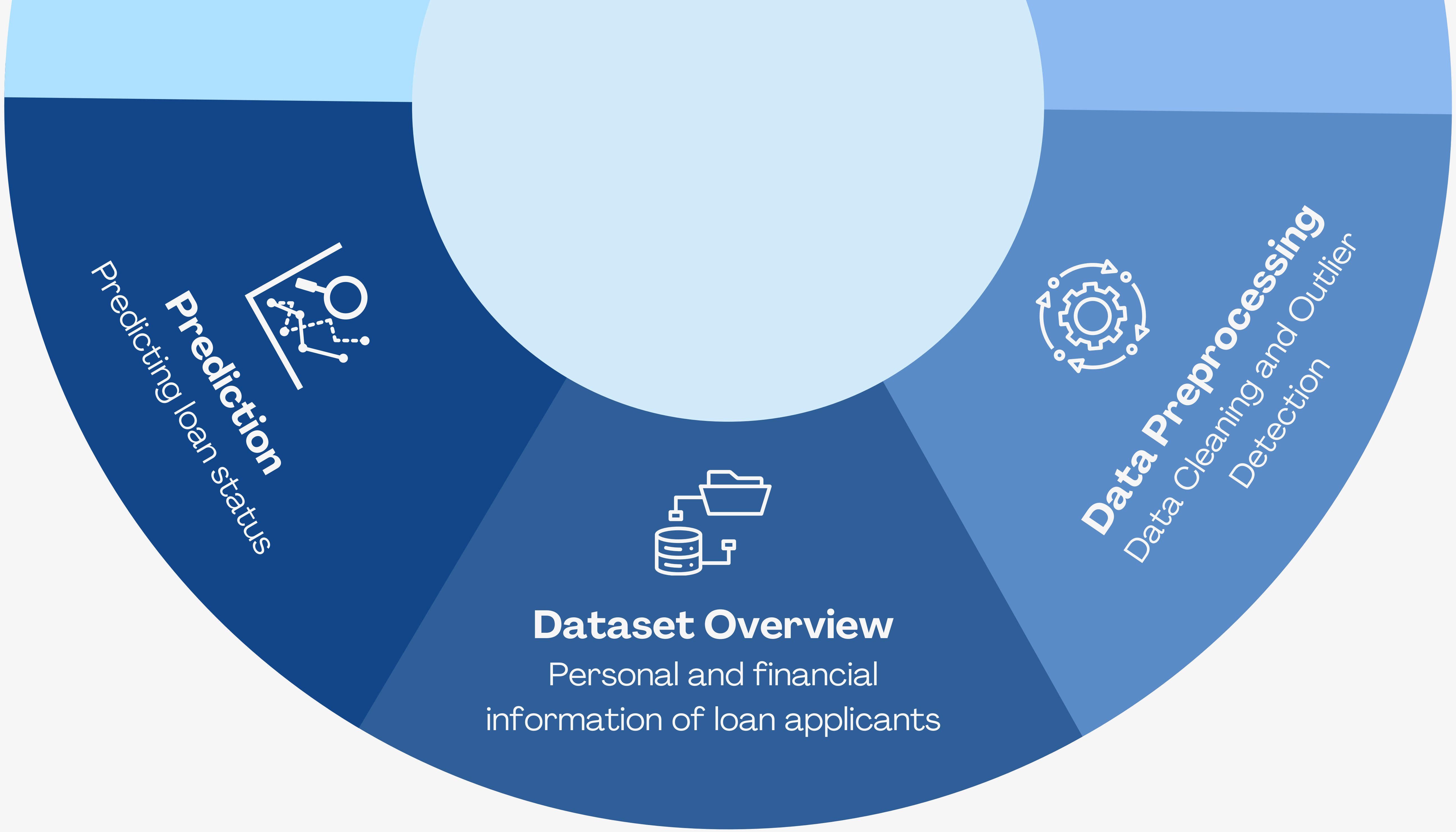
Data Mining Project

BANK LOAN APPROVAL

Student: Phan Thuy Anh, Le Quynh Chi
Class: Business Analytics 62



Bank Loan Approval

Dataset Overview

Kaggle, two sets: a training dataset and a scoring dataset, both of which are designed to facilitate loan prediction tasks.

Dataset Attribute

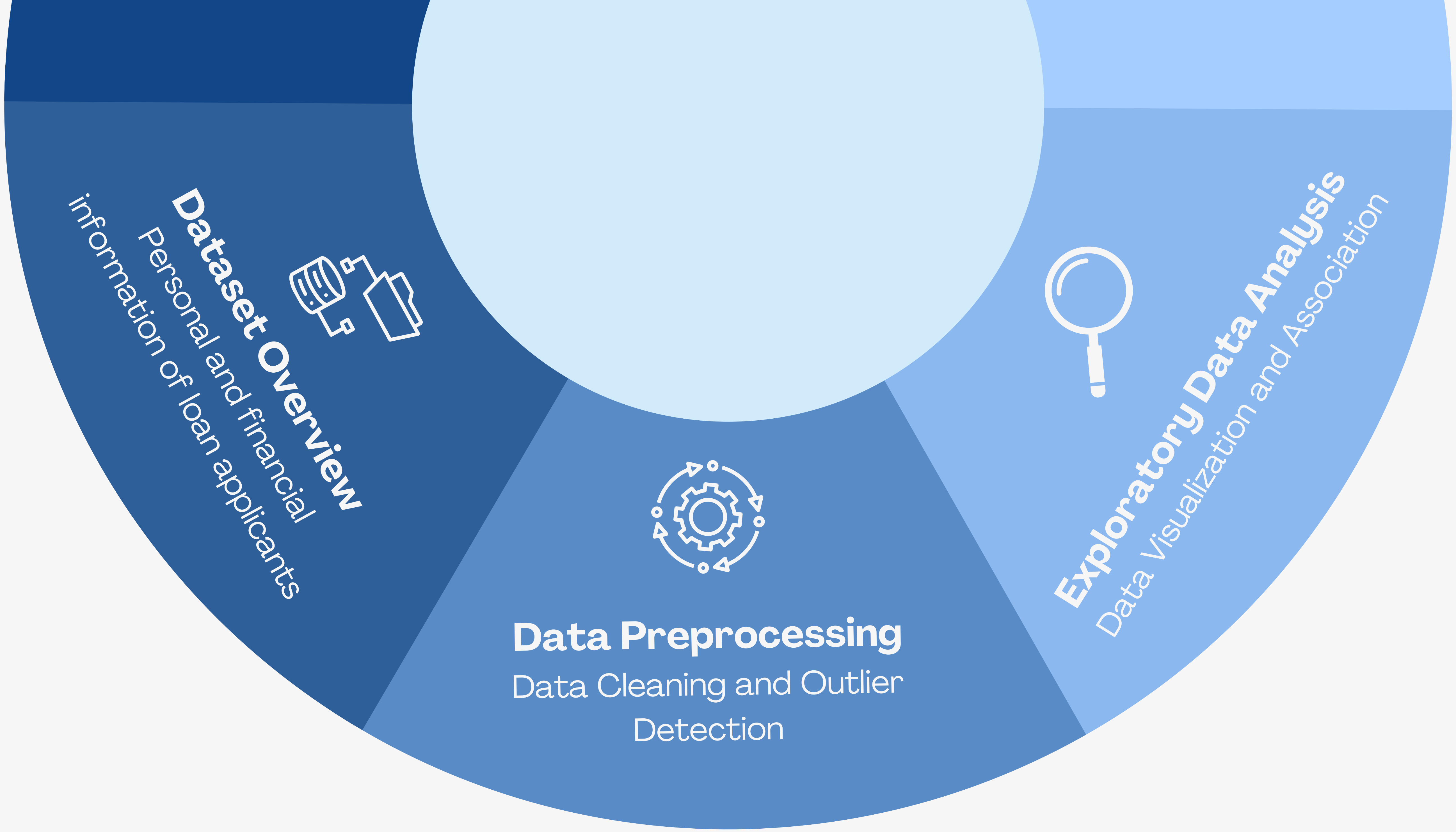
Purpose	The purpose of the loan application.
Monthly Debt	The applicant's monthly debt payments.
Years of Credit History	The applicant's number of years of credit history.
Months since Last Delinquent	The number of months since the applicant's last delinquent payment.
Number of Open Accounts	The number of open credit accounts the applicant has.
Number of Credit Problems	The number of credit problems the applicant has had.
Current Credit Balance	The current balance of the applicant's credit accounts.
Maximum Open Credit	The maximum open credit limit for the applicant.
Bankruptcies	The number of bankruptcies on the applicant's credit record.
Tax Liens	The number of tax liens on the applicant's record.

Training Dataset

Missing data	• 'Credit Score' and 'Annual Income' each had 19,149 missing values (19.2% of the total values) • 'Years in the current job' lacked 4,222 entries (4.2% of the total values) • 'Months since last delinquent' had 53,140 missing values (53.1% of the total values)
Data types	• Integers, floats, and objects.
Duplicate rows	• 10214 duplicates rows
Data formatting checks	• Although not immediately relevant to this dataset, potential formatting inconsistencies such as date formats and special characters were considered.

Testing Dataset

Missing data	• Credit Score' and 'Annual Income' each have 1,981 missing values (19.8% of the total values). • 'Years in current job' lacks 427 entries (4.3% of the total values). • 'Months since last delinquent' has 5,306 missing values (53.1% of the total values).
Data types	• Integers, floats, and objects.
Duplicate rows	• 0 duplicates rows
Data formatting checks	• Although not immediately relevant to this dataset, potential formatting inconsistencies such as date formats and special characters were considered.



Bank Loan Approval

Data Preprocessing

Ensuring the quality and reliability of the dataset for analysis.

