title("Feature Importances")
plt.bar(range(X.shape[1]), importances[indices], color="lightblue", align="center")
plt.xticks(range(X.shape[1]), X.columns[indices], rotation=45)
plt.xlim([-1, X.shape[1]])
plt.ylabel('Importance')
plt.xlabel('Features')
plt.show()
```

Chi-Squared Feature Selection

- Under this method, the most important features are: person_home_ownership , person_income, loan_int_rate

```
Selected Features:
[[9.600e+03 1.114e+01]
 [9.900e+03 7.140e+00]
 [1.000e+04 1.474e+01]
 [1.080e+04 1.099e+01]
 [1.098e+04 7.290e+00]]
Feature Scores:
[1.01077894e+01 1.95190752e+07 9.77589884e+02 5.36140359e+02
 1.15135413e+02 3.60111518e+02 2.50230539e+03 1.45155524e+02
 6.44883224e+02]
```

Code

```
# Feature Selection using SelectKBest and Chi-Squared Test to check whether the our previous findings
are robust
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# Selecting the top 2 features
selector = SelectKBest (score_func=chi2 , k=2)
X_selected = selector.fit_transform (X, Y)
# Displaying the selected features
print ("Selected Features:" )
print (X_selected [:5, :])

# The scores for each feature
print ("Feature Scores:" )
print (selector.scores_ )
```

CLASSIFICATION METHODS AND EVALUATION METRICS

- KNN
- Logistic Regression
- Decision Tree
- Random Forest

- Accuracy
- Confusion Matrix
  - Precision, Recall
- F1 Score
- ROC Curve

Training and Testing set

Chi-Squared Feature Selection

- person_home_ownership , person_income, loan_int_rate

Random Forest Feature Selection

- loan_percent_income, person_income, loan_int_rate

TARGET VARIABLE

- loan_status
  - Non default (0)
  - Default (1)

X [person_home_ownership, person_income, loan_int_rate, loan_percent_income]

X_train (80%)
X_test (20%)

Y [loan_status]

Y_train (80%)
Y_test (20%)

EVALUATION OF THE MODELS

Actual Predicted	Negative (0)	Positive (1)
Negative (0)	TN	FP
Positive (1)	FN	TP

ACCURACY

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions made}}$$

RECALL

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

PRECISION

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

F1 SCORE

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Precision is more important in our case because we want to make sure that nobody will default.

The harmonic mean of precision and recall taking both false positives and false negatives into account

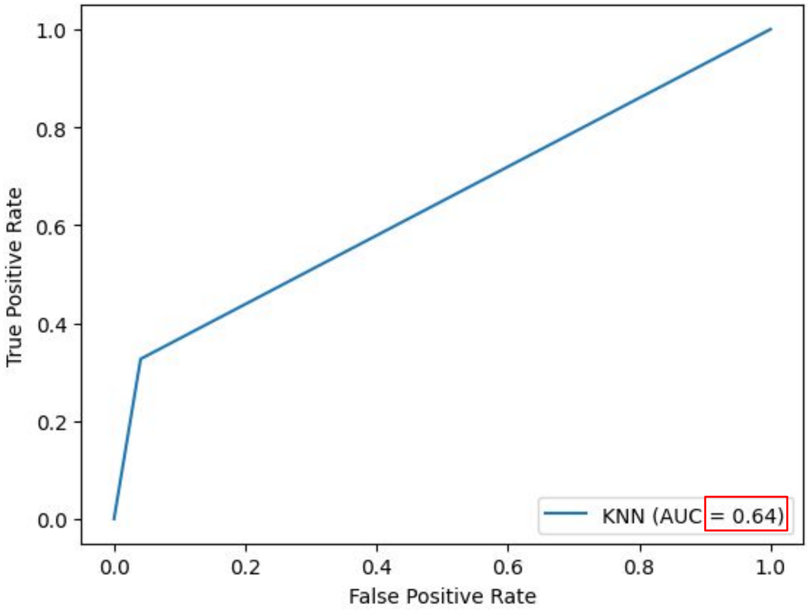
KNN method

Algorithm used for classification and regression.
KNN to classify individuals into two segments: ‘No loan default’ and ‘Loan default’ based on their annual income, home ownership, percent income, and interest rate.

STEPS

- Find the best k (here 2),
- Predict Y using X_test values,
- **Evaluate the model**
    - Accuracy under k=2 to see if the predictions match the test sample
    - Confusion matrix
    - Classification report F1
    - Roc Curve

