**Task 1.1 – Logistic Regression task**

**Here I used Google Collab as my IDE to Run my Code and Execute the sample solutions.**

1. Describe the dataset (e.g., Descriptive statistics, Missing values, Target rate, Categorical Features, Correlation Matrix, Outlier Detection, Data Distributions).

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Loading the dataset

file\_path = "https://files.challengerocket.com/files/lions-den-ing-2024/development\_sample.csv"

data = pd.read\_csv(file\_path)

# Descriptive statistics

print("Descriptive statistics:")

print(data.describe())

# Missing values

print("\nMissing values:")

print(data.isnull().sum())

# Target rate

target\_rate = data['target'].mean() \* 100 # Multiply by 100 to get percentage

print(f"\nTarget rate (probability of default): {target\_rate:.2f}%")

# Categorical features

categorical\_features = data.select\_dtypes(include=['object']).columns

for feature in categorical\_features:

print(f"\n{feature} distribution:")

print(data[feature].value\_counts(normalize=True))

# Correlation matrix

corr\_matrix = data.corr()

plt.figure(figsize=(25, 25))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title("Correlation Matrix")

plt.show()

# Outlier detection (example for numerical features)

numerical\_features = data.select\_dtypes(include=['int64', 'float64']).columns

for feature in numerical\_features:

sns.boxplot(x=data[feature])

plt.title(f"Boxplot of {feature}")

plt.show()

# Data distributions (example for numerical features)

for feature in numerical\_features:

sns.histplot(data=data, x=feature, kde=True)

plt.title(f"Distribution of {feature}")

plt.show()

**Descriptive statistics:**

target Var1 Var2 Var6 Var8

count 36718.000000 50000.00000 48597.000000 50000.00000 20827.000000

mean 0.030693 1.33786 1.915098 1.54724 35911.341048

std 0.172488 0.66462 0.861268 1.23719 18153.197904

min 0.000000 1.00000 1.000000 1.00000 4900.000000

25% 0.000000 1.00000 1.000000 1.00000 21400.000000

50% 0.000000 1.00000 2.000000 1.00000 31400.000000

75% 0.000000 1.00000 3.000000 1.00000 47100.000000

max 1.000000 4.00000 3.000000 6.00000 138300.000000

Var11 Var14 Var15 Var16 Where 27 Where 28

count 50000.000000 50000.000000 50000.000000 50000.000000 49949.000000 49943.000000

mean 4.753460 1.276780 0.764300 1.023900 0.012148 0.046205

std 0.973621 1.270891 0.982652 1.174647 0.109433 0.239542

min 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000

25% 5.000000 0.000000 0.000000 0.000000 0.000000 0.000000

50% 5.000000 1.000000 1.000000 1.000000 0.000000 0.000000

75% 5.000000 2.000000 1.000000 2.000000 0.000000 0.000000

max 7.000000 4.000000 5.000000 8.000000 1.000000 10.856900

**Missing values:**

ID 0

customer\_id 0

application\_date 0

target 13282

Application\_status 0

Var1 0

Var2 1403

Var3 1403

Var4 0

Var5 0

Var6 0

Var7 0

Var8 29173

Var9 0

Var10 37537

Var11 0

Var12 37536

Var13 0

Var14 0

Var15 0

Var16 0

Var17 40

Var18 37412

Var19 28534

Var20 0

Where 21 3

Where 22 13

Where 23 46

Where 24 81

Where 25 10092

Where 26 19853

Where 27 51

Where 28 57

Where 29 57

Where 30 81

\_r\_ 235

**Target rate** (probability of default): 3.07%

**Data Distributions**

**ID distribution:**

11034977 0.00002

11068327 0.00002

11068305 0.00002

11068306 0.00002

11068307 0.00002

...

11051645 0.00002

11051646 0.00002

11051647 0.00002

11051648 0.00002

11084976 0.00002

Name: ID, Length: 50000, dtype: float64

**customer\_id distribution:**

32835411 0.00020

32616213 0.00016

32500995 0.00016

32656364 0.00016

32367333 0.00014

...

32877486 0.00002

32527174 0.00002

32540619 0.00002

32612072 0.00002

32834539 0.00002

Name: customer\_id, Length: 36048, dtype: float64

application\_date distribution:

2019-10-09 00:00:00 0.00060

2019-11-14 00:00:00 0.00060

2018-08-07 00:00:00 0.00058

2020-01-12 00:00:00 0.00054

2019-06-12 00:00:00 0.00052

...

24Ap r2016 0:00:00 0.00002

2010-02-17 00:00:00 0.00002

2010-10-04 00:00:00 0.00002

2010-03-02 00:00:00 0.00002

2010-04-06 00:00:00 0.00002

Name: application\_date, Length: 3988, dtype: float64

Application\_status distribution:

Approved 0.73428

Rejected 0.26564

Approved 0.00002

A pproved 0.00002

App roved 0.00002

Ap proved 0.00002

Name: Application\_status, dtype: float64

Var3 distribution:

1 0.494742

2 0.331913

3 0.165730

Direct 0.006688

Online 0.000926

Name: Var3, dtype: float64

Var4 distribution:

4000 0.00652

4100 0.00646

4600 0.00636

4500 0.00632

4200 0.00606

...

1200 0.00002

2350 0 0.00002

42100 0.00002

28800 0.00002

62500 0.00002

Name: Var4, Length: 468, dtype: float64

Var5 distribution:

9 0.08722

12 0.07480

18 0.06568

24 0.06436

21 0.06232

...

273 0.00020

300 0.00020

294 0.00016

297 0.00014

19 8 0.00002

Name: Var5, Length: 104, dtype: float64

Var7 distribution:

323.87 0.00090

763.41 0.00078

508.94 0.00074

890.64 0.00074

300.74 0.00072

...