Check missing values

```
missing_values = training.isnull().sum()
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
Loan ID                          514
Customer ID                      514
Loan Status                      514
Current Loan Amount              514
Term                             514
Credit Score                   19668
Annual Income                  19668
Years in current job            4736
Home Ownership                   514
Purpose                          514
Monthly Debt                     514
Years of Credit History          514
Months since last delinquent   53655
Number of Open Accounts          514
Number of Credit Problems        514
Current Credit Balance           514
Maximum Open Credit              516
Bankruptcies                     718
Tax Liens                        524
dtype: int64
```

Check proportion of missing values

```
def missing_values_table(training):
      mis_val = training.isnull().sum()
      mis_val_percent = 100*training.isnull().sum() / len(training)
      mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
      mis_val_table_ren_columns = mis_val_table.rename(columns = {0:'Missing Values' , 1:'% of Total Values'})
       return mis_val_table_ren_columns.round(1)
```

```
missing_values_table(training)
```

Loan ID	0	0.0
Customer ID	0	0.0
Loan Status	0	0.0
Current Loan Amount	0	0.0
Term	0	0.0
Credit Score	19149	19.2
Annual Income	19149	19.2
Years in current job	4222	4.2
Home Ownership	0	0.0
Purpose	0	0.0
Monthly Debt	0	0.0
Years of Credit History	0	0.0
Months since last delinquent	53140	53.1
Number of Open Accounts	0	0.0
Number of Credit Problems	0	0.0
Current Credit Balance	0	0.0
Maximum Open Credit	0	0.0
Bankruptcies	194	0.2

Drop the null values

```
columns_to_drop_missingvalues = ['Loan ID', 'Customer ID', 'Loan Status', 'Current Loan Amount', 'Term', 'Home Ownership', 'Purpose', 'Monthly Debt', 'Years of Credit History', 'Number of Open Accounts', 'Number of Credit Problems', 'Current Credit Balance', 'Maximum Open Credit','Tax Liens']
```

```
training.dropna(subset=columns_to_drop_missingvalues , inplace=True)
```

Drop the necessary columns

```
training.drop(['Customer ID' ,'Loan ID'] , axis=1 , inplace=True)
```

Dealing with null values: Credit Score

```
import matplotlib.pyplot as plt
credit = training['Credit Score']
credit.plot.hist(bins=20,alpha=0.5)
```

```
plt.xlabel('Credit Score')
plt.title('Distribution of Credit Score')
plt.show()
```

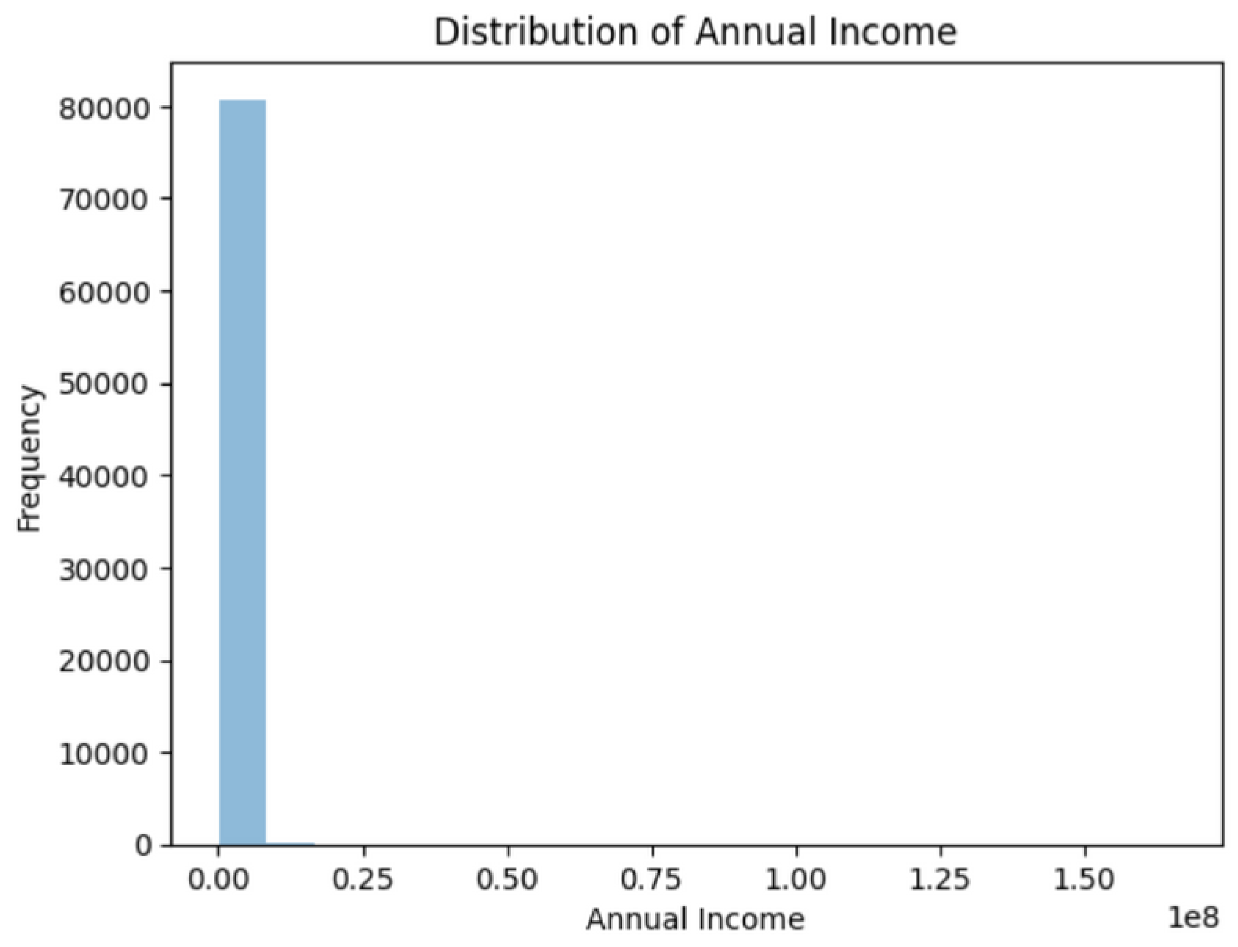
A positive skewed distribution: use the Mode strategy

```
from sklearn.impute import SimpleImputer
mode_imputer = SimpleImputer(strategy='most_frequent')
column_impute = 'Credit Score'
imputed_column = mode_imputer.fit_transform(training[[column_impute]])
training[column_impute] = imputed_column
```

Dealing with null values: Annual Income

```
import matplotlib.pyplot as plt
credit = training['Annual Income']
credit.plot.hist(bins=20,alpha=0.5)
```

```
plt.xlabel('Annual Income')
plt.title('Distribution of Annual Income')
plt.show()
```



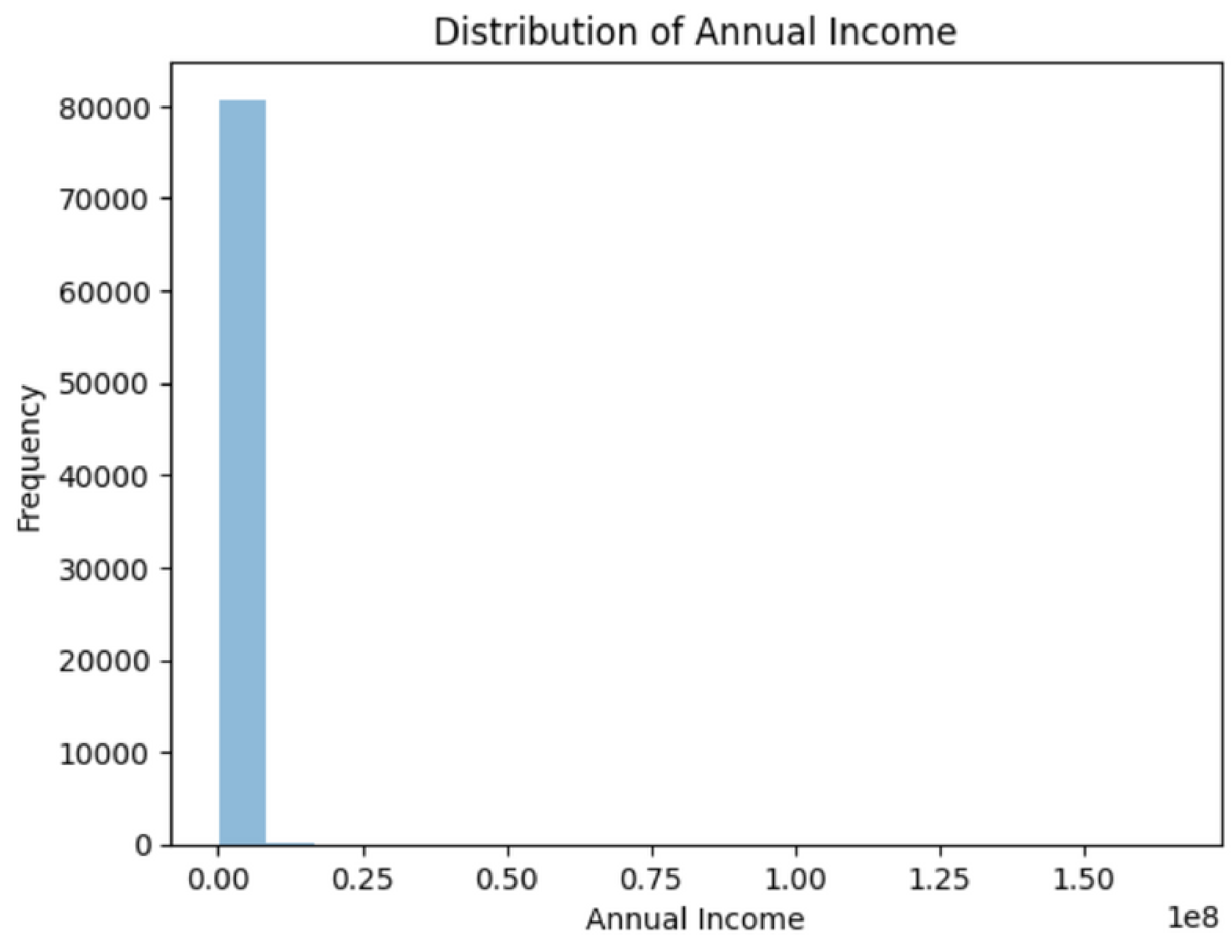
A positive skewed distribution: use the Mode strategy

```
from sklearn.impute import SimpleImputer
mode_imputer = SimpleImputer(strategy='most_frequent')
annual_income = 'Annual Income'
imputed_annual = mode_imputer.fit_transform(training[[annual_income]])
training[annual_income] = imputed_annual
```