```
Accuracy: 0.8339
Confusion Matrix:
 [[3653  154]
 [ 635  308]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      3807
           1       0.67      0.33      0.44       943

    accuracy                           0.83      4750
   macro avg       0.76      0.64      0.67      4750
weighted avg       0.82      0.83      0.81      4750
```



Code

```
# From sklearn.model_selection import
train_test_split
from sklearn.neighbors import
KNeighborsClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import
StandardScaler
from sklearn.linear_model import
LogisticRegression
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix,
RocCurveDisplay
#Create the variable X and Y
X = data_clean[['person_income',
'loan_int_rate', 'loan_percent_income',
'person_home_ownership']]
Y = data_clean['loan_status']
# Splitting data into training and testing sets
X_train, X_test, Y_train, Y_test =
train_test_split(X, Y, test_size=0.2,
```

```
# Using KNN for classification with k=2
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
# Evaluate the model KNN
accuracy = accuracy_score(Y_test, Y_pred)
conf_matrix = confusion_matrix(Y_test, Y_pred)
classification_rep1 = classification_report(Y_test, Y_pred)
print(f"Accuracy: {accuracy:.4f}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep1)
#ROC Curve for KNN
fpr, tpr, tresholds = metrics.roc_curve(Y_test, Y_pred)
roc_auc = metrics.auc(fpr,tpr)
display = metrics.RocCurveDisplay (fpr=fpr, tpr=tpr, roc_auc=roc_auc,
estimator_name = 'KNN')
display.plot()
plt.show()
```

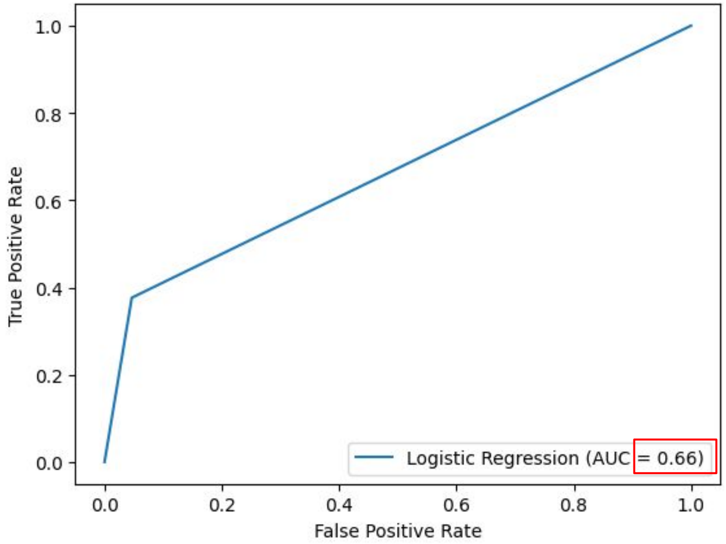
Logistic Regression method

Regression analysis that is suited for binary classification problems
Measures the relationship between Y and X's by estimating probabilities using a logistic function.

STEPS

- Use (X_train, Y_train) to fit the logistic regression model
    - finding the coefficients that minimize a loss function.
- Predict the model using X_test values,
- **Evaluate the model**
    - Accuracy
    - Confusion matrix
    - Classification report F1
    - Roc Curve

```
Logistic Regression:
Accuracy: 0.8389
Confusion Matrix:
 [[3630  177]
 [ 588  355]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.95      0.90      3807
           1       0.67      0.38      0.48       943

    accuracy                           0.84      4750
   macro avg       0.76      0.66      0.69      4750
weighted avg       0.82      0.84      0.82      4750
```



Code

```
#Logistic Regression
#Standardize features (important for logistic
regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Logistic Regression
logreg = LogisticRegression(random_state=0)
logreg.fit(X_train_scaled, Y_train)
# Make predictions
logreg_predictions =
logreg.predict(X_test_scaled)
```

