Import Libraries

```
import numpy as np                                              # Basic libraries of python for numeric and dataframe computations
import pandas as pd
import statsmodels.api as sm
import scipy.stats as stats
import matplotlib.pyplot as plt                                 # Basic library for data visualization
import seaborn as sns                                           # Slightly advanced library for data visualization

from sklearn.preprocessing import LabelEncoder                  # Used to encode categorical variable
from sklearn.preprocessing import StandardScaler                # StandardScaler (mean=0, std=1)
from sklearn.model_selection import train_test_split            # Used to split the data into train and test sets.
from sklearn import metrics                                     # Metrics to evaluate the model

from statsmodels.stats.outliers_influence import variance_inflation_factor #Multicollinearity assesment
from scipy.stats import mannwhitneyu, chi2_contingency           # Used in feature selection
from itertools import combinations
from itertools import product

from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix, classification_report, accuracy_score, precision_recall_curve
import matplotlib.pyplot as plt
```

Exploring the Data

Loading Data

```
# Load Dataset
# If you have a local file: df = pd.read_csv("path_to_file.csv")
# For illustration, assuming dataset is similar to UCI German Credit Data
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/german.data"
columns = [
    "Account_status", "Duration", "Credit_history", "Purpose", "Credit_amount",
    "Savings_bonds", "Present_employment_since", "Installment_rate", "Personal_status_sex",
    "Other_debtors_guarantors", "Present_residence_since", "Property", "Age",
    "Other_installment_plans", "Housing", "Number_existing_credits", "Job",
    "People_liable", "Telephone", "Foreign_worker", "Credit_risk"
]
df = pd.read_csv(url, sep='\s+', header=None, names=columns)
df.head(10)
```

	Account_status	Duration	Credit_history	Purpose	Credit_amount	Savings_bonds	Present_employment_since	Installment_rate	Personal_
0	A11	6	A34	A43	1169	A65	A75	4	
1	A12	48	A32	A43	5951	A61	A73	2	
2	A14	12	A34	A46	2096	A61	A74	2	
3	A11	42	A32	A42	7882	A61	A74	2	
4	A11	24	A33	A40	4870	A61	A73	3	
5	A14	36	A32	A46	9055	A65	A73	2	
6	A14	24	A32	A42	2835	A63	A75	3	
7	A12	36	A32	A41	6948	A61	A73	2	
8	A14	12	A32	A43	3059	A64	A74	2	
9	A12	30	A34	A40	5234	A61	A71	4	

10 rows × 21 columns

```
df.info()
df['Credit_risk'].value_counts(normalize=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Account_status           1000 non-null   object
 1   Duration                 1000 non-null   int64 
 2   Credit_history           1000 non-null   object
 3   Purpose                  1000 non-null   object
 4   Credit_amount            1000 non-null   int64 
 5   Savings_bonds            1000 non-null   object
 6   Present_employment_since  1000 non-null   object
 7   Installment_rate         1000 non-null   int64 
 8   Personal_status_sex      1000 non-null   object
 9   Other_debtors_guarantors  1000 non-null   object
 10  Present_residence_since  1000 non-null   int64 
 11  Property                 1000 non-null   object
 12  Age                      1000 non-null   int64 
 13  Other_installment_plans  1000 non-null   object
 14  Housing                  1000 non-null   object
 15  Number_existing_credits  1000 non-null   int64 
 16  Job                      1000 non-null   object
 17  People_liable            1000 non-null   int64 
 18  Telephone                1000 non-null   object
 19  Foreign_worker           1000 non-null   object
 20  Credit_risk              1000 non-null   int64 
dtypes: int64(8), object(13)
memory usage: 164.2+ KB
```

Credit_risk	proportion
1	0.7
2	0.3

dtype: float64

- 1000 entries, for all the columns. No missing values
- The German Credit Data entail:

1. Numerical Data (7)
2. Categorical Data (13)

We are working with an imbalanced dataset. Bad : Good credit is equivalent to 3:7

Map target variable

```
df['Credit_risk'] = df['Credit_risk'].map({1: 0, 2: 1})  # 1 = bad, 0 = good
```

Encoding Categorical Variables

For categorical variables: binary, nominal, and ordinal variables are present within our dataset.

For ordinal variables we apply label encoding.

For binary and nominal variable, we apply one-hot encoding. This avoids implying any kind of rank or order.