Dealing with null values: Bankruptcies

```
import matplotlib.pyplot as plt
credit = training['Bankruptcies']
credit.plot.hist(bins=20,alpha=0.5)
```

```
plt.xlabel('Bankruptcies')
plt.title('Distribution of 'Bankruptcies')
plt.show()
```



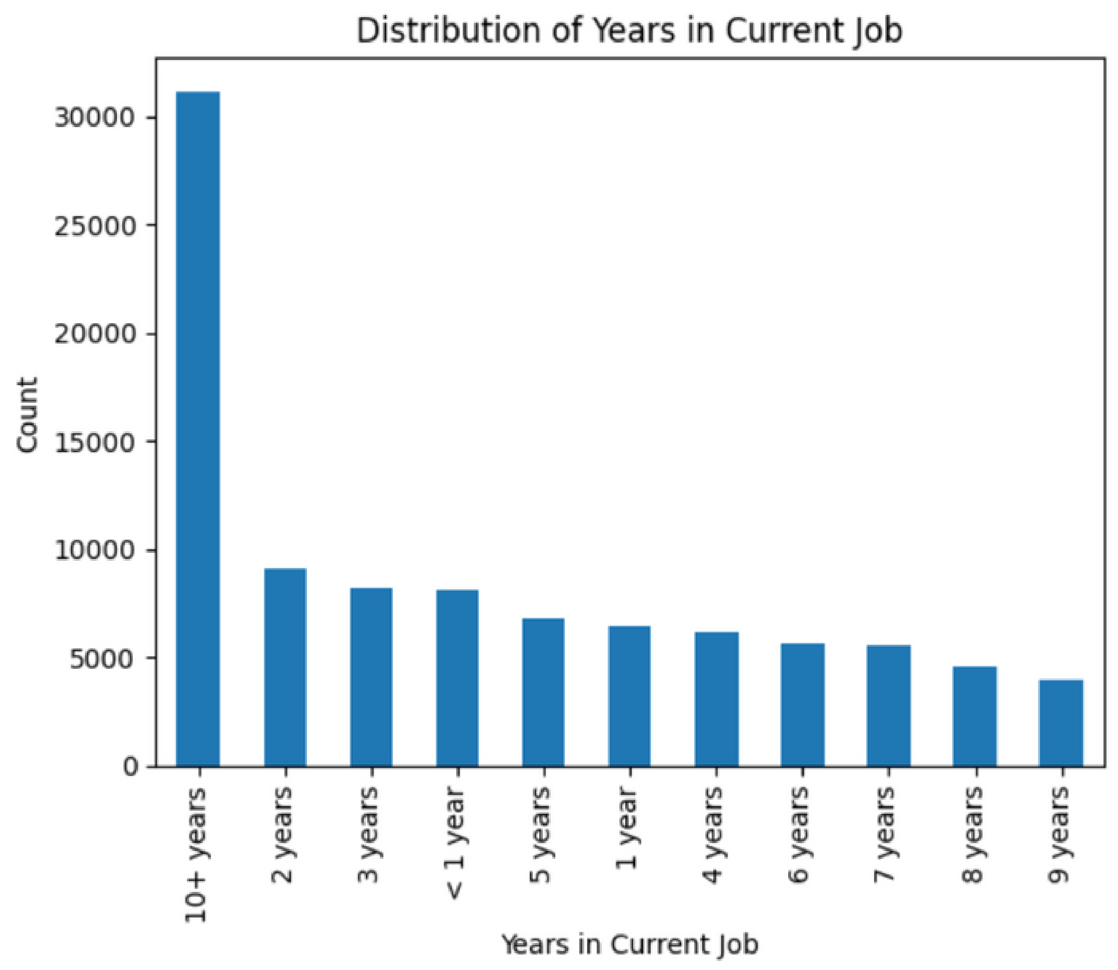
A positive skewed distribution: use the Mode strategy

```
from sklearn.impute import SimpleImputer
mode_imputer = SimpleImputer(strategy='most_frequent')
bankruptcies = 'Bankruptcies'
imputed_bankruptcies = mode_imputer.fit_transform(training[[bankruptcies]])
training[bankruptcies] = imputed_bankruptcies
```

Dealing with null values: Years in current job

```
import matplotlib.pyplot as plt
training['Years in current job'].value_counts().plot(kind='bar')
```

```
plt.xlabel('Years in Current Job')
plt.ylabel('Count')
plt.title('Distribution of Years in Current Job')
plt.show()
```



A positive skewed distribution: use the Median strategy

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
training['Years in current job'] = le.fit_transform(training['Years in current job'])
median_imputer = SimpleImputer(strategy='median')
imputed_year = median_imputer.fit_transform(training[['Years in current job']])
training['Years in current job'] = imputed_year
```

Check missing values again

```
training.drop(columns='Months since last delinquent', axis=1, inplace=True)
missing_values_table(training)
```

	Missing Values	% of Total Values
Loan ID	0	0.0
Customer ID	0	0.0
Loan Status	0	0.0
Current Loan Amount	0	0.0
Term	0	0.0
Credit Score	0	0.0
Annual Income	0	0.0
Years in current job	0	0.0
Home Ownership	0	0.0
Purpose	0	0.0
Monthly Debt	0	0.0
Years of Credit History	0	0.0
Number of Open Accounts	0	0.0
Number of Credit Problems	0	0.0
Current Credit Balance	0	0.0
Maximum Open Credit	0	0.0
Bankruptcies	0	0.0
Tax Liens	0	0.0

Check for duplicates

```
duplicate_rows = training.duplicated().sum()
print("\nDuplicate Rows:")
print(duplicate_rows)
```

```
Duplicate Rows:
10214
```

Remove duplicates

```
training.drop_duplicates(inplace=True)
```

Encode categorical attributes

```
training['Loan Status'] = le.fit_transform(training['Loan Status'])
training['Term'] = le.fit_transform(training['Term'])
training['Home Ownership'] = le.fit_transform(training['Home Ownership'])
training['Purpose'] = le.fit_transform(training['Purpose'])
```

Verify unique values for categorical attributes

```
categorical_columns = training.select_dtypes(include='object').columns
for column in categorical_columns:
    unique_values = training[column].nunique()
    print(f"\nUnique values in {column}: {unique_values}")
```

```
Unique values in Loan ID: 81999

Unique values in Customer ID: 81999

Unique values in Loan Status: 2

Unique values in Term: 2

Unique values in Years in current job: 11

Unique values in Home Ownership: 4

Unique values in Purpose: 16
```

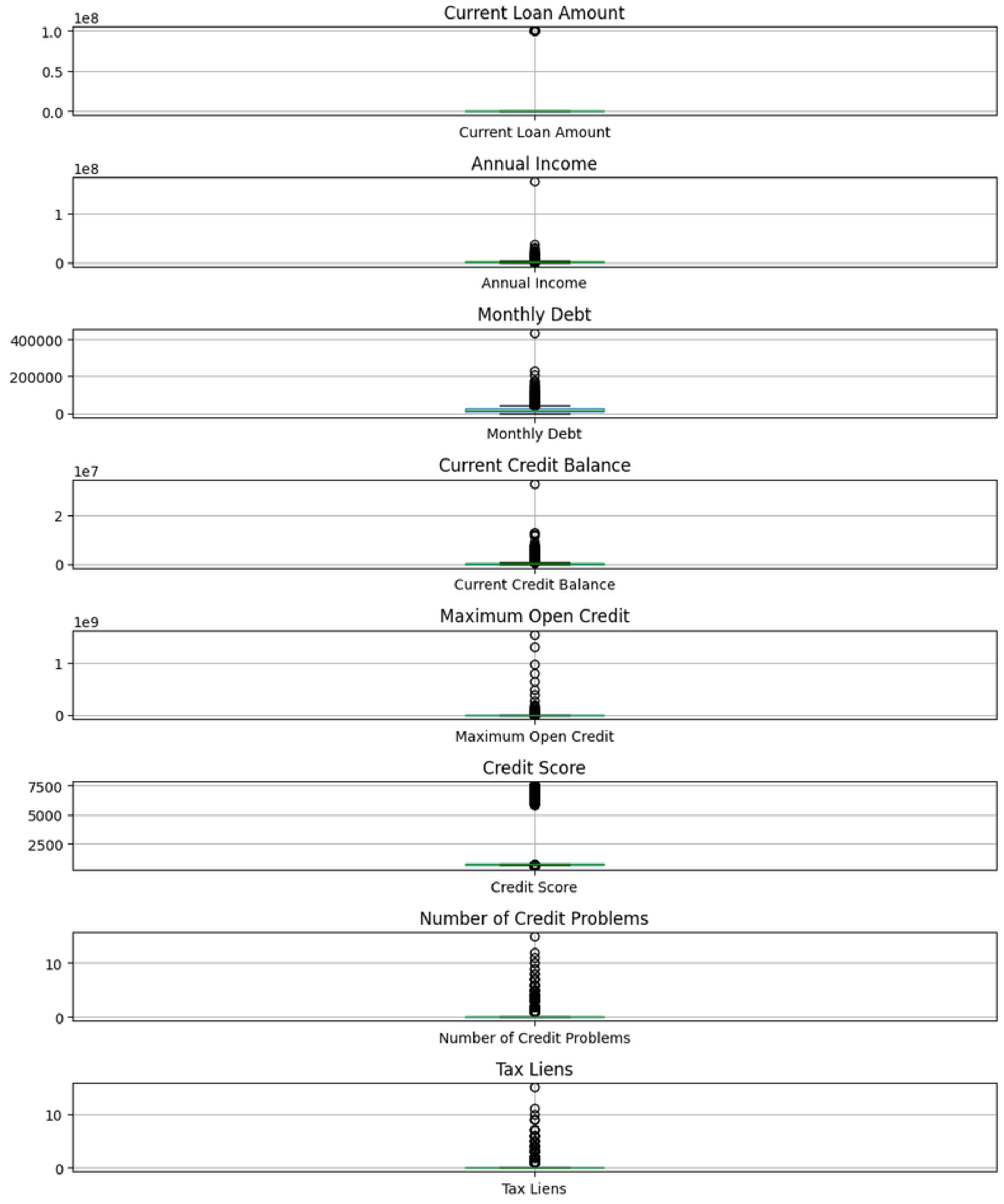
Outlier detection