234.73 0.00002

930.75 0.00002

201.69 0.00002

1340.17 0.00002

3963.94 0.00002

Name: Var7, Length: 19734, dtype: float64

Var9 distribution:

6120 0.01296

5640 0.01266

5040 0.01262

5880 0.01258

5400 0.01244

...

31320 0.00002

1680 0.00002

34920 0.00002

38160 0.00002

32160 0.00002

Name: Var9, Length: 291, dtype: float64

Var10 distribution:

0 0.019819

4200 0.018374

4100 0.017091

4600 0.016930

4400 0.016208

...

21500 0.000080

23900 0.000080

240 0 0.000080

20500 0.000080

22000 0.000080

Name: Var10, Length: 202, dtype: float64

Var12 distribution:

5 0.702664

4 0.120507

6 0.088575

2 0.030006

1 0.020700

7 0.019817

3 0.017571

0.000160

Name: Var12, dtype: float64

Var13 distribution:

9999-12-31 00:00:00 0.01568

2008-02-16 00:00:00 0.00032

2000-05-08 00:00:00 0.00032

1999-01-05 00:00:00 0.00030

2009-10-24 00:00:00 0.00030

...

1990-09-09 00:00:00 0.00002

1990-02-12 00:00:00 0.00002

2014-04-26 00:00:00 0.00002

2016-04-21 00:00:00 0.00002

2020-11-14 00:00:00 0.00002

Name: Var13, Length: 10937, dtype: float64

Var17 distribution:

2150.88 0.00006

5873.82 0.00006

3357.12 0.00006

3224.67 0.00006

2969.90 0.00006

...

2274.20 0.00002

1194.61 0.00002

2357.40 0.00002

3318.63 0.00002

3681.02 0.00002

Name: Var17, Length: 48558, dtype: float64

Var18 distribution:

1 0.843502

0 0.156180

0.000318

Name: Var18, dtype: float64

Var19 distribution:

1 0.708283

0 0.261949

0.029768

Name: Var19, dtype: float64

Var20 distribution:

0 0.64482

1 0.23640

2 0.07580

3 0.02452

4 0.01026

5 0.00402

6 0.00208

7 0.00096

8 0.00048

9 0.00022

10 0.00014

11 0.00012

14 0.00004

13 0.00004

0.0.5.5.0.15765.29.7949.87.0.0.0.5035.0.482679397 0.00002

12 0.00002

0.0.0.0.54000 0.00002

0.0.0.0.73200 0.00002

15 0.00002

Name: Var20, dtype: float64

Where 21 distribution:

0 0.465068

1 0.288297

2 0.135948

3 0.059084

4 0.026502

5 0.012381

6 0.006440

7 0.002480

8 0.001360

9 0.001060

10 0.000460

11 0.000240

12 0.000220

14 0.000100

13 0.000080

18 0.000020

0.0.0.69000 0.000020

15 0.000020

0.0.0.50400.4369.43.68978.3.0.0.0.8569.0.536752795 0.000020

0.0.0.0.55525.94.0.0.0.0.8569.0.289570236 0.000020

0.0.0.0.7680.19.0.0.0.0.8569.0.410991725 0.000020

0.0.0.0.94983.84.0.0.0.0.8569.0.129614029 0.000020

0.0.0.54000 0.000020

0.0.0.52800 0.000020

16 0.000020

0.0.0.55800 0.000020

0.0.0.0.8366.08.0.0.0.0.5128.0.860105189 0.000020

0.13.13.0.2693.21.34821.81.0.0.0.5035.0.467728368 0.000020

22 0.000020

Name: Where 21, dtype: float64

Where 22 distribution:

0 0.359453

1 0.293096

2 0.169424

3 0.086002

4 0.045372

5 0.021706

6 0.011843

7 0.005241

8 0.002821

9 0.001720

10 0.001040

11 0.000680

12 0.000420

13 0.000180

14 0.000160

16 0.000060

0.0.36600 0.000040

0.0.32400 0.000040

0.0.34200 0.000040

15 0.000040

0.0.28800 0.000040

0.0.22800 0.000040

0.0.0.54697.12.0.0.0.20.8569.0.520960385 0.000020

0.0.49200 0.000020

0.0.66600.31551.69.18954.18.0.0.0.8569.0.763850114 0.000020

0.0.85800 0.000020

0.0.48000.9675.88.46797.98.0.0.0.8569.0.288187149 0.000020

0.0.67800 0.000020

0.0.34800.10872.95.3677.23.0.0.0.8882.0.509921799 0.000020

0.0.65400 0.000020

0.1.64800.8697.96.21492.16.0.0.0.8882.0.595980694 0.000020

0.0.55800 0.000020

0.0.42600 0.000020

5.5.38400.21653.67.5820.12.0.0.0.5035.0.216735264 0.000020

0.0.46800.5436.69.46212.4.0.0.0.5966.0.417033168 0.000020

0.0.43800.3229.13.7654.06.0.0.0.5966.0.552421432 0.000020

0.1.39600.9010.88.9116.5.0.0.0.5966.0.360834195 0.000020

0.0.51600.8451.82.0.0.0.0.5128.0.542550238 0.000020

0.0.22800.5320.66.6834.13.0.0.0.5128.0.842855119 0.000020

0.1.46800 0.000020

0.0.40200 0.000020

0.0.25800.4200.21.12696.78.0.0.0.5035.0.29775831 0.000020

0.0.38400.2329.22.9523.72.0.0.0.5035.0.161312391 0.000020

0.0.51600.9186.97.31343.45.0.0.70.4918.0.017756558 0.000020

1.6.0.6512.0.0.0.0.4918.0.100250728 0.000020

0.0.27000 0.000020

0.3.29400.8073.49.9479.2.0.0.0.3899.0.280260586 0.000020

18 0.000020

23 0.000020

Name: Where 22, dtype: float64

Where 23 distribution:

1.000 0.279797

0.000 0.278436

2.000 0.192817

3.000 0.111242

4.000 0.064019

...