For property and other installment plans following further discussuin, we agreed to treat the variables as nominal as there's no natural or objective ordering to it.

```
#Identify categorical features
ordinal_features = {
    'Account_status': ["no checking", "<0", "0<=...<200", ">=200"],
    'Credit_history': [
        "no credits/all paid", "critical", "delayed", "existing paid", "all paid"
    ],
    'Savings_bonds': ["unknown", "<100", "100<=...<500", "500<=...<1000", ">=1000"],
    'Present_employment_since': ["unemployed", "<1", "1<=...<4", "4<=...<7", ">=7"],
    'Housing': ["for free", "rent", "own"],
    'Job': [
        "unemployed/unskilled-nonresident", "unskilled-resident",
        "skilled", "management/self-employed"
    ],
    'Other_debtors_guarantors': ["none", "co-applicant", "guarantor"]
```

```
}

#Standardize ordinal categories
ordinal_mappings = {
    'Account_status': {
        'A14': 0, 'A11': 1, 'A12': 2, 'A13': 3
    },
    'Credit_history': {
        'A30': 0, 'A34': 1, 'A33': 2, 'A32': 3, 'A31': 4
    },
    'Savings_bonds': {
        'A65': 0, 'A61': 1, 'A62': 2, 'A63': 3, 'A64': 4
    },
    'Present_employment_since': {
        'A71': 0, 'A72': 1, 'A73': 2, 'A74': 3, 'A75': 4
    },
    'Housing': {
        'A151': 0, 'A152': 1, 'A153': 2
    },
    'Job': {
        'A171': 0, 'A172': 1, 'A173': 2, 'A174': 3
    }
    ,
    'Other_debtors_guarantors': {
        'A101': 0, 'A102': 1, 'A103': 2
    }
}

# Apply ordinal encoding
df_encoded = df.copy()
for col, mapping in ordinal_mappings.items():
    df_encoded[col] = df_encoded[col].map(mapping)

# Step 5: One-Hot Encode the remaining categorical columns
categorical_cols = [
    col for col in df.columns
    if df[col].dtype == 'object' and col not in ordinal_mappings
]

df_encoded = pd.get_dummies(df_encoded, columns=categorical_cols, drop_first=True,dtype=int)

# Step 6: Final check
print(df_encoded.head())
print("\nFinal shape of dataset:", df_encoded.shape)
df_encoded.info()
```

```
   Property_A123  Property_A124  Other_installment_plans_A142  \
0              0              0                             0
1              0              0                             0
2              0              0                             0
3              0              0                             0
4              0              1                             0
```

```
 12  Job                          1000 non-null   int64
 13  People_liable                1000 non-null   int64
 14  Credit_risk                  1000 non-null   int64
 15  Purpose_A41                  1000 non-null   int64
 16  Purpose_A410                 1000 non-null   int64
 17  Purpose_A42                  1000 non-null   int64
 18  Purpose_A43                  1000 non-null   int64
 19  Purpose_A44                  1000 non-null   int64
 20  Purpose_A45                  1000 non-null   int64
 21  Purpose_A46                  1000 non-null   int64
 22  Purpose_A48                  1000 non-null   int64
 23  Purpose_A49                  1000 non-null   int64
 24  Personal_status_sex_A92      1000 non-null   int64
 25  Personal_status_sex_A93      1000 non-null   int64
 26  Personal_status_sex_A94      1000 non-null   int64
 27  Property_A122                1000 non-null   int64
 28  Property_A123                1000 non-null   int64
 29  Property_A124                1000 non-null   int64
 30  Other_installment_plans_A142  1000 non-null   int64
 31  Other_installment_plans_A143  1000 non-null   int64
 32  Telephone_A192               1000 non-null   int64
 33  Foreign_worker_A202          1000 non-null   int64
dtypes: int64(34)
memory usage: 265.8 KB
```

Train-Test Split

```
# Split features and target
X = df_encoded.drop("Credit_risk", axis=1)  #'Credit_risk' is the target variable
y = df_encoded["Credit_risk"]

# Split into train and test sets (e.g. 80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Normalization

Features are on different scales for instance credit amount, age, number of credit cards.

```
# Get numeric columns with more than 2 unique values (excludes one-hot encoded columns)
numeric_cols = [col for col in X_train.select_dtypes(include=['int64', 'float64']).columns if X_train[col].nunique() > 2]

scaler = StandardScaler()

# Fit on training data and transform both train and test
X_train[numeric_cols] = scaler.fit_transform(X_train[numeric_cols])
X_test[numeric_cols] = scaler.transform(X_test[numeric_cols])

X_train[numeric_cols].describe().T
```

	count	mean	std	min	25%	50%	75%	max
Account_status	800.0	4.218847e-17	1.000626	-1.030445	-1.030445	0.014366	1.059177	2.103989
Duration	800.0	9.769963e-17	1.000626	-1.448750	-0.770774	-0.262292	0.246190	3.297082
Credit_history	800.0	-2.087219e-16	1.000626	-2.111986	-1.169135	0.716567	0.716567	1.659418
Credit_amount	800.0	-1.776357e-17	1.000626	-1.073974	-0.683830	-0.354796	0.274096	5.200792
Savings_bonds	800.0	1.332268e-16	1.000626	-1.223166	-0.205980	-0.205980	-0.205980	2.845578
Present_employment_since	800.0	6.772360e-17	1.000626	-1.954297	-0.310304	-0.310304	1.333690	1.333690
Installment_rate	800.0	1.465494e-16	1.000626	-1.751413	-0.860109	0.031196	0.922500	0.922500
Other_debtors_guarantors	800.0	4.440892e-18	1.000626	-0.316463	-0.316463	-0.316463	-0.316463	3.702116
Present_residence_since	800.0	2.664535e-17	1.000626	-1.671440	-0.766124	0.139192	1.044509	1.044509
Age	800.0	-3.075318e-16	1.000626	-1.451955	-0.750474	-0.224364	0.564801	3.458408
Housing	800.0	1.176836e-16	1.000626	-1.766639	0.125344	0.125344	0.125344	2.017327
Number_existing_credits	800.0	2.242651e-16	1.000626	-0.710931	-0.710931	-0.710931	1.017777	4.475195
Job	800.0	1.398881e-16	1.000626	-2.951471	0.107048	0.107048	0.107048	1.636308

The mean is 0 and standard deviation is 1 for the train dataset

Multi Collinearity Check

VIF check

```
# Multicollinearity check only for numerical and dummy variables in X_train
# Drop any constant columns (if any)
X_vif = X_train.copy()
# Explicitly select only numeric columns
X_vif = X_vif.select_dtypes(include=np.number)  # Select only numeric columns
X_vif = X_vif.loc[:, X_vif.std() > 0]

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data["Feature"] = X_vif.columns
vif_data["VIF"] = [variance_inflation_factor(X_vif.values, i) for i in range(X_vif.shape[1])]