```python
attributes = ['Current Loan Amount', 'Annual Income', 'Monthly Debt', 'Current
Credit Balance', 'Maximum Open Credit', 'Credit Score', 'Number of Credit
Problems', 'Tax Liens']
```

```python
#Create box plots for each attribute
fig, axes = plt.subplots(nrows=len(attributes), ncols=1, figsize=(10, 12))
fig.subplots_adjust(hspace=0.5)
```

```python
for i, attribute in enumerate(attributes):
    training.boxplot(column=attribute, ax=axes[i])
    axes[i].set_title(attribute)
```

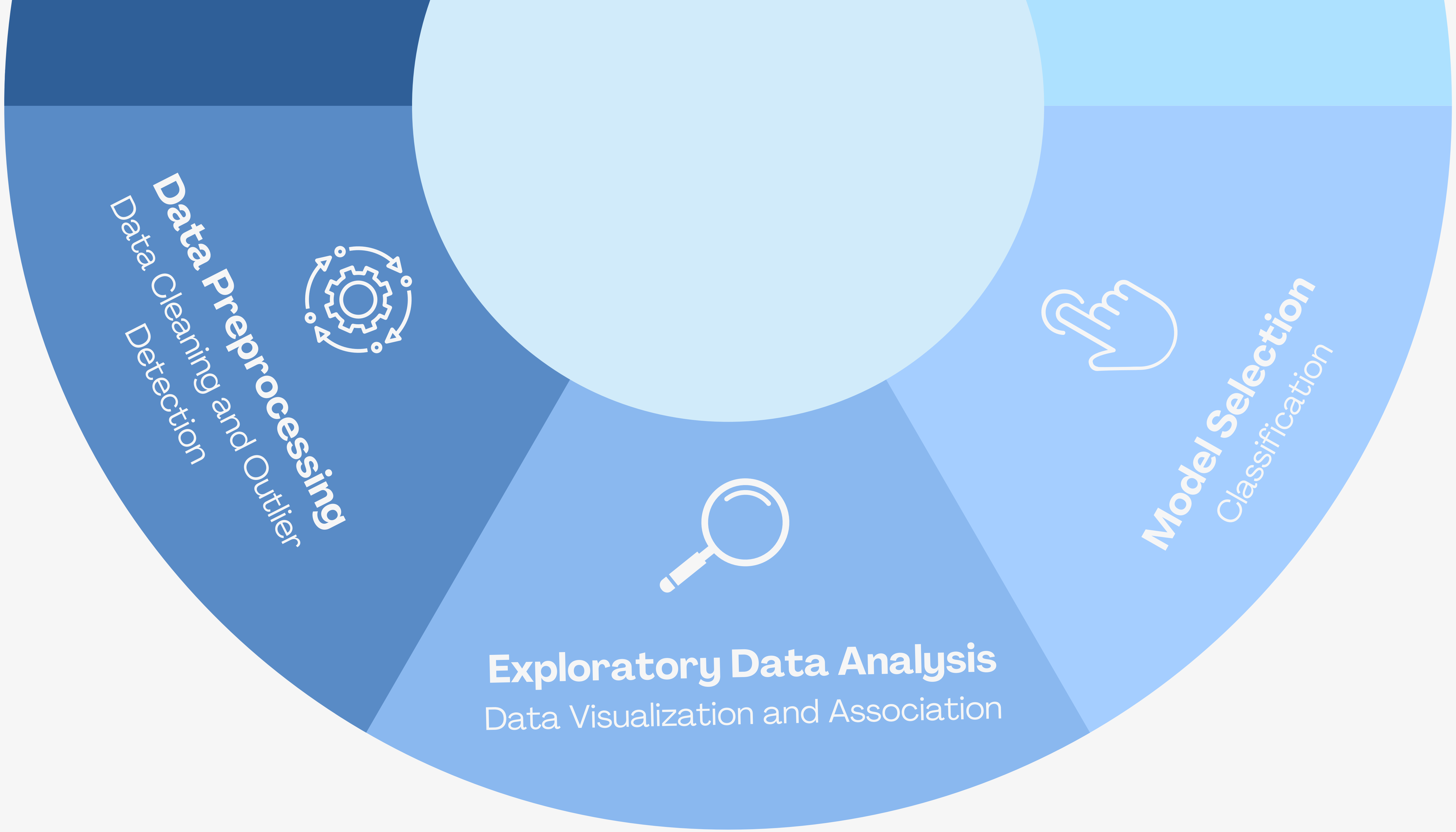
```python
#Display the plots
plt.tight_layout()
plt.show()
```



Outlier detection

```
columns_with_outliers = ['Annual Income', 'Monthly Debt', 'Current Credit Balance', 'Maximum Open Credit',
'Credit Score', 'Number of Credit Problems', 'Tax Liens']
def remove_outliers(data, col):
    z_scores = (data[col] - data[col].mean()) / data[col].std()
    data = data.loc[abs(z_scores) < 3]
    return data
for col in columns_with_outliers:
    training = remove_outliers(training, col)
```

```
training.to_csv('new_training_data.csv', index=False)
```



Bank Loan Approval

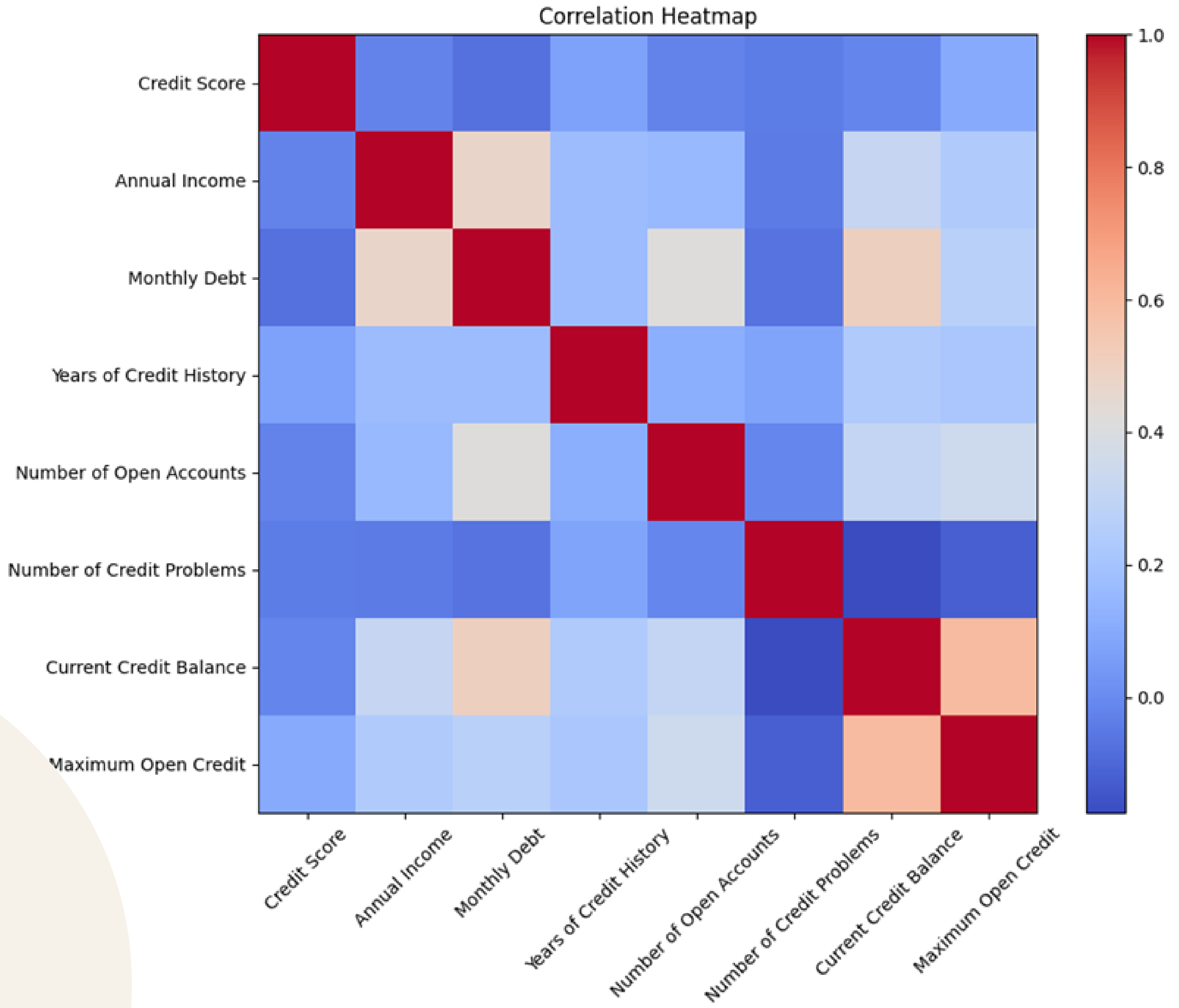
Exploratory Data Analysis

Uncover potential correlations between numerical attributes.

Correlation

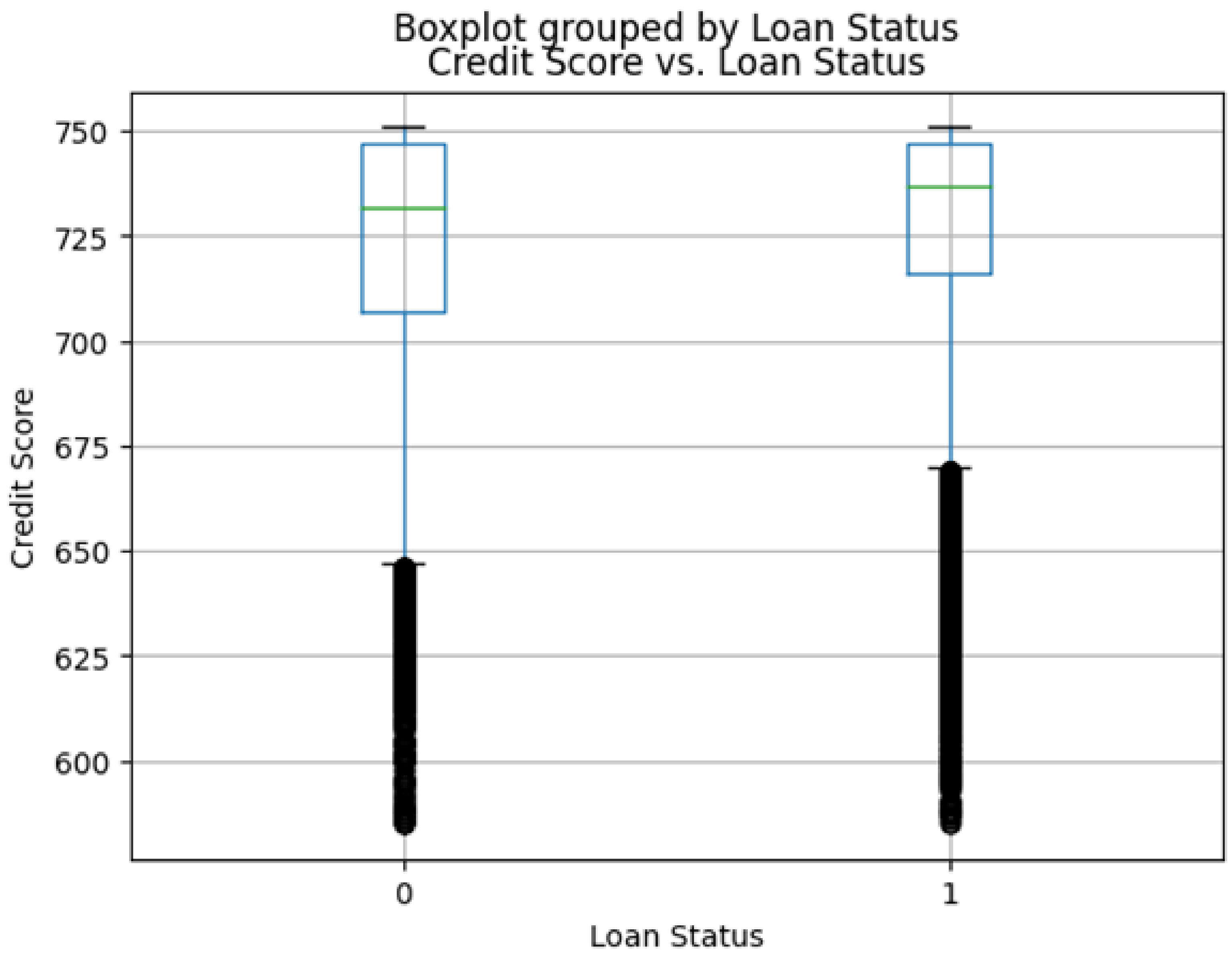
```
# Calculate correlation coefficients between numerical attributes
numerical_columns= ['Credit Score', 'Annual Income', 'Monthly Debt', 'Years of Credit History', 'Number of Open Accounts', 'Number of Credit Problems', 'Current Credit Balance', 'Maximum Open Credit']
correlation_matrix= training[numerical_columns].corr()
```