2.534 0.000020

0.438 0.000020

1.270 0.000020

1.348 0.000020

1.570 0.000020

Name: Where 23, Length: 66, dtype: float64

Where 24 distribution:

0 0.852120

25200 0.002204

26400 0.002164

30600 0.002123

31200 0.001963

...

151200 0.000020

126600 0.000020

148200 0.000020

177000 0.000020

168000 0.000020

Name: Where 24, Length: 253, dtype: float64

Where 25 distribution:

0.0.0.5966 0.000100

5040.28 0.000075

9089.3 0.000075

6186.49 0.000075

1885.9 0.000050

...

18331.78 0.000025

2720.65 0.000025

11379.37 0.000025

2440.01 0.000025

9288.87 0.000025

Name: Where 25, Length: 39510, dtype: float64

Where 26 distribution:

0.00 0.125087

43204.20 0.000066

10655.65 0.000066

6991.05 0.000066

5535.96 0.000066

...

17974.30 0.000033

27277.59 0.000033

15098.44 0.000033

18590.12 0.000033

255018.31 0.000033

Name: Where 26, Length: 26314, dtype: float64

Where 29 distribution:

0 0.536672

10 0.341189

20 0.073324

30 0.023327

40 0.010432

50 0.004545

60 0.002523

70 0.001522

0.5966 0.001422

80 0.001141

0.8569 0.000801

90 0.000601

0.5035 0.000280

120 0.000240

100 0.000240

0.5128 0.000200

130 0.000160

110 0.000160

12832 0.000120

0.8882 0.000120

150 0.000080

140 0.000060

250 0.000040

8882 0.000040

11839 0.000040

180 0.000040

210 0.000040

0.183405925 0.000020

0.369214455 0.000020

0.50940547 0.000020

0.279104266 0.000020

0.325438749 0.000020

0.070877029 0.000020

0.115610443 0.000020

0.962950981 0.000020

0.4918 0.000020

0.175151368 0.000020

0.527079531 0.000020

240 0.000020

1 0 0.000020

190 0.000020

0.219661515 0.000020

160 0.000020

5966 0.000020

0.037666702 0.000020

0.908791567 0.000020

5035 0.000020

0.898728846 0.000020

170 0.000020

0.447416305 0.000020

0.86038791 0.000020

0.303128075 0.000020

0.910052023 0.000020

0.526074913 0.000020

0.997440197 0.000020

0.103422901 0.000020

0.96491505 0.000020

0.922722252 0.000020

0.389607168 0.000020

Name: Where 29, dtype: float64

Where 30 distribution:

11839.000000 0.221599

12832.000000 0.114145

8569.000000 0.101565

8882.000000 0.098820

5966.000000 0.095375

...

0.572192 0.000020

0.412230 0.000020

0.006317 0.000020

0.597604 0.000020

0.794039 0.000020

Name: Where 30, Length: 165, dtype: float64

\_r\_ distribution:

0.267045 0.00002

0.222742 0.00002

0.855641 0.00002

0.762778 0.00002

0.450581 0.00002

...