# View results
vif_data.sort_values(by="VIF", ascending=False)
```

	Feature	VIF
13	People_liable	10.703134
24	Personal_status_sex_A93	9.774321
30	Other_installment_plans_A143	6.003692
23	Personal_status_sex_A92	5.250265
27	Property_A123	2.490061
3	Credit_amount	2.450078
28	Property_A124	2.389480
31	Telephone_A192	2.252183
17	Purpose_A43	2.246161
25	Personal_status_sex_A94	2.231143
26	Property_A122	1.976687
1	Duration	1.949752
16	Purpose_A42	1.740164
14	Purpose_A41	1.584746
10	Housing	1.569991
11	Number_existing_credits	1.479305
22	Purpose_A49	1.461552
2	Credit_history	1.451736
12	Job	1.401117
9	Age	1.353027
29	Other_installment_plans_A142	1.345850
6	Installment_rate	1.339059
5	Present_employment_since	1.244693
20	Purpose_A46	1.240105
8	Present_residence_since	1.227373
15	Purpose_A410	1.152310
32	Foreign_worker_A202	1.130287
7	Other_debtors_guarantors	1.124920
19	Purpose_A45	1.115821
18	Purpose_A44	1.081482
21	Purpose_A48	1.075086
0	Account_status	1.064646
4	Savings_bonds	1.042924

Low Multicollinearity between the numerical variables.

Feature Selection

We employ two statistical inference techniques, such as the Mann–Whitney–Wilcoxon and Pearson Chi-squared independence tests, to infer the factors that influence credit risk from a small random sample of customers from the German Credit Data. The same techniques used in Portuguese banking institution.

```
# Recombine X_train with y_train for feature selection
train_data = X_train.copy()
train_data["Credit_risk"] = y_train

# Identify categorical (encoded as dummy) and numeric features
categorical_features = [col for col in train_data.columns if col not in numeric_cols + ["Credit_risk"]]

#Mann–Whitney–Wilcoxon
numerical_results = []

for col in numeric_cols:
    group_0 = train_data[train_data["Credit_risk"] == 0][col]
    group_1 = train_data[train_data["Credit_risk"] == 1][col]

    stat, p = mannwhitneyu(group_0, group_1, alternative="two-sided")
    numerical_results.append({"Feature": col, "P-Value": p})

numerical_results_df = pd.DataFrame(numerical_results).sort_values("P-Value")
print(numerical_results_df)

#Chi-Squared Test for Categorical Variables
chi2_results = []

for col in categorical_features:
    contingency_table = pd.crosstab(train_data[col], train_data["Credit_risk"])
    stat, p, dof, expected = chi2_contingency(contingency_table)
    chi2_results.append({"Feature": col, "P-Value": p})

chi2_results_df = pd.DataFrame(chi2_results).sort_values("P-Value")
print(chi2_results_df)

selected_numerical = numerical_results_df[numerical_results_df["P-Value"] < 0.05]["Feature"].tolist()
selected_categorical = chi2_results_df[chi2_results_df["P-Value"] < 0.05]["Feature"].tolist()

final_selected_features = selected_numerical + selected_categorical
print(final_selected_features)

# Create a DataFrame with only the selected features
X_train_selected = X_train[final_selected_features]
```

```
                   Feature       P-Value
0           Account_status  2.597220e-09
1                 Duration  2.402491e-08
9                      Age  9.966921e-06
5   Present_employment_since  3.371988e-04
2           Credit_history  2.222794e-03
3            Credit_amount  4.915690e-03
12                     Job  8.464107e-02
6         Installment_rate  2.039964e-01
11  Number_existing_credits  3.394051e-01
7   Other_debtors_guarantors  5.445353e-01
10                 Housing  5.569557e-01
8    Present_residence_since  8.044292e-01
4            Savings_bonds  8.584650e-01
                       Feature   P-Value
4                  Purpose_A43  0.000427
15               Property_A124  0.000442
17  Other_installment_plans_A143  0.002423
11      Personal_status_sex_A93  0.008266
10      Personal_status_sex_A92  0.013130
7                  Purpose_A46  0.019741
1                  Purpose_A41  0.026872
19          Foreign_worker_A202  0.038519
2                 Purpose_A410  0.302201
9                  Purpose_A49  0.339039
16  Other_installment_plans_A142  0.461109
18              Telephone_A192  0.461157
14               Property_A123  0.466546
```

```
8                    Purpose_A48  0.614471
5                    Purpose_A44  0.901910
12       Personal_status_sex_A94  0.959197
3                    Purpose_A42  0.962680
0                  People_liable  0.974812
13                 Property_A122  1.000000
6                    Purpose_A45  1.000000
['Account_status', 'Duration', 'Age', 'Present_employment_since', 'Credit_history', 'Credit_amount', 'Purpose_A43', 'Property_A124', 'Ot
```

Feature Selection: From Mann Whitney U Test:

- Account_status: Even though it is not a 1-to-1 equivalent of the variable "Salary" from the Protuguese Bank Data Study, we feel it is close enough since both of them try to account for the account in which Salary is being received.
- Duration: Significant + Equivalent of "Term in Portugese Study
- Age: Present in both datasets
- Present_employment_since: Significant, but not in Portuguese Study
- Credit_history: Significant + Equivalent to "Other Credit" despite difference since both look at if the client has taken some other credits. The german dataset goes further in-depth by looking at the repayment habits
- Credit_Amount: Significant + Equivalent to "Capital Outstanding in Portuguese data.
- Job: Signficant, but not in Portuguese Study.

  From Pearson Chi-Squared Test:
- Purpose: Questionable significance (A43, A46, A41), but not in Portuguese Study
- Property: Questionable significance (A124), but not in Portuguese Study
- Other_installment_plans: Significant + Equivalent to "Other Credit" but solely for outside entities.
- Personal_status_sex: Significant + Equivalent to "Sex" & "Marital Status", however the two are combined.
- Foreign_worker: Significant, but not in Portuguese Study

  In conclusion, the variable we will use are:
- As a result of the Mann Whitney U Test:
  - Account_Status
  - Duration
  - Age
  - Credit_history
  - Credit Amount
- As a result of the Pearson Chi Squared Test:
  - Other_installement_plans
  - Personal_status_sex

Logistic Regression Model & Wald Test