```
plt.figure(figsize=(10, 8))
plt.title('Correlation Heatmap')
heatmap= plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar(heatmap)
plt.xticks(range(len(numerical_columns)), numerical_columns, rotation=45)
plt.yticks(range(len(numerical_columns)), numerical_columns)
plt.show()
```



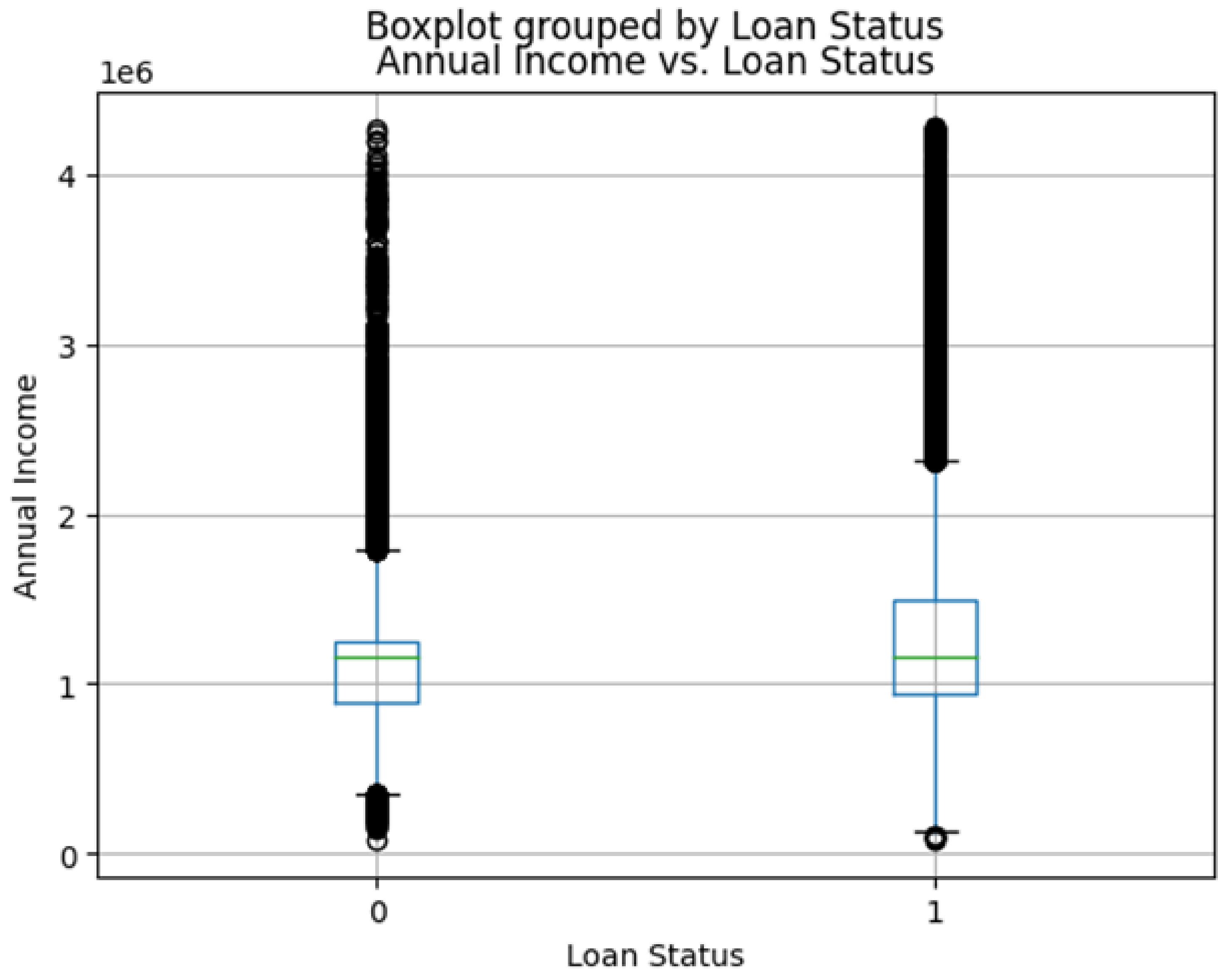
Relationship between Credit Score and Loan Status

```
plt.figure(figsize=(10, 6))
training.boxplot(column='Credit Score', by='Loan Status')
plt.title('Credit Score vs. Loan Status')
plt.xlabel('Loan Status')
plt.ylabel('Credit Score')
plt.show()
```



Relationship between Annual Income and Loan Status

```
plt.figure(figsize=(10, 6))
training.boxplot(column='Annual Income', by='Loan Status')
plt.title('Annual Income vs. Loan Status')
plt.xlabel('Loan Status')
plt.ylabel('Annual Income')
plt.show()
```



Effect of Loan Term on Loan Status

```
# Create a contingency table
loan_term_crosstab= pd.crosstab(training['Loan Status'], training['Term'])

# Perform a chi-squared test for independence
chi2, p, _, _ =stats.chi2_contingency(loan_term_crosstab)
print(f"Loan Term vs. Loan Status - Chi-squared Statistic: {chi2}")
print(f"Loan Term vs. Loan Status - P-Value: {p}")
```