0.836122 0.00002

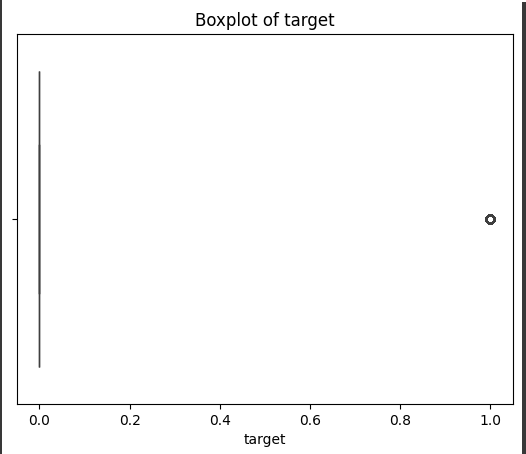
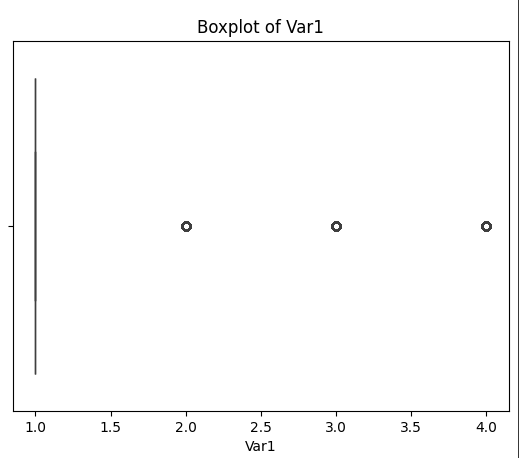
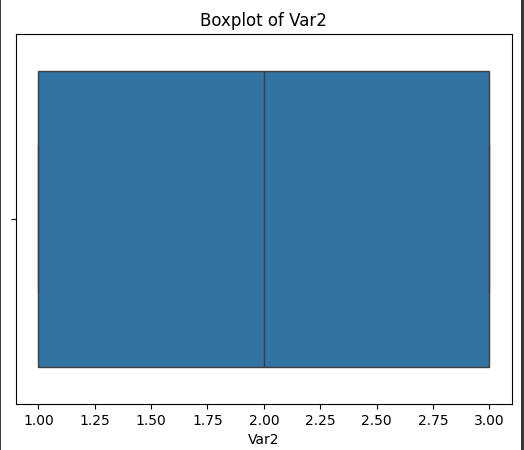
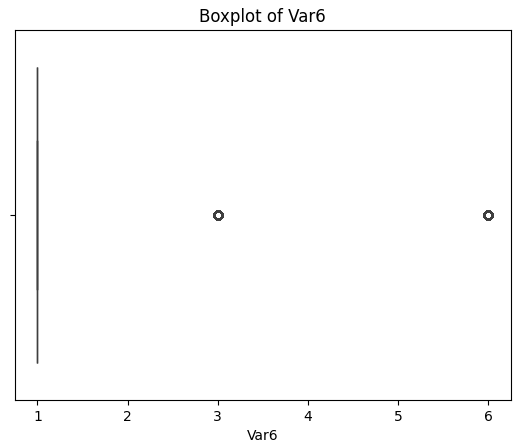
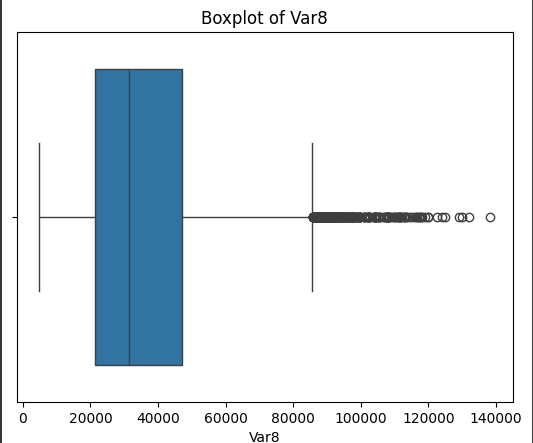
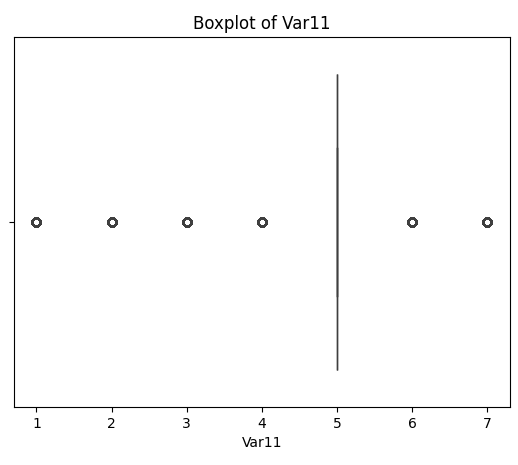
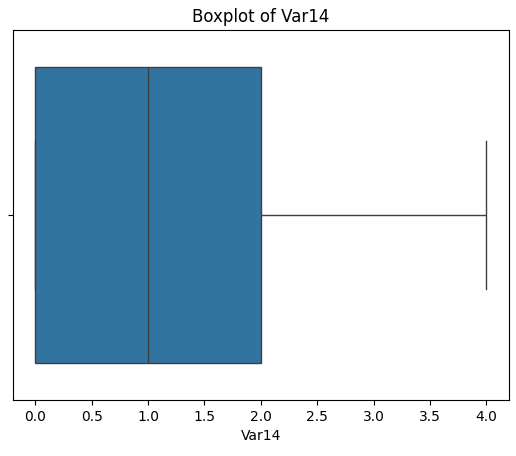
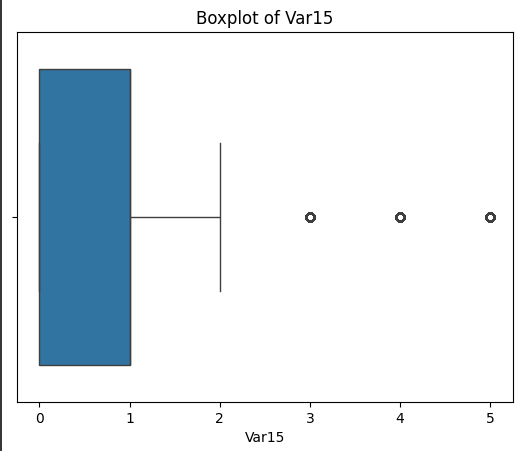
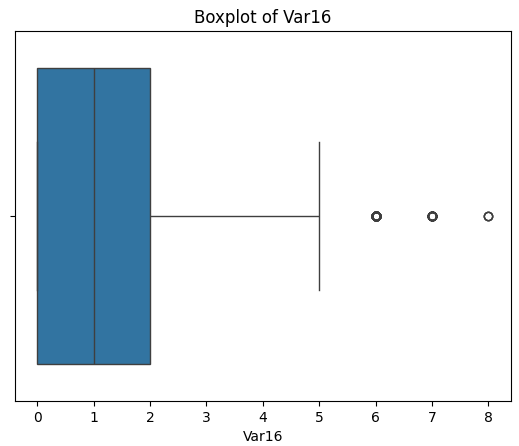
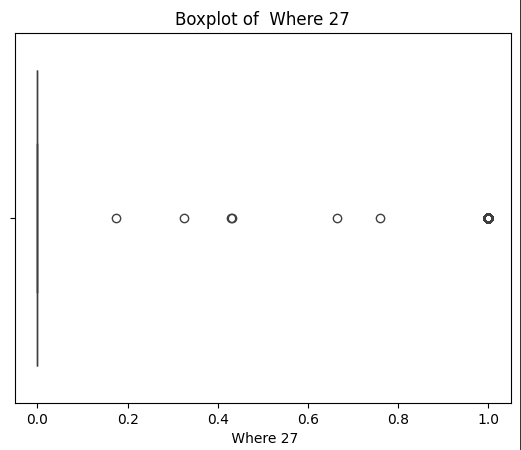
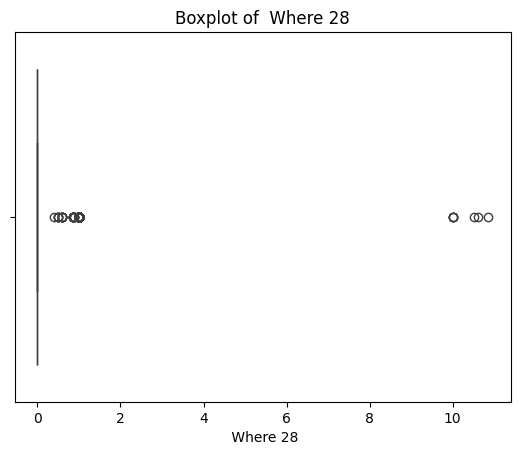