```
# Select the final features in both train and test data
X_train_selected = X_train[final_selected_features]
X_test_selected = X_test[final_selected_features]
#
X_train_selected = pd.get_dummies(X_train_selected,
                                  columns=[col for col in selected_categorical if X_train_selected[col].dtype == 'object'],
                                  drop_first=True)
X_train_selected.head()
```

	Account_status	Duration	Age	Present_employment_since	Credit_history	Credit_amount	Purpose_A43	Property_A124	Other_inst
29	0.014366	3.297082	2.406187	1.333690	-0.226284	1.199912	0	1	
535	2.103989	-0.008051	-0.224364	-1.132300	-1.169135	-0.359630	0	0	
695	-1.030445	-1.279256	1.266282	-0.310304	0.716567	-0.733547	0	0	
557	-1.030445	-0.008051	-0.575104	-0.310304	-2.111986	0.567050	0	0	
836	-1.030445	-0.770774	-1.276585	-0.310304	0.716567	-0.854388	1	0	

Next steps: Generate code with X_train_selected | View recommended plots | New interactive sheet

First Model

```
# Add constant to training data and fit the model
logit_model = sm.Logit(y_train, X_train).fit()
print(logit_model.summary())
#Null Deviance
null_deviance = -2 * logit_model.llnull
print("Null Deviance:", null_deviance)
#Residual Deviance
residual_deviance = -2 * logit_model.llf
print("Residual Deviance:", residual_deviance)
print("AIC:", logit_model.aic)
```

```
Optimization terminated successfully.
         Current function value: 0.503885
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:            Credit_risk   No. Observations:                  800
Model:                          Logit   Df Residuals:                      767
Method:                           MLE   Df Model:                           32
Date:                Mon, 05 May 2025   Pseudo R-squ.:                  0.1766
Time:                        19:29:20   Log-Likelihood:                -403.11
converged:                       True   LL-Null:                       -489.54
Covariance Type:            nonrobust   LLR p-value:                 3.018e-21
================================================================================================
                                   coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Account_status                   0.3690      0.089      4.135      0.000       0.194       0.544
Duration                         0.2749      0.114      2.406      0.016       0.051       0.499
Credit_history                   0.2244      0.104      2.148      0.032       0.020       0.429
Credit_amount                    0.3151      0.127      2.488      0.013       0.067       0.563
Savings_bonds                   -0.0772      0.093     -0.829      0.407      -0.260       0.105
Present_employment_since        -0.2010      0.097     -2.079      0.038      -0.391      -0.011
Installment_rate                 0.3350      0.102      3.295      0.001       0.136       0.534
Other_debtors_guarantors        -0.0690      0.096     -0.717      0.473      -0.258       0.120
Present_residence_since          0.1031      0.096      1.079      0.281      -0.084       0.291
Age                             -0.3586      0.109     -3.293      0.001      -0.572      -0.145
Housing                         -0.1718      0.105     -1.640      0.101      -0.377       0.034
Number_existing_credits          0.1140      0.108      1.056      0.291      -0.098       0.326
Job                              0.1311      0.102      1.279      0.201      -0.070       0.332
People_liable                    0.4397      0.238      1.848      0.065      -0.027       0.906
Purpose_A41                     -1.5878      0.366     -4.340      0.000      -2.305      -0.871
Purpose_A410                    -0.7683      0.781     -0.983      0.325      -2.299       0.763
Purpose_A42                     -0.7178      0.264     -2.718      0.007      -1.235      -0.200
Purpose_A43                     -1.1061      0.251     -4.409      0.000      -1.598      -0.614
Purpose_A44                     -0.3783      0.732     -0.517      0.605      -1.813       1.056
Purpose_A45                     -0.2672      0.549     -0.487      0.626      -1.342       0.808
Purpose_A46                      0.0550      0.417      0.132      0.895      -0.761       0.872
Purpose_A48                     -1.6307      1.155     -1.412      0.158      -3.894       0.633
Purpose_A49                     -0.4835      0.317     -1.524      0.127      -1.105       0.138
Personal_status_sex_A92         -0.3212      0.334     -0.961      0.336      -0.976       0.334
Personal_status_sex_A93         -0.8920      0.352     -2.533      0.011      -1.582      -0.202
Personal_status_sex_A94         -0.2257      0.418     -0.540      0.589      -1.045       0.594
Property_A122                    0.4771      0.259      1.842      0.065      -0.030       0.985
Property_A123                    0.4129      0.240      1.721      0.085      -0.057       0.883
Property_A124                    1.1414      0.342      3.334      0.001       0.470       1.812
Other_installment_plans_A142    -0.1776      0.435     -0.409      0.683      -1.029       0.674
Other_installment_plans_A143    -0.6062      0.222     -2.728      0.006      -1.042      -0.171
Telephone_A192                  -0.4324      0.207     -2.089      0.037      -0.838      -0.027
Foreign_worker_A202             -1.1313      0.684     -1.654      0.098      -2.472       0.209
================================================================================================
Null Deviance: 979.0715314237384
Residual Deviance: 806.2158009978755
AIC: 872.2158009978755
```

Among all the variables suggested by exploratory analysis, only a few were found to be significant at p-value 5%. The significant variables:

Account_status, Duration, Age, Purpose_A41,Purpose_A42, Purpose_A43, Purpose_A124, and Other_installment_plans_A143.

Only the variable Telephone, that was not suggested by the exploratory analysis to be relevant, is now found to be relevant too.