```
Loan Term vs. Loan Status - Chi-squared Statistic: 1340.372478324552
Loan Term vs. Loan Status - P-Value: 1.9046640586743346e-293
```

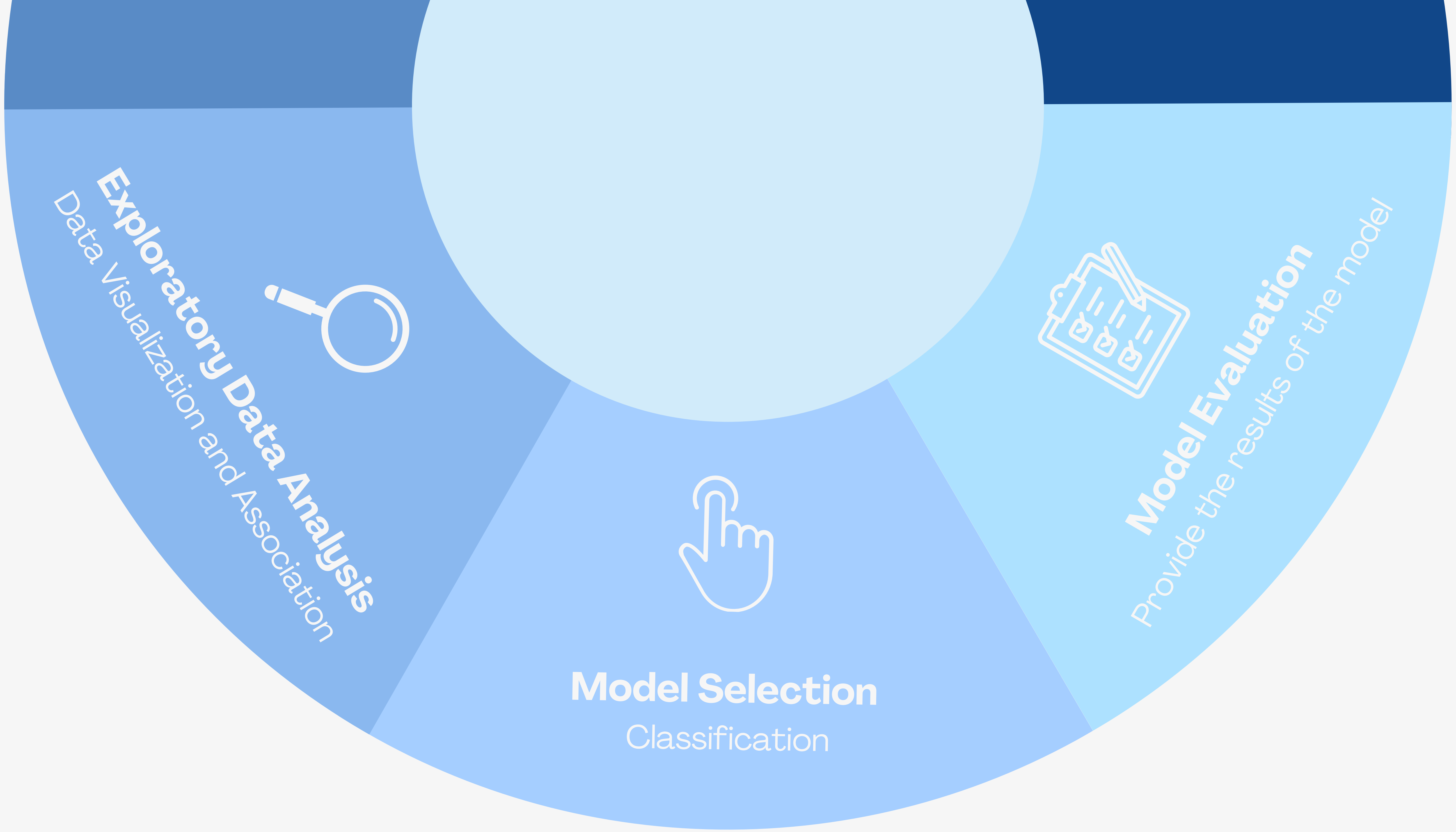
Effect of Home Ownership on Loan Approval

```
# Create a contingency table
home_ownership_crosstab= pd.crosstab(training['Loan Status'],
training['Home Ownership'])
# Create a contingency table
home_ownership_crosstab= pd.crosstab(training['Loan Status'],
training['Home Ownership'])
print("Contingency Table - Home Ownership vs. Loan Status:")
print(home_ownership_crosstab)
```

```
Contingency Table - Home Ownership vs. Loan Status:
Home Ownership     0      1     2      3
Loan Status
0                 24   7292  1549   8148
1                143  30838  5661  26148
```

```
# Perform a chi-squared test for independence
chi2, p, _, _ =stats.chi2_contingency(home_ownership_crosstab)
print(f"Home Ownership vs. Loan Status - Chi-squared Statistic: {chi2}")
print(f"Home Ownership vs. Loan Status - P-Value: {p}")
```

```
Home Ownership vs. Loan Status - Chi-squared Statistic: 236.0558702702138
Home Ownership vs. Loan Status - P-Value: 6.782577568085861e-51
```



Bank Loan Approval

Model Selection

Employing 3 models to classify new applicants into approved or denied categories

Preparing dataset

```
import pandas as pd
testing = pd.read_csv('new_testing_data.csv')
training = pd.read_csv('new_training_data.csv')
```

```
X_train = training.drop('Loan Status', axis=1)
y_train = training['Loan Status']
X_test = testing
```

```
#Encode y_train
from sklearn.preprocessing import LabelEncoder
labelencoder_y=LabelEncoder()
y_train = labelencoder_y.fit_transform(y_train)
```

Scaling

```
from sklearn.preprocessing import StandardScaler
SC = StandardScaler()
```

```
X_train_Scaled = SC.fit_transform(X_train)
X_test_Scaled = SC.fit_transform(X_test)
```

Test model: Logistic regression

```
from sklearn.linear_model import LogisticRegression
#Initialize the LR Model
logistic_regression = LogisticRegression()
```

```
#Train the model
logistic_regression.fit(X_train, y_train)
```

```
#Calculate the accuracy score
from sklearn.model_selection import cross_val_score
accuracy_lr = cross_val_score(estimator=logistic_regression, X=X_train, y=y_train, cv=10)
print(f"The accuracy of the Logistic Regression model is \t {accuracy_lr.mean()}")
print(f"The deviation in the accuracy of Logistic Regression model is \t {accuracy_lr.std()}")
```

```
The accuracy of the Logistic Regression model is        0.7867874657826123
The deviation in the accuracy of Logistic Regression model is    5.541816185047616e-05
```

Test model: KNN

```
from sklearn.neighbors import  KNeighborsClassifier
#Initialize the KNN Model
logistic_regression =  KNeighborsClassifier(n_neighbors=5)
```

```
#Train the model
knn..fit(X_train, y_train)
```

```
#Calculate the accuracy score
from sklearn.model_selection import cross_val_score
accuracy_lr = cross_val_score(estimator=knn, X=X_train, y=y_train, cv=10)
print(f"The accuracy of the KNN model is \t {accuracy_lr.mean()}")
print(f"The deviation in the accuracy of KNN model is \t {accuracy_lr.std()}")
```

```
The accuracy of the KNN model is         0.7465760504506473
The deviation in the accuracy of KNN model is    0.004741167747951971
```

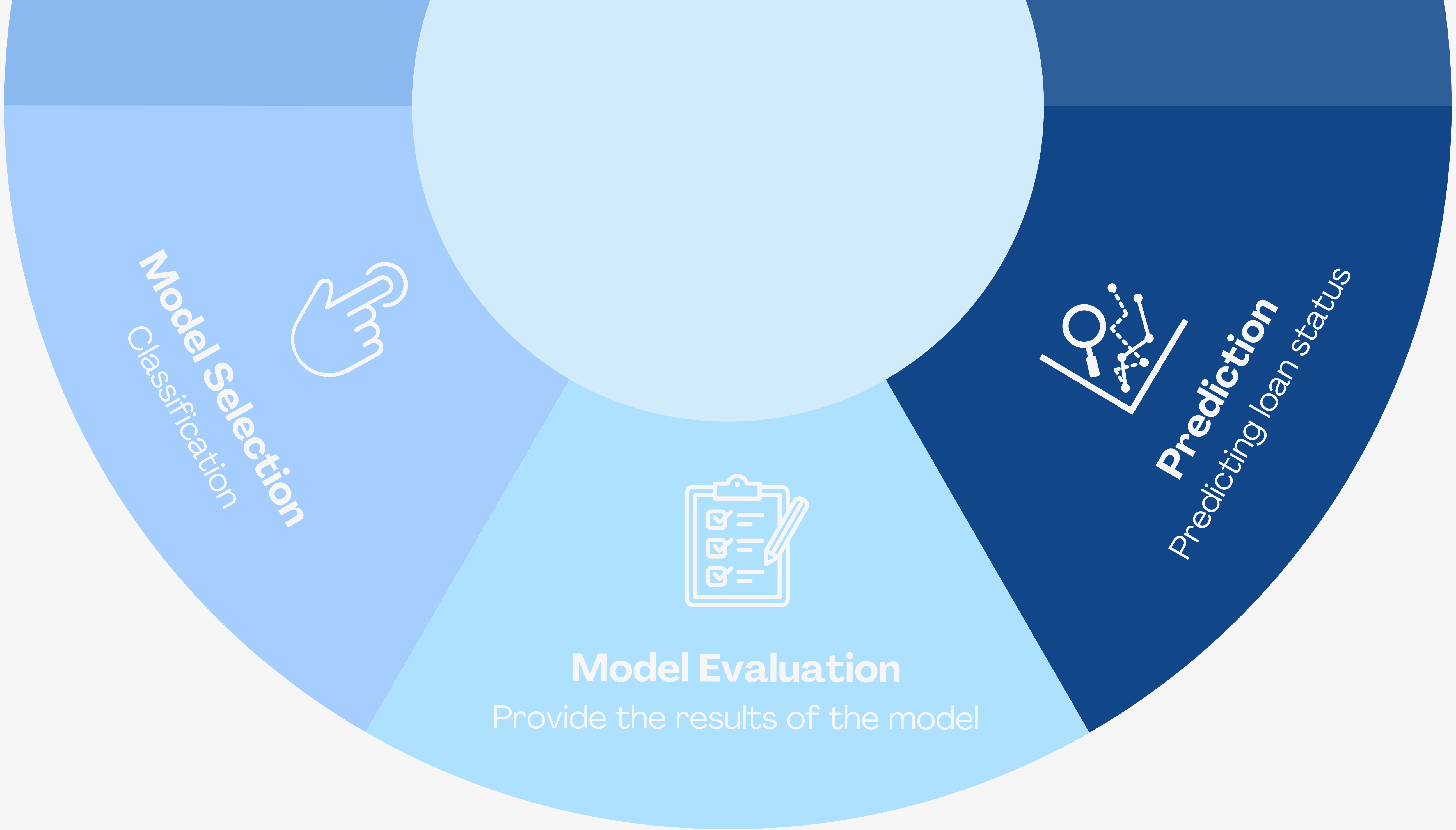
Test model: SVM