```
# Evaluate the model Logistic Regression
accuracy = accuracy_score(Y_test, logreg_predictions)
conf_matrix = confusion_matrix(Y_test,
logreg_predictions)
classification_rep2 = classification_report(Y_test,
logreg_predictions)
print("Logistic Regression:")
print(f"Accuracy: {accuracy:.4f}" )
print("Confusion Matrix:\n" , conf_matrix)
print("Classification Report:\n" , classification_rep2)
#ROC Curve for Logistic regression
fpr, tpr, tresholds = metrics.roc_curve(Y_test,
logreg_predictions)
roc_auc = metrics.auc(fpr,tpr)  #displays the area under
the curve
display = metrics.RocCurveDisplay (fpr=fpr, tpr=tpr,
roc_auc=roc_auc, estimator_name =  'Logistic Regression' )
display.plot()
plt.show()
```

Decision Tree method

Predicts the value of a target variable by learning decision rules inferred from data features.

- **Node** represents a feature (or attribute),
- **Link** represents a decision/rule
- **Leaf** represents an outcome

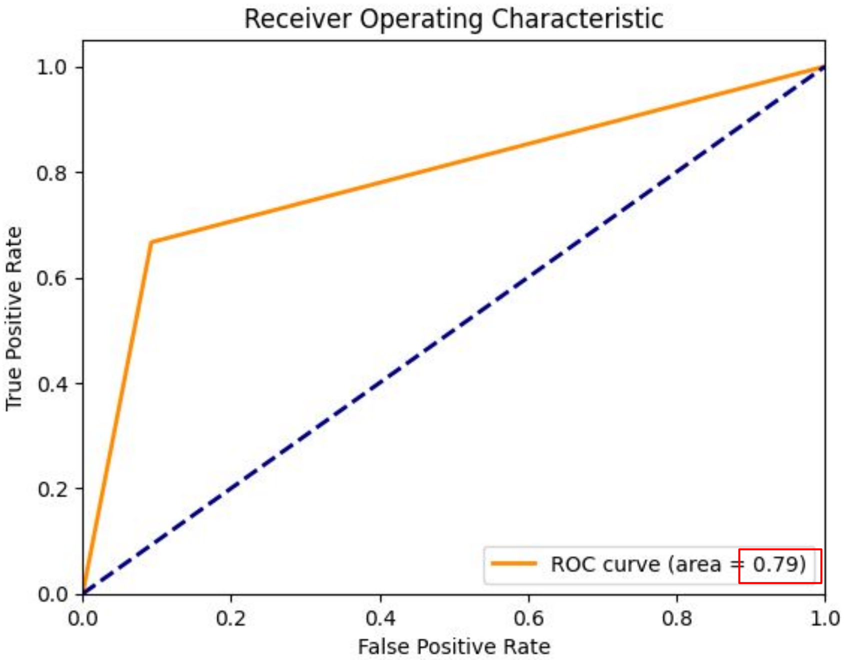
Path from the root to the leaf represent classification rules.

STEPS:

- We initialize the decision tree classifier using the train data.
- We use the X_test data to make predictions and print the report.
- We evaluate the model by looking at precision, accuracy and ROC curve.

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.91	0.91	5699
1	0.64	0.67	0.65	1426
accuracy			0.86	7125
macro avg	0.78	0.79	0.78	7125
weighted avg	0.86	0.86	0.86	7125



Code

```
# #For Decision Trees
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report,
roc_curve, auc
import matplotlib.pyplot as plt

# Split the dataset into training and test sets
X_train, X_test, Y_train, Y_test =
train_test_split(X, Y, test_size=0.3,
random_state=42)

# Initialize and train the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, Y_train)

# Make predictions and evaluate the model
y_pred1 = clf.predict(X_test)
report = classification_report(Y_test, y_pred1)
print('Classification Report:\n', report)
```

```
# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(Y_test,
clf.predict_proba(X_test)[:,1], pos_label=1)
roc_auc = auc(fpr, tpr)

# Plotting the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2,
label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2,
linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```

Random Forest method

It combines multiple decision trees to improve the predictive performance (F1,Accuracy, AUC-ROC curve) and control over-fitting.