```
# Extract coefficient table with p-values
summary_table = logit_model.summary2().tables[1]

# Filter variables that are significant at 5% level (p < 0.05)
significant_vars = summary_table[summary_table['P>|z|'] < 0.05]

# Print results
print("Significant variables at 5% level based on Wald test:")
print(significant_vars[['Coef.', 'Std.Err.', 'z', 'P>|z|']])
```

```
Significant variables at 5% level based on Wald test:
                                Coef.  Std.Err.         z     P>|z|
Account_status               0.368983  0.089226  4.135400  0.000035
Duration                     0.274881  0.114245  2.406075  0.016125
Credit_history               0.224444  0.104497  2.147847  0.031726
Credit_amount                0.315094  0.126668  2.487548  0.012863
Present_employment_since    -0.201048  0.096725 -2.078545  0.037659
Installment_rate             0.334960  0.101649  3.295246  0.000983
Age                         -0.358603  0.108884 -3.293435  0.000990
Purpose_A41                 -1.587757  0.365807 -4.340418  0.000014
Purpose_A42                 -0.717768  0.264116 -2.717626  0.006575
Purpose_A43                 -1.106118  0.250882 -4.408917  0.000010
Personal_status_sex_A93     -0.891965  0.352171 -2.532763  0.011317
Property_A124                1.141361  0.342366  3.333744  0.000857
Other_installment_plans_A143 -0.606165  0.222184 -2.728207  0.006368
Telephone_A192              -0.432433  0.206965 -2.089399  0.036672
```

The variables for which the null hypothesis of theWald test is rejected, at a significance level of 5%, and therefore are significant covariables in the model, are as indicated above.

```
# Add constant to training data and fit the model
X_train_const = sm.add_constant(X_train_selected)
logit_model = sm.Logit(y_train, X_train_const).fit()
print(logit_model.summary())
#Null Deviance
null_deviance = -2 * logit_model.llnull
print("Null Deviance:", null_deviance)
#Residual Deviance
residual_deviance = -2 * logit_model.llf
print("Residual Deviance:", residual_deviance)
print("AIC:", logit_model.aic)
```

```
Optimization terminated successfully.
         Current function value: 0.527394
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:           Credit_risk   No. Observations:                  800
Model:                         Logit   Df Residuals:                      785
Method:                          MLE   Df Model:                           14
Date:               Mon, 05 May 2025   Pseudo R-squ.:                  0.1381
Time:                       19:30:18   Log-Likelihood:                -421.92
converged:                      True   LL-Null:                       -489.54
Covariance Type:           nonrobust   LLR p-value:                 6.248e-22
================================================================================================
                                   coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                           -0.0623      0.301     -0.207      0.836      -0.653       0.528
Account_status                   0.3313      0.085      3.891      0.000       0.164       0.498
Duration                         0.3390      0.107      3.183      0.001       0.130       0.548
Age                             -0.3408      0.099     -3.450      0.001      -0.534      -0.147
Present_employment_since        -0.1401      0.090     -1.554      0.120      -0.317       0.037
Credit_history                   0.1659      0.087      1.916      0.055      -0.004       0.336
Credit_amount                    0.1494      0.107      1.395      0.163      -0.061       0.359
Purpose_A43                     -0.7829      0.209     -3.747      0.000      -1.192      -0.373
Property_A124                    0.7245      0.247      2.935      0.003       0.241       1.208
Other_installment_plans_A143    -0.5159      0.210     -2.462      0.014      -0.927      -0.105
Personal_status_sex_A93         -0.4958      0.264     -1.882      0.060      -1.012       0.021
Personal_status_sex_A92         -0.0618      0.266     -0.233      0.816      -0.582       0.459
Purpose_A46                      0.3775      0.390      0.968      0.333      -0.387       1.142
Purpose_A41                     -1.0933      0.324     -3.377      0.001      -1.728      -0.459
Foreign_worker_A202             -1.1995      0.668     -1.797      0.072      -2.508       0.109
```

```
==============================================================================================
Null Deviance: 979.0715314237384
Residual Deviance: 843.8310712423104
AIC: 873.8310712423104
```

Build up on the factors that were significant from the exploratory analysis.

Among all the variables suggested by exploratory analysis, only a few were found to be significant at p-value 5%.

Generate a slider using jupyter widgets Close

```
# Extract coefficient table with p-values
summary_table = logit_model.summary2().tables[1]

# Filter variables that are significant at 5% level (p < 0.05)
significant_vars = summary_table[summary_table['P>|z|'] < 0.05]

# Print results
print("Significant variables at 5% level based on Wald test:")
print(significant_vars[['Coef.', 'Std.Err.', 'z', 'P>|z|']])
```

```
Significant variables at 5% level based on Wald test:
                                Coef.  Std.Err.         z     P>|z|
Account_status               0.331346  0.085164  3.890664  0.000100
Duration                     0.339009  0.106520  3.182573  0.001460
Age                         -0.340752  0.098782 -3.449529  0.000562
Purpose_A43                 -0.782922  0.208939 -3.747128  0.000179
Property_A124                0.724537  0.246893  2.934616  0.003340
Other_installment_plans_A143 -0.515924  0.209513 -2.462487  0.013798
Purpose_A41                 -1.093338  0.323789 -3.376699  0.000734
```

The final model identifies key covariates that significantly influence credit risk, as determined through the Wald test, with the model specification guided by the Mann-Whitney-Wilcoxon and Pearson Chi-Square variable selection techniques.

Key Significant Predictors of Default:

Account_status, Duration & Age (Qualitative Factors)

Purpose_A43 (Television/Radio) and Purpose_A41 (Used Car): These loan purposes fall under consumption-related borrowing, often not backed by income-generating assets. This suggests that borrowers seeking credit for depreciating goods have a higher likelihood of default, consistent with consumer overextension theory.

Property_A124 (No property): Borrowers who do not own property lack tangible collateral, which not only weakens their bargaining position with lenders but also reduces recovery prospects in case of default. This aligns with increased credit risk.

Other_Installment_Plans_A143 (None): Surprisingly, individuals without any existing installment plans (i.e., no current borrowing track record) are flagged as riskier. This may reflect thin credit files, a known concern in retail lending where lack of past credit data limits accurate assessment of repayment behavior.

***Question:** *For borrowers with other installment plans through banks or stores — does this reflect high financial leverage? A more granular analysis incorporating DTI (debt-to-income) ratios and payment behavior across different credit types could further clarify this relationship.

This model is based on the selected variables on Mann-Whitney Wilcoxon Test & Pearson-Chi Squared Variables.

Although this reduced model has a slightly higher AIC than the full model, we prioritize it due to:

- Greater parsimony: Fewer, more interpretable variables
- Stronger statistical significance across selected covariates
- More stable estimation, with reduced multicollinearity
- Meaningful insights aligned with economic theory and credit risk frameworks

In line with the Portuguese paper, we favor statistical robustness and interpretability over mere goodness-of-fit. This model serves as a reliable foundation for policy and credit decision-making..

Interaction Between Variables

We considered interactions between the quantitative and qualitative variables present in model 2.