```
from sklearn.svm import SVC
#Initialize the SVM Model
svm = SVC()
```

```
#Train the model
svm.fit(X_train, y_train)
```

```
#Calculate the accuracy score
from sklearn.model_selection import cross_val_score
accuracy_lr = cross_val_score(estimator=svm, X=X_train, y=y_train, cv=10)
print(f"The accuracy of the SVM model is \t {accuracy_lr.mean()}")
print(f"The deviation in the accuracy of Logistic Regression model is \t {accuracy_lr.std()}")
```

```
The accuracy of the SVM model is         0.7868125284392539
The deviation in the accuracy of SVM model is    4.5179344785687504e-05
```



Bank Loan Approval

Model Evaluation

Evaluating the models' performance on the scoring dataset

Evaluate Logistic Regression

```
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict

# Evaluate Logistic Regression
y_pred_lr = cross_val_predict(logistic_regression, X_train, y_train, cv=10)
conf_matrix_lr = confusion_matrix(y_train, y_pred_lr)

print("Confusion Matrix for Logistic Regression:")
print(conf_matrix_lr)
```

```
Confusion Matrix for Logistic Regression:
[[    0 17013]
 [    2 62788]]
```

Evaluate K-Nearest Neighbors

```
# Evaluate K-Nearest Neighbors
y_pred_knn = cross_val_predict(knn, X_train, y_train, cv=10)
conf_matrix_knn = confusion_matrix(y_train, y_pred_knn)

print("Confusion Matrix for K-Nearest Neighbors:")
print(conf_matrix_knn)
```

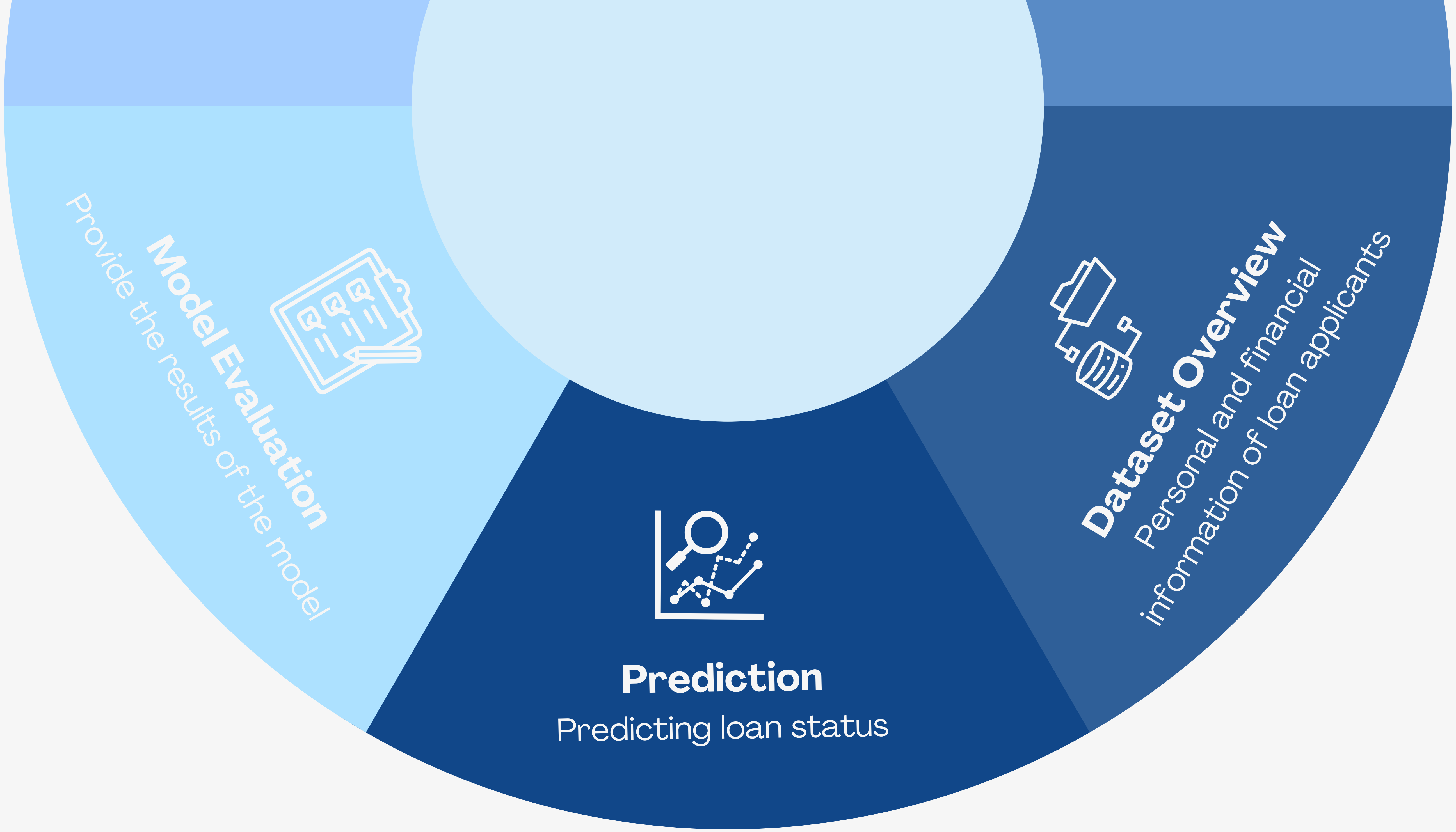
```
Confusion Matrix for K-Nearest Neighbors:
[[ 2295 14718]
 [ 5506 57284]]
```

Evaluate Support Vector Machine

```
# Evaluate Support Vector Machine
y_pred_svm= cross_val_predict(svm, X_train, y_train, cv=10)
conf_matrix_svm= confusion_matrix(y_train, y_pred_svm)

print("Confusion Matrix for Support Vector Machine:")
print(conf_matrix_svm)
```

```
Confusion Matrix for Logistic Regression:
[[    0 17013]
 [    2 62788]]
```



Bank Loan Approval

Prediction

Predicting the ability to repay debt of applicants and support credit approval decision.

Applying model

```
y_pred = logistic_regression.predict(X_test_Scaled)
testing.to_csv('predictions.csv', index=False)
print(testing)
```

	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Purpose	Monthly Debt	Years of Credit History	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens	Cluster	Loan Status
0	611314.0	1	747.0	2074116.0	1.0	1	3	42000.83	21.8	9.0	0.0	621908.0	1058970.0	0.0	0.0	1	Charged Off
1	266662.0	1	734.0	1919190.0	1.0	1	3	36624.40	19.4	11.0	0.0	679573.0	904442.0	0.0	0.0	1	Charged Off
2	153494.0	1	709.0	871112.0	2.0	3	3	8391.73	12.5	10.0	0.0	38532.0	388036.0	0.0	0.0	3	Charged Off
3	176242.0	1	727.0	780083.0	1.0	3	3	16771.87	16.5	16.0	1.0	156940.0	531322.0	1.0	0.0	0	Charged Off
4	321992.0	1	744.0	1761148.0	1.0	1	3	39478.77	26.0	14.0	0.0	359765.0	468072.0	0.0	0.0	1	Charged Off
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9994	157806.0	1	731.0	1514376.0	6.0	3	3	4795.41	12.5	9.0	0.0	87058.0	234410.0	0.0	0.0	3	Charged Off
9995	132550.0	1	718.0	763192.0	4.0	1	3	12401.87	9.9	8.0	0.0	74309.0	329692.0	0.0	0.0	3	Charged Off
9996	223212.0	0	747.0	853803.0	11.0	3	3	4354.42	27.2	8.0	1.0	99636.0	568370.0	1.0	0.0	0	Charged Off
9997	99999999.0	1	721.0	972097.0	1.0	1	3	12232.20	16.8	8.0	1.0	184984.0	240658.0	0.0	0.0	2	Charged Off
9998	99999999.0	1	748.0	1079960.0	6.0	1	3	12239.61	19.7	14.0	0.0	179018.0	607882.0	0.0	0.0	2	Charged Off

9999 rows × 17 columns

Cluster data

```
#Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(testing)
```

```
#Initialize K-Means with the desired number of clusters
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4, random_state=42)
```

```
#Fit the K-Means model to the data
kmeans.fit(data_scaled)
```

```
#Obtain cluster assignments for each data point
cluster_assignments = kmeans.labels_
```

Cluster data

```
#Add cluster assignments to the original DataFrame
testing['Cluster'] = cluster_assignments
print(testing)
```

	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Purpose	Monthly Debt	Years of Credit History	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens	Cluster
0	611314.0	1	747.0	2074116.0	1.0	1	3	42000.83	21.8	9.0	0.0	621908.0	1058970.0	0.0	0.0	1
1	266662.0	1	734.0	1919190.0	1.0	1	3	36624.40	19.4	11.0	0.0	679573.0	904442.0	0.0	0.0	1
2	153494.0	1	709.0	871112.0	2.0	3	3	8391.73	12.5	10.0	0.0	38532.0	388036.0	0.0	0.0	3
3	176242.0	1	727.0	780083.0	1.0	3	3	16771.87	16.5	16.0	1.0	156940.0	531322.0	1.0	0.0	0
4	321992.0	1	744.0	1761148.0	1.0	1	3	39478.77	26.0	14.0	0.0	359765.0	468072.0	0.0	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9994	157806.0	1	731.0	1514376.0	6.0	3	3	4795.41	12.5	9.0	0.0	87058.0	234410.0	0.0	0.0	3
9995	132550.0	1	718.0	763192.0	4.0	1	3	12401.87	9.9	8.0	0.0	74309.0	329692.0	0.0	0.0	3
9996	223212.0	0	747.0	853803.0	11.0	3	3	4354.42	27.2	8.0	1.0	99636.0	568370.0	1.0	0.0	0
9997	99999999.0	1	721.0	972097.0	1.0	1	3	12232.20	16.8	8.0	1.0	184984.0	240658.0	0.0	0.0	2
9998	99999999.0	1	748.0	1079960.0	6.0	1	3	12239.61	19.7	14.0	0.0	179018.0	607882.0	0.0	0.0	2

9999 rows × 16 columns

Calculate cluster