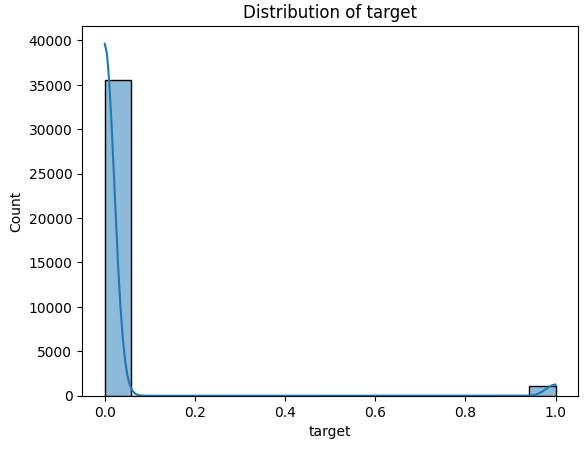
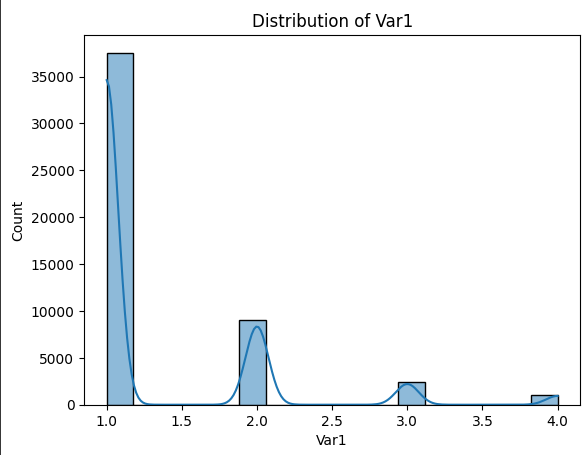
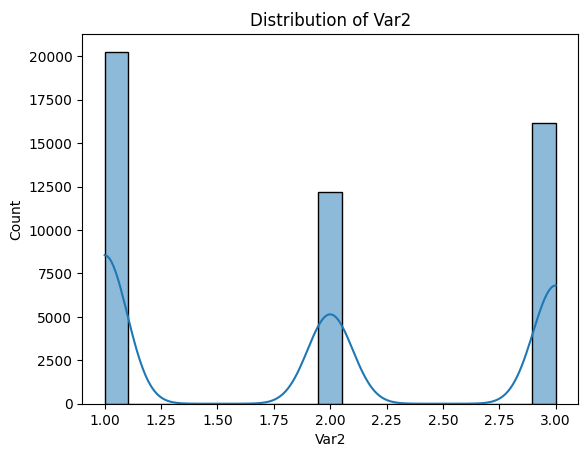
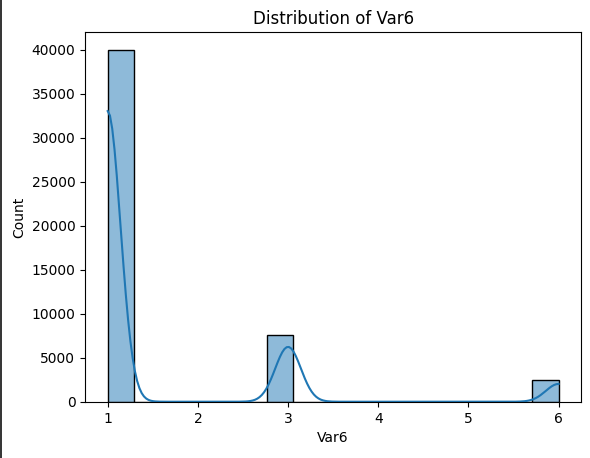
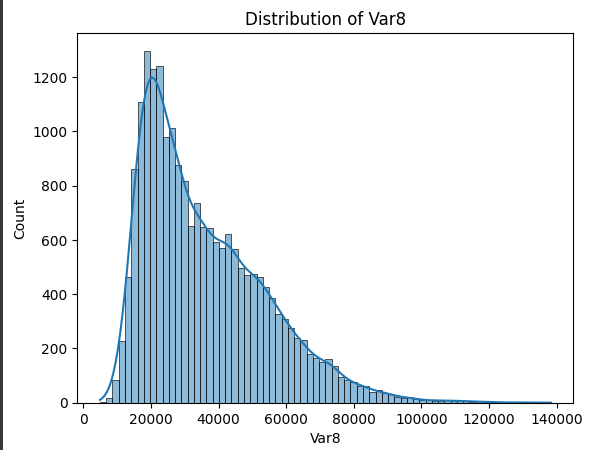
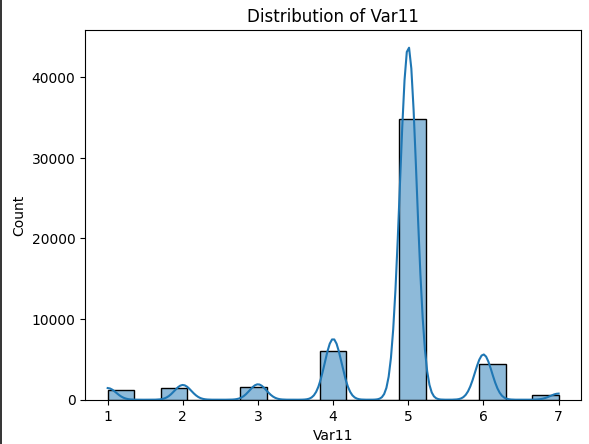
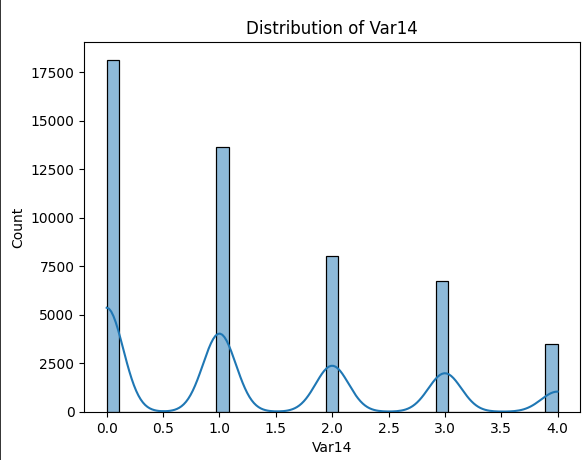
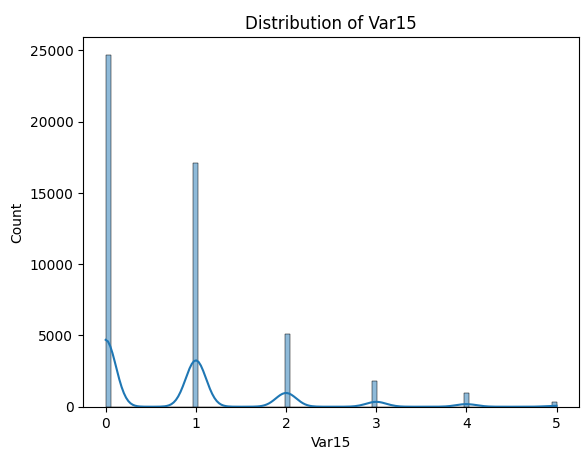
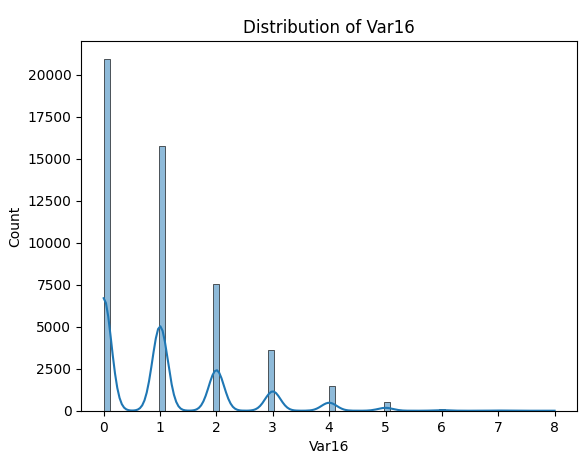
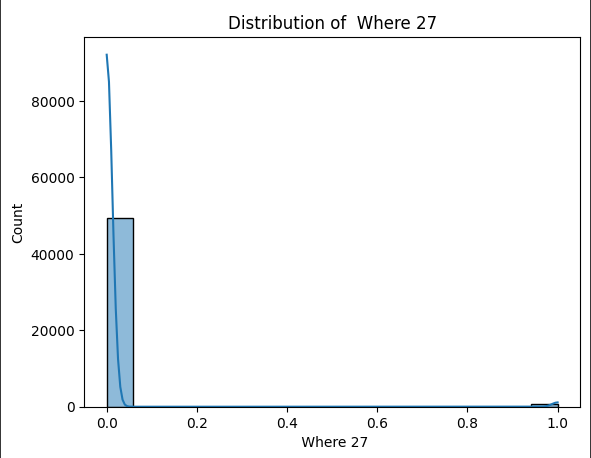
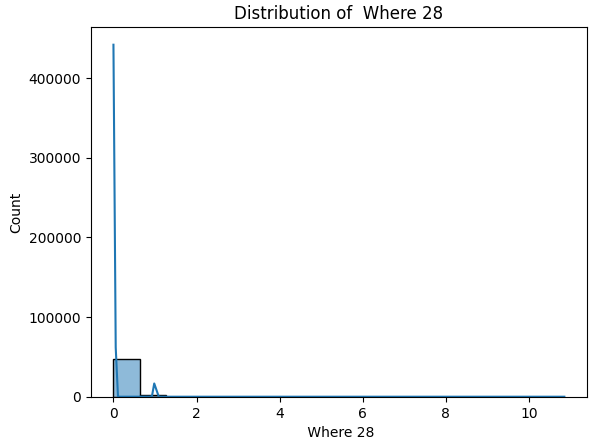
0.541544 0.00002