```
# --- Define quantitative and qualitative variables from Model 2 ---
quantitative_vars = ['Duration', 'Age']
qualitative_vars = ['Account_status', 'Purpose_A41', 'Purpose_A43',
                    'Property_A124', 'Other_installment_plans_A143']
```

```
# --- Generate interaction terms manually ---
interaction_terms = []
for q_var, cat_var in product(quantitative_vars, qualitative_vars):
    interaction_name = f"{q_var}_x_{cat_var}"
    X_train[interaction_name] = X_train[q_var] * X_train[cat_var]
    interaction_terms.append(interaction_name)

# --- Refit model with interaction terms ---
X_model3 = sm.add_constant(X_train[quantitative_vars + qualitative_vars + interaction_terms])
model3 = sm.Logit(y_train, X_model3).fit()

# --- Print model summary and deviance ---
print(model3.summary())

null_deviance = -2 * model3.llnull
residual_deviance = -2 * model3.llf
print("Null Deviance:", null_deviance)
print("Residual Deviance:", residual_deviance)
print("AIC:", model3.aic)
```

```
Optimization terminated successfully.
         Current function value: 0.535325
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:           Credit_risk   No. Observations:                  800
Model:                         Logit   Df Residuals:                      782
Method:                          MLE   Df Model:                           17
Date:               Mon, 05 May 2025   Pseudo R-squ.:                  0.1252
Time:                       19:50:09   Log-Likelihood:                -428.26
converged:                      True   LL-Null:                       -489.54
Covariance Type:           nonrobust   LLR p-value:                 5.028e-18
==========================================================================================================
                                              coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------------------------------
const                                      -0.3792      0.205     -1.854      0.064      -0.780       0.022
Duration                                    0.3397      0.206      1.651      0.099      -0.064       0.743
Age                                        -0.6046      0.235     -2.577      0.010      -1.064      -0.145
Account_status                              0.3480      0.086      4.067      0.000       0.180       0.516
Purpose_A41                                -1.2506      0.395     -3.169      0.002      -2.024      -0.477
Purpose_A43                                -0.9043      0.227     -3.980      0.000      -1.350      -0.459
Property_A124                               0.6699      0.250      2.681      0.007       0.180       1.160
Other_installment_plans_A143               -0.5007      0.213     -2.354      0.019      -0.918      -0.084
Duration_x_Account_status                   0.0568      0.086      0.664      0.507      -0.111       0.225
Duration_x_Purpose_A41                      0.2597      0.350      0.741      0.459      -0.427       0.946
Duration_x_Purpose_A43                     -0.0113      0.191     -0.059      0.953      -0.385       0.362
Duration_x_Property_A124                   -0.1273      0.195     -0.652      0.515      -0.510       0.256
Duration_x_Other_installment_plans_A143     0.0782      0.206      0.379      0.704      -0.326       0.482
Age_x_Account_status                        0.0024      0.093      0.026      0.979      -0.179       0.184
Age_x_Purpose_A41                           0.1389      0.302      0.459      0.646      -0.454       0.732
Age_x_Purpose_A43                          -0.2328      0.267     -0.872      0.383      -0.756       0.290
Age_x_Property_A124                         0.4960      0.225      2.206      0.027       0.055       0.937
Age_x_Other_installment_plans_A143          0.0768      0.229      0.336      0.737      -0.371       0.525
==========================================================================================================
Null Deviance: 979.0715314237384
Residual Deviance: 856.5202749149821
AIC: 892.5202749149821
```

After many experiences, the only interaction that always came out significant, according to the Wald test, for a significance level of 5%, was the interaction between Age and property_A124 (No property).

Duration of the facility was no longer significant.

However, the AIC of the model was higher than the initial models. Thus, we stick with model 2. Unlike the portuguese paper the model with interaction of variables yielded the lowest AIC.

Model Estimates

```
Start coding or generate with AI.
```

Model Evaluation

```
#Goodness of fit test using Hosmer Lemeshow Test:
def hosmer_lemeshow_test(y_true, y_prob, g=10):
    """Hosmer-Lemeshow goodness-of-fit test"""
    data = pd.DataFrame({'y_true': y_true, 'y_prob': y_prob})
```

```
    data['decile'] = pd.qcut(data['y_prob'], q=g, duplicates='drop')

    obs = data.groupby('decile')['y_true'].agg(['sum', 'count'])
    obs.columns = ['events', 'total']
    obs['non_events'] = obs['total'] - obs['events']

    exp = data.groupby('decile')['y_prob'].agg(['mean'])
    obs['exp_events'] = obs['total'] * exp['mean']
    obs['exp_non_events'] = obs['total'] * (1 - exp['mean'])

    hl_stat = (((obs['events'] - obs['exp_events']) ** 2) / obs['exp_events'] +
               ((obs['non_events'] - obs['exp_non_events']) ** 2) / obs['exp_non_events']).sum()

    p_value = 1 - stats.chi2.cdf(hl_stat, g - 2)
    print(f"Hosmer–Lemeshow Test: Chi2 = {hl_stat:.4f}, df = {g-2}, p-value = {p_value:.4f}")
    return hl_stat, p_value