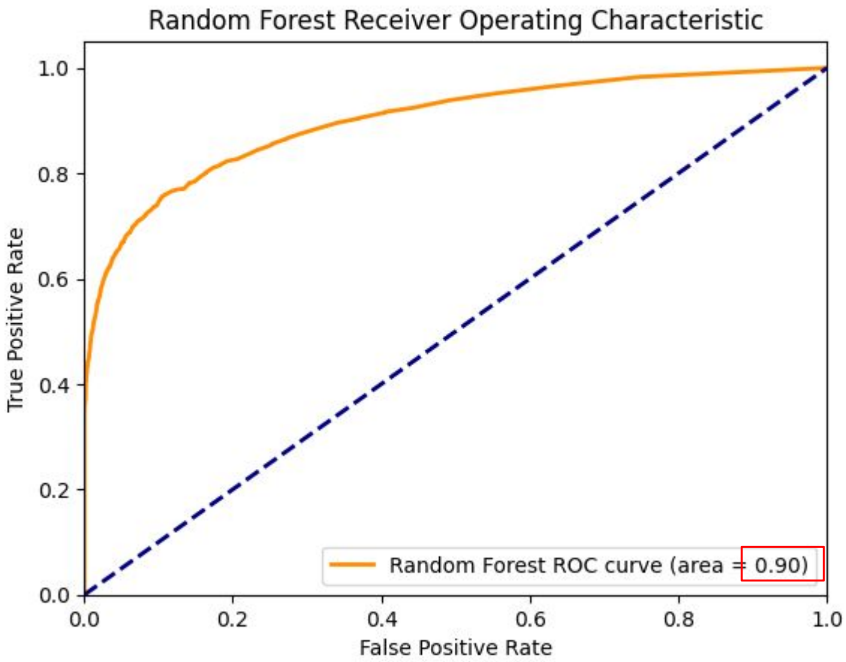
It does not work well for datasets having a lot of outliers, something which needs addressing previously.

STEPS:

- We initialize the random forest classifier using the train data.
- We use the X_test data to make predictions and print the report.
- We evaluate the model by looking at precision, accuracy and ROC curve.

Random Forest Classification Report:

	precision	recall	f1-score	support
0	0.91	0.96	0.94	5699
1	0.82	0.63	0.71	1426
accuracy			0.90	7125
macro avg	0.86	0.80	0.82	7125
weighted avg	0.89	0.90	0.89	7125



Code

```
# For andom Forest
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the Random Forest classifier
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, Y_train)

# Make predictions and evaluate the model
y_pred_rf = rf_clf.predict(X_test)
rf_report = classification_report(Y_test, y_pred_rf)
print('Random Forest Classification Report:\n' ,
rf_report)

# Calculate ROC curve and AUC
rf_fpr, rf_tpr, rf_thresholds = roc_curve(Y_test,
rf_clf.predict_proba(X_test)[:,1], pos_label=1)
rf_roc_auc = auc(rf_fpr, rf_tpr)
```

```
# Plotting the ROC curve
plt.figure()
plt.plot(rf_fpr, rf_tpr, color= 'darkorange' , lw=2,
label='Random Forest ROC curve (area = %0.2f)'  %
rf_roc_auc)
plt.plot([0, 1], [0, 1], color= 'navy', lw=2,
linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel( 'False Positive Rate' )
plt.ylabel( 'True Positive Rate' )
plt.title( 'Random Forest Receiver Operating
Characteristic' )
plt.legend(loc= 'lower right' )
plt.show()
```

Comparative Table

	Accuracy	Precision	Recall	F1 Score	AUC ROC
KNN	0.834526	0.673699	0.335905	0.448292	0.647598
Logistic	0.840421	0.673887	0.392707	0.496234	0.672577
Decision Tree	0.859088	0.642954	0.665498	0.654032	0.786513
Random Forest	0.896842	0.815525	0.626227	0.708449	~~0.795391~~ 0.9

Code

```
#Create a comparative table
import pandas as pd
from sklearn.metrics import accuracy_score,
precision_recall_fscore_support, roc_auc_score

# Assuming you have multiple classifiers and their
predictions
classifiers = ['KNN', 'Logistic', 'Decision Tree', 'Random
Forest']
predictions = [Y_pred, logreg_predictions, y_pred1,
y_pred_rf]  # Replace with your actual predictions

# Create a DataFrame to store the results
results_df = pd.DataFrame(index=classifiers,
columns=['Accuracy', 'Precision', 'Recall', 'F1 Score', 'AUC
ROC'])

# Fill in the DataFrame with metrics for each classifier
for i, classifier in enumerate(classifiers):
    y_true = Y_test
    y_pred = predictions[i]

    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision, recall, f1_score, _ =
precision_recall_fscore_support(y_true, y_pred,
average='binary')
    auc_roc = roc_auc_score(y_true, y_pred)

    # Fill in the DataFrame
    results_df.loc[classifier] = [accuracy,
precision, recall, f1_score, auc_roc]

# Display the comparative table
print(results_df)
```