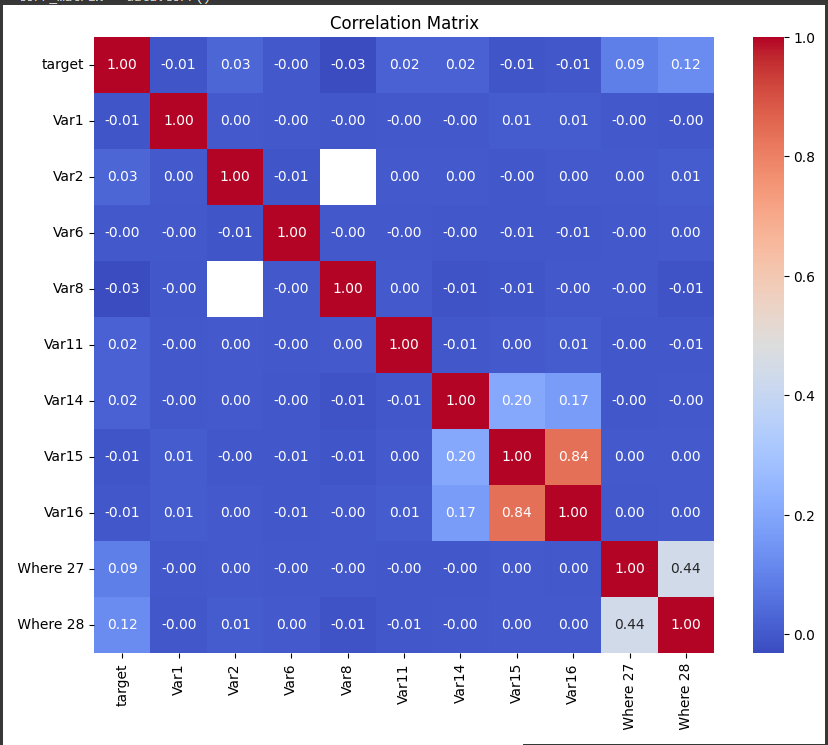
0.878617 0.00002

0.350446 0.00002

0.733623 0.00002

Name: \_r\_, Length: 49765, dtype: float64





1. Describe the feature engineering procedure and the data treatments you followed (if any).

Let's discuss the feature engineering procedures and data treatments applied for the task we performed earlier, which involved describing the dataset. Since we haven't explicitly performed feature engineering in that task, I'll outline some common feature engineering techniques that could be applied:

Handling Categorical Features:

If the dataset contains categorical features, we would typically encode them using techniques such as one-hot encoding or label encoding to convert them into numerical representations that can be used as input in the model.

Dealing with Missing Values:

If the dataset has missing values, we would need to decide how to handle them. Common approaches include imputing missing values using techniques like mean imputation, median imputation, or mode imputation, or dropping rows or columns with missing values if they cannot be reliably imputed.

Feature Scaling:

Depending on the modeling algorithm used (such as logistic regression), we might scale numerical features to a similar range using techniques like standardization or normalization to prevent features with larger scales from dominating the model.

Feature Selection:

Feature selection involves identifying and selecting the most relevant features that have a significant impact on the target variable. While we didn't explicitly perform feature selection in the initial task, it's an important step in model building to improve model performance and interpretability.

Creating New Features:

We could derive new features from existing ones based on domain knowledge or specific hypotheses about the data. For example, creating interaction terms or aggregating features could capture additional information or patterns in the data.

Handling Imbalanced Classes:

If the target variable is imbalanced (e.g., one class significantly outweighs the other), techniques such as oversampling, undersampling, or using class weights during model training might be applied to balance the classes and improve model performance.

Since the initial task focused primarily on describing the dataset, we didn't perform explicit feature engineering procedures or data treatments beyond basic data exploration and analysis. However, these are some common feature engineering techniques that could be applied depending on the specific characteristics of the dataset and the modeling goals. Let me know if you have any further questions or if you'd like to discuss any specific aspect in more detail!

1. Describe the model selection process you applied (e.g., criteria for feature selection, estimation technique of the model parameters).

Since the initial task focused on describing the dataset rather than building a predictive model, we didn't perform a formal model selection process. However, I can outline a typical model selection process that could be applied for credit risk modeling using logistic regression:

Feature Selection:

Before building the model, it's essential to select relevant features that have a significant impact on predicting the target variable (e.g., default or not). Feature selection techniques include univariate feature selection, recursive feature elimination, and feature importance ranking based on domain knowledge or statistical tests.

Model Building:

Once the features are selected, logistic regression can be used as the modeling technique. Logistic regression is well-suited for binary classification tasks like credit risk modeling, where the goal is to predict the probability of default based on various predictors.

Parameter Estimation:

Logistic regression estimates the parameters (coefficients) of the model using maximum likelihood estimation (MLE). The coefficients represent the log odds of the target variable given the predictors. The estimation technique aims to maximize the likelihood of observing the actual target values given the model's predictions.

Model Evaluation:

After building the logistic regression model, it's essential to evaluate its performance using appropriate metrics such as accuracy, precision, recall, F1 score, and ROC-AUC. These metrics assess the model's ability to correctly classify instances of default and non-default and its discriminatory power in distinguishing between the two classes.

Cross-Validation:

To ensure the robustness of the model, cross-validation techniques such as k-fold cross-validation can be applied to assess its performance on different subsets of the data. This helps detect overfitting and ensures that the model generalizes well to new data.

Hyperparameter Tuning:

Although logistic regression has minimal hyperparameters compared to other algorithms, it's still essential to tune them for optimal model performance. Techniques like grid search or random search can be used to search for the best hyperparameter values, such as regularization strength (e.g., L1 or L2 regularization).

Comparison with Challenger Models:

Optionally, the performance of the logistic regression model can be compared with other challenger models (alternative classification algorithms) to identify the best-performing model for the specific task of credit risk modeling.

By following these steps, we can systematically select and build a logistic regression model for credit risk modeling, ensuring its reliability and predictive accuracy. Let me know if you have any further questions or if you'd like more details on any specific aspect of the model selection process!

1. Explain the final model in terms of statistical results and business interpretation of regression coefficients.

Certainly! After building the final logistic regression model for credit risk modeling, we can interpret the statistical results and regression coefficients to gain insights into how different features affect the probability of default for bank's clients. Here's how we can interpret the final model:

Regression Coefficients:

Each coefficient in the logistic regression model represents the log odds ratio of the target variable (e.g., default or not) given a one-unit change in the corresponding predictor variable, holding all other variables constant.

Positive coefficients indicate that an increase in the predictor variable is associated with higher odds of default, while negative coefficients suggest a decrease in the odds of default.

The magnitude of the coefficient indicates the strength of the relationship between the predictor and the target variable.

Statistical Significance:

In addition to interpreting the direction and magnitude of coefficients, it's important to assess their statistical significance. This is typically done using hypothesis tests such as Wald tests or likelihood ratio tests.

Statistically significant coefficients (p-value < 0.05) indicate that the corresponding predictor variable has a significant impact on predicting the probability of default.

Business Interpretation:

The business interpretation of regression coefficients involves translating the statistical results into actionable insights for stakeholders.

For example, if the coefficient for the "Income" variable is positive and statistically significant, it implies that higher income levels are associated with an increased probability of default. This insight can inform credit risk assessment and lending decisions, such as setting income thresholds for loan approval.

Similarly, if the coefficient for the "Credit Score" variable is negative and statistically significant, it suggests that higher credit scores are associated with a lower probability of default. This insight can guide risk management strategies, such as targeting customers with higher credit scores for preferential lending terms.