    # -------------- Run Evaluation --------------
# Predict probabilities
y_train_pred_prob = logit_model.predict(X_train_const)

# Hosmer-Lemeshow
hosmer_lemeshow_test(y_train, y_train_pred_prob, g=10)
```

```
Hosmer–Lemeshow Test: Chi2 = 15.9220, df = 8, p-value = 0.0435
<ipython-input-61-86a67d44551d>:7: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future ve
  obs = data.groupby('decile')['y_true'].agg(['sum', 'count'])
<ipython-input-61-86a67d44551d>:11: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future v
  exp = data.groupby('decile')['y_prob'].agg(['mean'])
(np.float64(15.922023449377225), np.float64(0.043509891765540876))
```

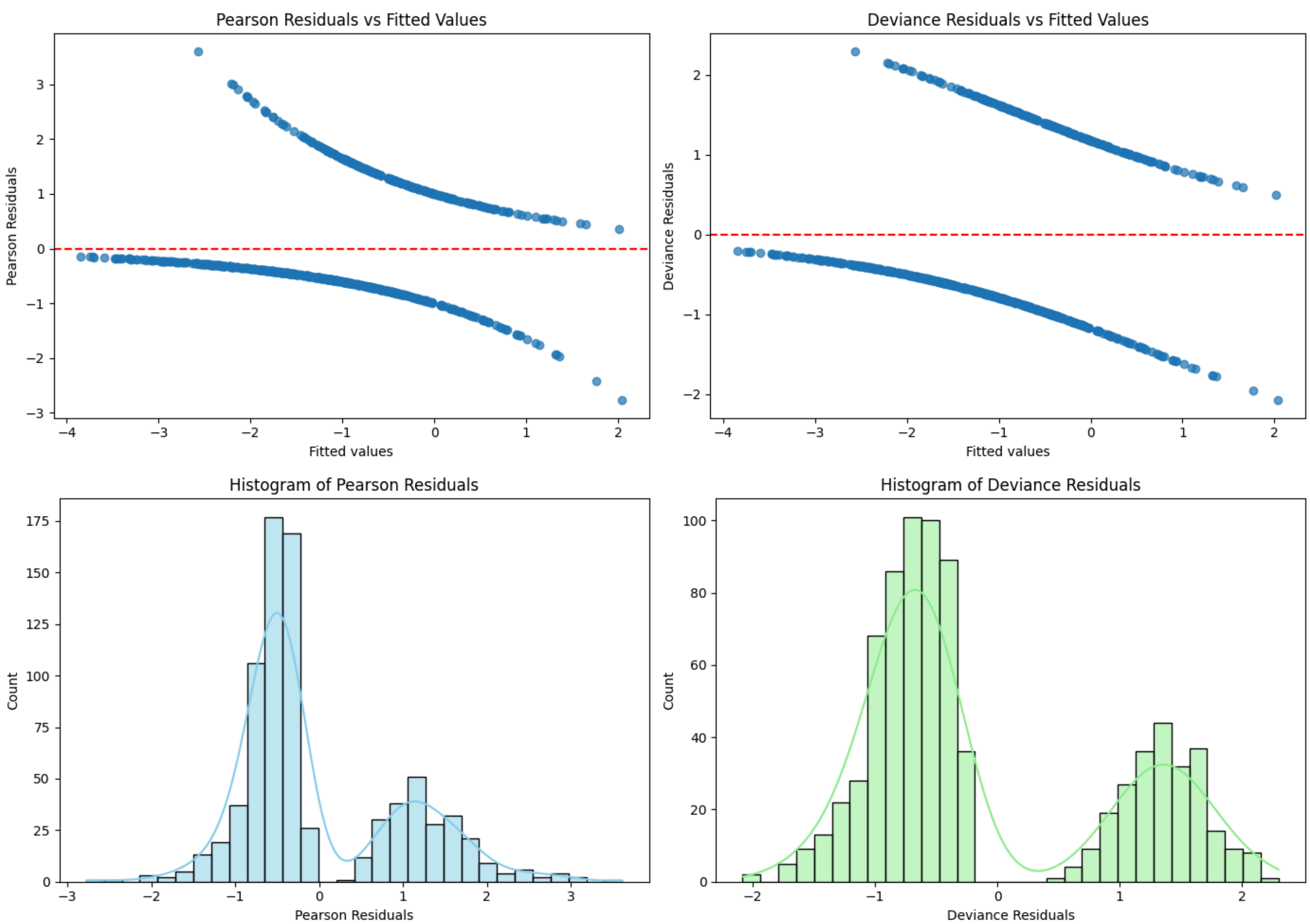
Interpretation:

The p-value is just below 0.05, which suggests that there is a statistically significant difference between the observed and predicted outcomes.

This may indicate the model does not calibrate perfectly — the predicted probabilities deviate from actual default rates in some bins. However, since the p-value is only marginally below 0.05, the model is still usable, though improvement is possible.