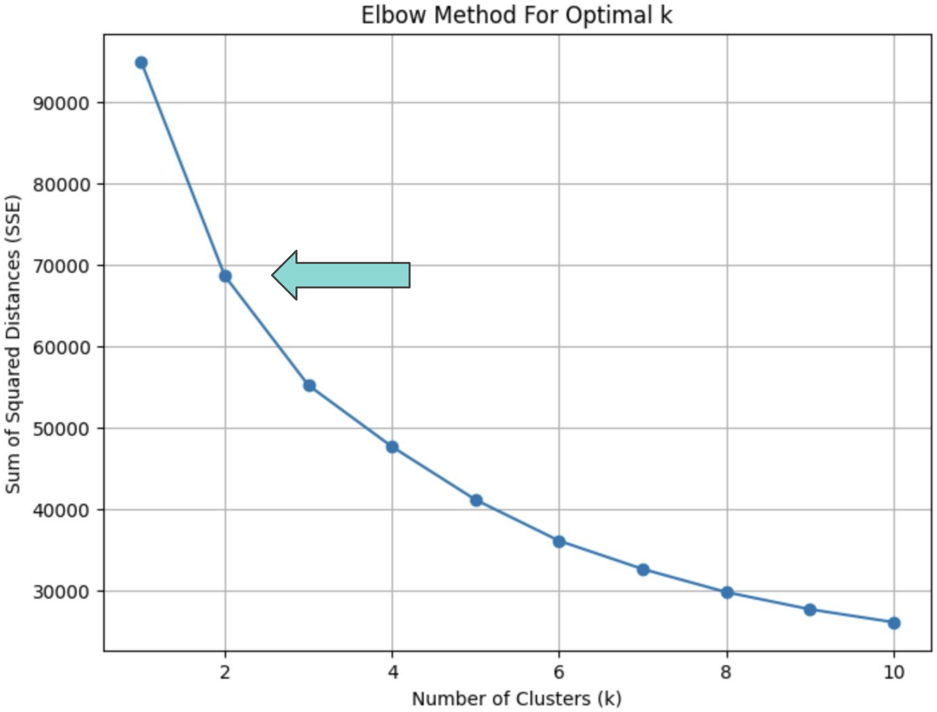
Elbow Method to get optimal number of clusters

The elbow method consists of three steps:

1. Run k-means several times
2. Increment k with each iteration
3. Record the SSE

On the right side of this slide is displayed the graph of the Elbow method (SSE: measure of error against Number of clusters).

We chose k = 2 as from this point the curve starts to bend.



Code

```
kmeans_kwargs = {  "init": "random", "n_init": 10, "max_iter": 300,
"random_state": 42,}

sse = []
for k in range (1,11):
 kmeans = KMeans (n_clusters=k, **kmeans_kwargs)
 kmeans.fit(X)
 sse.append(kmeans.inertia_)

import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), sse, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Sum of Squared Distances (SSE)')
plt.grid(True)
plt.show()
```

Fit k-means with k=2

```
▼ KMeans
KMeans(init='random', n_clusters=2, n_init=10, random_state=42)
```

Using the Elbow method we decided to set 2 clusters.

Clusters are: groups of data objects that are more similar to other objects in their cluster than they are to data objects.

The K-means (*unsupervised*) algorithm randomly initializes centroids and then repeats the ‘expectation-maximization’ process until the centroid positions do not change.
The ‘expectation-maximization’ process consists of assigning each point to the closest centroid and then to compute the new centroid mean of each cluster.

K-means does ‘Partitional Clustering’ meaning that no object can be a member of more than one cluster and every cluster must have at least one object.

Code

```
from sklearn.cluster import KMeans
kmeans= KMeans(init="random", n_clusters=2,n_init=10,max_iter=300,random_state=42)
kmeans.fit(X)
```

In conclusion

- Model picked: Random Forest
  - 89% accuracy
  - 90% AUC
- We can predict default quite well and relatively easily with the data at hand and this model
- As a last step, using the Elbow Method to choose we set k to 2 and clustered our data set using the k-means algorithm

Thank You

Questions are welcome!