```
#Group the data by the 'Cluster' column
cluster_groups = testing.groupby('Cluster')
```

```
#Calculate mean for each cluster
cluster_means = cluster_groups.mean()
```

```
from IPython.display import display
#Display the mean results
print("Mean:")
display(cluster_means)
```

Mean:

Cluster	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Purpose	Monthly Debt	Years of Credit History	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens
0	1.109986e+07	0.753043	1036.803478	1.158262e+06	4.187826	1.938261	3.733043	15320.054887	19.876609	10.819130	1.261739	156711.025217	3.846473e+05	1.005217	0.192174
1	1.317279e+06	0.528215	972.953935	1.806371e+06	3.303647	1.416507	3.543570	30454.125712	21.506603	14.887524	0.027255	545850.431094	1.319302e+06	0.001919	0.016123
2	1.000000e+08	0.814213	726.271066	1.354057e+06	4.147208	1.893401	3.736041	17383.476914	18.272792	11.012183	0.021320	287247.844670	6.800957e+05	0.000000	0.009137
3	2.551055e+05	0.808138	1080.308614	1.007403e+06	4.353489	2.189960	3.882107	13350.972712	16.251150	9.271344	0.021297	194361.365279	4.415699e+05	0.000000	0.006845

Calculate cluster

```
#Calculate median for each cluster
cluster_median = cluster_groups.median()
```

```
from IPython.display import display
#Display the median results
print("Median:")
display(cluster_median)
```

```
Median:
```

Cluster	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Purpose	Monthly Debt	Years of Credit History	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens
0	245586.0	1.0	729.0	936985.0	3.0	2.0	3.0	14055.725	18.3	10.0	1.0	128155.0	329505.0	1.0	0.0
1	437756.0	1.0	729.0	1553288.0	2.0	1.0	3.0	27903.020	20.0	14.0	0.0	419767.0	875292.0	0.0	0.0
2	99999999.0	1.0	735.0	1235000.0	3.0	1.0	3.0	15493.360	16.9	10.0	0.0	228703.0	527010.0	0.0	0.0
3	223080.0	1.0	736.0	853803.0	3.0	3.0	3.0	12899.670	15.4	9.0	0.0	163267.0	369336.0	0.0	0.0

Calculate cluster

```python
#Calculate standard deviation for each cluster
cluster_std = cluster_groups.std()
```

```python
from IPython.display import display
#Display the standard deviation results
print("Standard Deviation:")
display(cluster_std)
```

Standard Deviation:

Cluster	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Purpose	Monthly Debt	Years of Credit History	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens
0	3.105913e+07	0.431429	1384.067299	6.078753e+05	3.542534	0.943401	2.116914	8922.617714	7.198559	4.457944	0.797197	128314.001513	2.605762e+05	0.406613	0.779709
1	9.317620e+06	0.499299	1240.424688	1.157493e+06	3.126544	0.770294	1.917290	14692.877531	7.154783	5.484794	0.180740	646960.897791	3.513553e+06	0.043777	0.143099
2	0.000000e+00	0.389132	24.784982	7.032298e+05	3.403677	0.955199	2.233196	10364.656260	6.868219	4.945419	0.151390	233389.491995	6.273161e+05	0.000000	0.095199
3	1.480988e+05	0.393802	1469.773308	4.227934e+05	3.474678	0.932816	2.368123	6943.998150	6.191396	3.701228	0.144386	141343.207282	3.289105e+05	0.000000	0.082461

Cluster analysis

1

2

3

4

Moderate likelihood of loan approval
(mean Loan Status: 0.76)
Highest Current Loan Amount
(mean: 13.28 million)
Majority own homes
(mean: 1.92)
Relatively high Monthly Debt
(mean: 15,728 million)
Diverse loan purposes

Cluster analysis

1

2

3

4

Very high likelihood of loan approval
(mean Loan Status: 1.0)
Extremely high mean Current Loan Amount
(mean: 17.79 million)
Very low occurrences of credit problems
(mean: 0.015)
Bankruptcies and Tax Liens are extremely rare

Cluster analysis

1

2

3

4

High likelihood of loan approval
(mean Loan Status: 0.90)
Highest Monthly Debt
(mean: 31,780)
Significantly higher number of open accounts
(mean: 15.17)
Very high Maximum Open Credit
(mean: 1.77 million)

Cluster analysis

1

2

3

4

Very low likelihood of loan approval
(mean Loan Status: 0.0)
Very high Credit Score
(mean: 2131.91)
Very low Current Loan Amount
(mean: 311.56)
Relatively high Annual Income
(mean: 1.15 million)
Relatively high Monthly Debt
(mean: 16,538)

Classfication

```
# Load Iris Dataset
X, Y = load_iris(return_X_y=True)
```

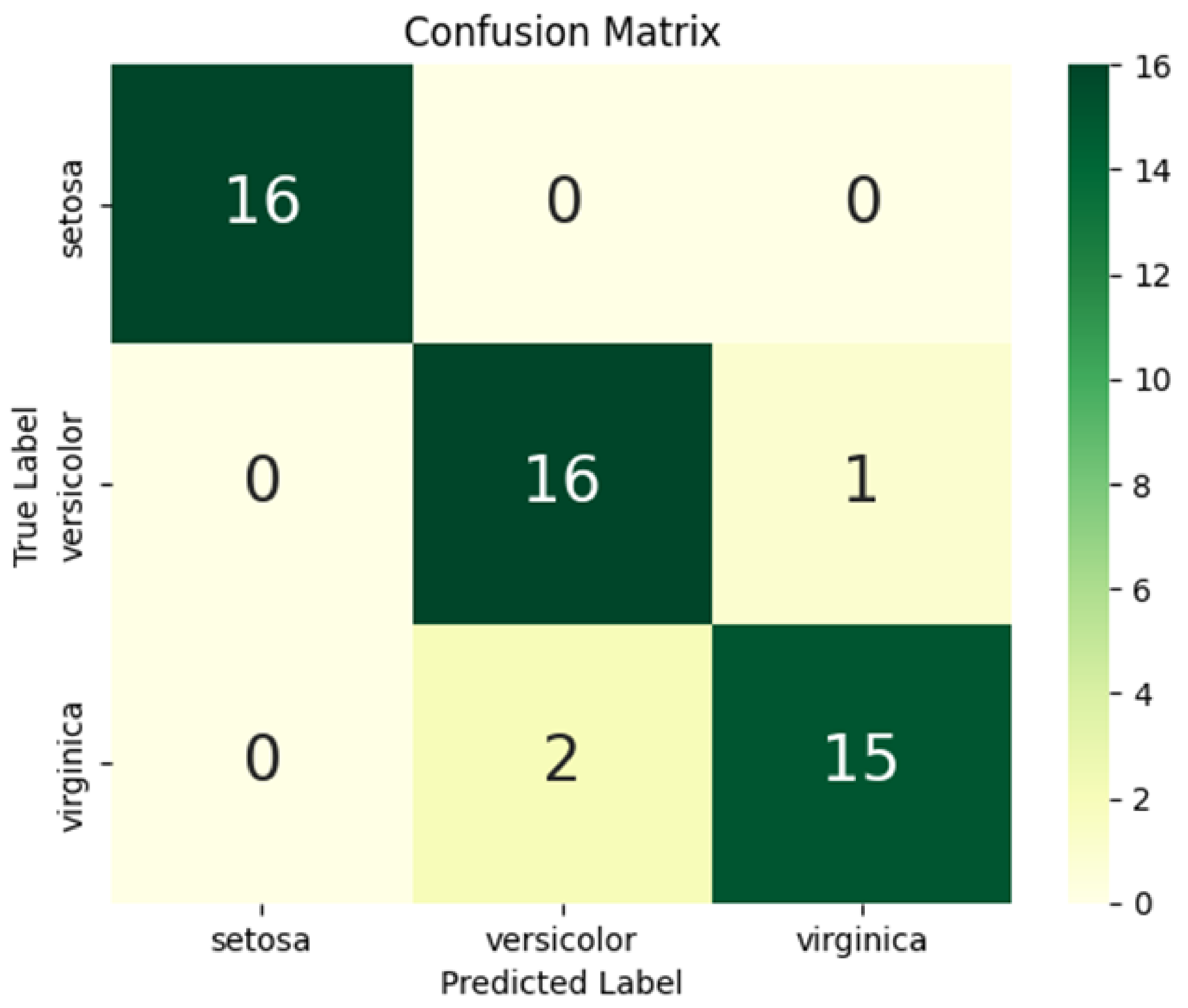
```
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # Scale the entire X data
```

```
# Split data into training and testing sets
train_x, test_x, train_y, test_y = train_test_split(X_scaled, Y, test_size=0.33, random_state=5)
```

```
# Applying Logistic Regression Modeling
log_model = LogisticRegression()
log_model.fit(train_x, train_y)
```

Classification

```
# Calculate Confusion Matrix
df_cm= pd.DataFrame(confusion_matrix(test_y, pred), index=load_iris().target_names, columns=load_iris().target_names)
sns.heatmap(df_cm, annot=True, cmap='YlGn', annot_kws={"size": 20}, fmt='g')
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



Classification

```
# Calculating the accuracy score using cross-validation
from sklearn.model_selection import cross_val_score
```

```
logistic_regression = LogisticRegression()
logistic_regression.fit(train_x, train_y)
```

```
accuracy_lr = cross_val_score(estimator=logistic_regression, X=train_x, y=train_y, cv=10)
print(f"The accuracy of the Logistic Regression model is \t {accuracy_lr.mean()}")
print(f"The deviation in the accuracy of Logistic Regression model is \t {accuracy_lr.std()}")
```

```
The accuracy of the Logistic Regression model is         0.97
The deviation in the accuracy of Logistic Regression model is     0.06403124237432847
```

THANK YOU FOR LISTENING