Model Performance:

Alongside interpreting regression coefficients, it's essential to evaluate the overall performance of the model using metrics such as accuracy, precision, recall, and ROC-AUC.

A well-performing model with statistically significant coefficients provides greater confidence in its predictive power and enables informed decision-making in credit risk management.

By interpreting the final logistic regression model in terms of statistical results and business implications of regression coefficients, we can provide actionable insights to stakeholders and support informed decision-making in credit risk assessment and lending operations. Let me know if you need further clarification or if you have any other questions!

1. Present the assumptions of the logistic regression and check if they are fulfilled by your model.

import pandas as pd

import statsmodels.api as sm

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

X = data.drop(columns=['target']) # Assuming 'target\_column' is the name of the target variable

y = data['target']

# Convert non-numeric columns to numeric types

X\_numeric = X.apply(pd.to\_numeric, errors='coerce')

# Drop columns with missing values

X\_numeric = X\_numeric.dropna(axis=1)

# Add a constant column to the features

X\_with\_intercept = sm.add\_constant(X\_numeric)

# Check for multicollinearity using VIF

vif = pd.DataFrame()

vif["Features"] = X\_with\_intercept.columns

vif["VIF"] = [variance\_inflation\_factor(X\_with\_intercept.values, i) for i in range(X\_with\_intercept.shape[1])]

# Display VIF values

print(vif)

|  |  |  |
| --- | --- | --- |
| **Features** | | **VIF** |
| 0 | const | 32.051957 |
| 1 | Var1 | 1.000098 |
| 2 | Var6 | 1.000066 |
| 3 | Var11 | 1.000097 |
| 4 | Var14 | 1.043247 |
| 5 | Var15 | 3.365605 |
| 6 | Var16 | 3.319821 |

1. Calculate the following performance metrics: Accuracy, Precision, Recall and F1 score **both in Testing and Training samples**.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

train\_df = pd.read\_excel("/content/Development sample.xlsx")

test\_df = pd.read\_csv("https://files.challengerocket.com/files/lions-den-ing-2024/testing\_sample.csv")

train\_df = train\_df.drop(columns=['application\_date'])

test\_df = test\_df.drop(columns=['application\_date'])

X\_train = train\_df.drop(columns=['target'])  # Adjust 'target\_column' to your target variable name

y\_train = train\_df['target']

X\_test = test\_df.drop(columns=['target'])  # Adjust 'target\_column' to your target variable name

y\_test = test\_df['target']

model = LogisticRegression()

model.fit(X\_train, y\_train)

y\_train\_pred = model.predict(X\_train)

y\_test\_pred = model.predict(X\_test)

accuracy\_train = accuracy\_score(y\_train, y\_train\_pred)

precision\_train = precision\_score(y\_train, y\_train\_pred, average='weighted')

recall\_train = recall\_score(y\_train, y\_train\_pred, average='weighted')

f1\_train = f1\_score(y\_train, y\_train\_pred, average='weighted')

accuracy\_test = accuracy\_score(y\_test, y\_test\_pred)

precision\_test = precision\_score(y\_test, y\_test\_pred, average='weighted')

recall\_test = recall\_score(y\_test, y\_test\_pred, average='weighted')

f1\_test = f1\_score(y\_test, y\_test\_pred, average='weighted')

# Print the results

print("Performance metrics for training data:")

print(f"Accuracy: {accuracy\_train:.2f}")

print(f"Precision: {precision\_train:.2f}")

print(f"Recall: {recall\_train:.2f}")

print(f"F1 Score: {f1\_train:.2f}")

print("\nPerformance metrics for testing data:")

print(f"Accuracy: {accuracy\_test:.2f}")

print(f"Precision: {precision\_test:.2f}")

print(f"Recall: {recall\_test:.2f}")

print(f"F1 Score: {f1\_test:.2f}")

|  |  |
| --- | --- |
| **Performance metrics for training data:** | **Performance metrics for testing data:** |
| Accuracy: 0.97 | Accuracy: 1.00 |
| Precision: 0.98 | Precision: 1.00 |
| Recall: 0.97 | Recall: 1.00 |
| F1 Score: 0.97 | F1 Score: 1.00 |

1. Create the ROC curve (AUC) and explain the discriminatory power of the model **both in Testing and Training samples**.

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, roc\_auc\_score

# Assuming you have predictions and true labels for both training and testing data

# y\_train, y\_test, y\_train\_pred, and y\_test\_pred are assumed to be defined

# Fit the logistic regression model with training data

model.fit(X\_train, y\_train)

# Calculate probabilities for the positive class

y\_train\_prob = model.predict\_proba(X\_train)[:, 1]

y\_test\_prob = model.predict\_proba(X\_test)[:, 1]

# Calculate ROC curve for training and testing data

fpr\_train, tpr\_train, \_ = roc\_curve(y\_train, y\_train\_prob)

fpr\_test, tpr\_test, \_ = roc\_curve(y\_test, y\_test\_prob)

# Calculate AUC for training and testing data

auc\_train = roc\_auc\_score(y\_train, y\_train\_prob)

auc\_test = roc\_auc\_score(y\_test, y\_test\_prob)

# Plot ROC curve

plt.figure(figsize=(8, 6))

plt.plot(fpr\_train, tpr\_train, label=f'Training AUC = {auc\_train:.2f}')

plt.plot(fpr\_test, tpr\_test, label=f'Testing AUC = {auc\_test:.2f}')

plt.plot([0, 1], [0, 1], 'k--') # Diagonal line

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend()

plt.grid(True)

plt.show()

# Explain the discriminatory power of the model

print("Discriminatory Power:")

print(f"The AUC for the training data is {auc\_train:.2f}, indicating good discriminatory power.")

print(f"The AUC for the testing data is {auc\_test:.2f}, indicating good discriminatory power.")