```
def residuals_analysis(model, X, y):
    # Pearson and Deviance Residuals
    pearson_resid = model.resid_pearson
    # Changed from model.resid_deviance to model.resid_dev to access the deviance residuals
    deviance_resid = model.resid_dev  # Use resid_dev here

    plt.figure(figsize=(14, 5))

    plt.subplot(1, 2, 1)
    plt.scatter(model.fittedvalues, pearson_resid, alpha=0.7)
    plt.axhline(0, color='red', linestyle='--')
    plt.title("Pearson Residuals vs Fitted Values")
    plt.xlabel("Fitted values")
    plt.ylabel("Pearson Residuals")

    plt.subplot(1, 2, 2)
    plt.scatter(model.fittedvalues, deviance_resid, alpha=0.7)
    plt.axhline(0, color='red', linestyle='--')
    plt.title("Deviance Residuals vs Fitted Values")
    plt.xlabel("Fitted values")
    plt.ylabel("Deviance Residuals")

    plt.tight_layout()
    plt.show()

    # Histograms
    plt.figure(figsize=(14, 5))

    plt.subplot(1, 2, 1)
    sns.histplot(pearson_resid, kde=True, bins=30, color='skyblue')
    plt.title("Histogram of Pearson Residuals")
    plt.xlabel("Pearson Residuals")

    plt.subplot(1, 2, 2)
    sns.histplot(deviance_resid, kde=True, bins=30, color='lightgreen')
    plt.title("Histogram of Deviance Residuals")
```

```
        plt.xlabel("Deviance Residuals")

        plt.tight_layout()
        plt.show()

    # Call the function *outside* the definition
    residuals_analysis(logit_model, X_train_const, y_train)

    # Extract residuals
    pearson_resid = logit_model.resid_pearson
    deviance_resid = logit_model.resid_dev  # Use resid_dev here as well

    # Compute statistics
    print("Pearson Residuals:")
    print("  Mean     :", np.mean(pearson_resid))
    print("  Variance :", np.var(pearson_resid, ddof=1))  # sample variance

    print("\nDeviance Residuals:")
    print("  Mean     :", np.mean(deviance_resid))
    print("  Variance :", np.var(deviance_resid, ddof=1))  # sample variance
```

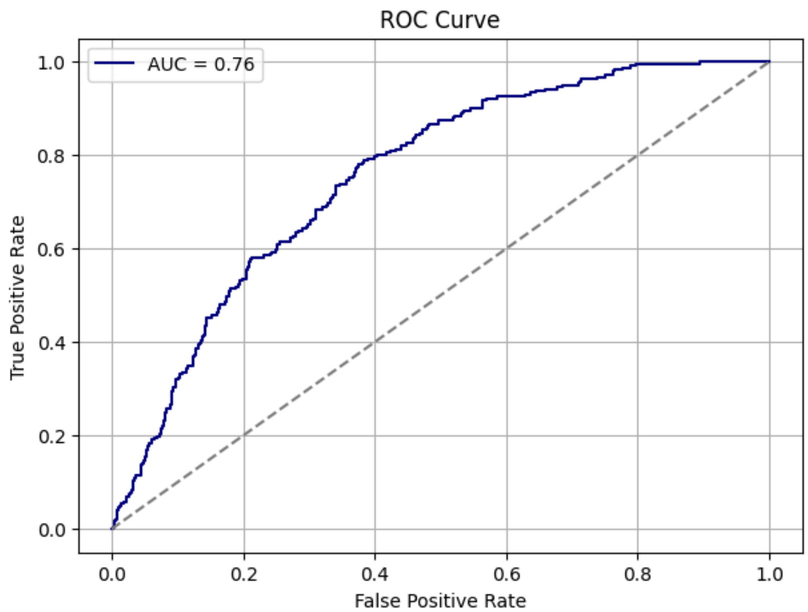



```
Pearson Residuals:
  Mean     : -0.017354688561047344
  Variance : 0.9428772562470751

Deviance Residuals:
  Mean     : -0.10917756879477458
  Variance : 1.0441743154192236
```

The residuals are mostly centered and reasonably dispersed, but the slightly negative means and small variance deviations might hint at minor model misspecification or imbalance in the classes (which matches your earlier low recall).

```
# Confusion matrix & ROC Curve
def classification_performance(model, X, y):
    y_prob = model.predict(X)
    y_pred = (y_prob >= 0.5).astype(int)

    # Confusion Matrix
    print("Confusion Matrix:")
    print(confusion_matrix(y, y_pred))

    # Classification Report
    print("\nClassification Report:")
    print(classification_report(y, y_pred))

    # Accuracy
    acc = accuracy_score(y, y_pred)
    print("Accuracy: {:.4f}".format(acc))

    # ROC and AUC
    fpr, tpr, _ = roc_curve(y, y_prob)
    auc_score = roc_auc_score(y, y_prob)
    print("AUC Score: {:.4f}".format(auc_score))

    # Plot ROC
    plt.figure(figsize=(7, 5))
    plt.plot(fpr, tpr, label='AUC = {:.2f}'.format(auc_score), color='navy')
    plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.legend()
    plt.grid()
    plt.show()

#Confusion Matrix, AUC, ROC
classification_performance(logit_model, X_train_const, y_train)
```

```
Confusion Matrix:
[[508  51]
 [173  68]]

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.91      0.82       559
           1       0.57      0.28      0.38       241

    accuracy                           0.72       800
   macro avg       0.66      0.60      0.60       800
weighted avg       0.69      0.72      0.69       800

Accuracy: 0.7200
AUC Score: 0.7553
```



- True Negatives (TN) = 508: Correctly predicted non-defaulters
- False Positives (FP) = 51: Predicted default, but actually non-default
- False Negatives (FN) = 173: Predicted non-default, but actually default
- True Positives (TP) = 68: Correctly predicted defaulters

The model is better at identifying non-defaulters than defaulters.

- High Recall (0.91) for Class 0 - Most non-defaulters are correctly identified.
- Low Recall (0.28) for Class 1 - The model misses many actual defaulters.
- F1-score (0.38) for defaulters suggests weak performance in detecting risky clients.

Overall, the model correctly classifies 72% of cases.

However, this is likely driven by the class imbalance (more non-defaulters).

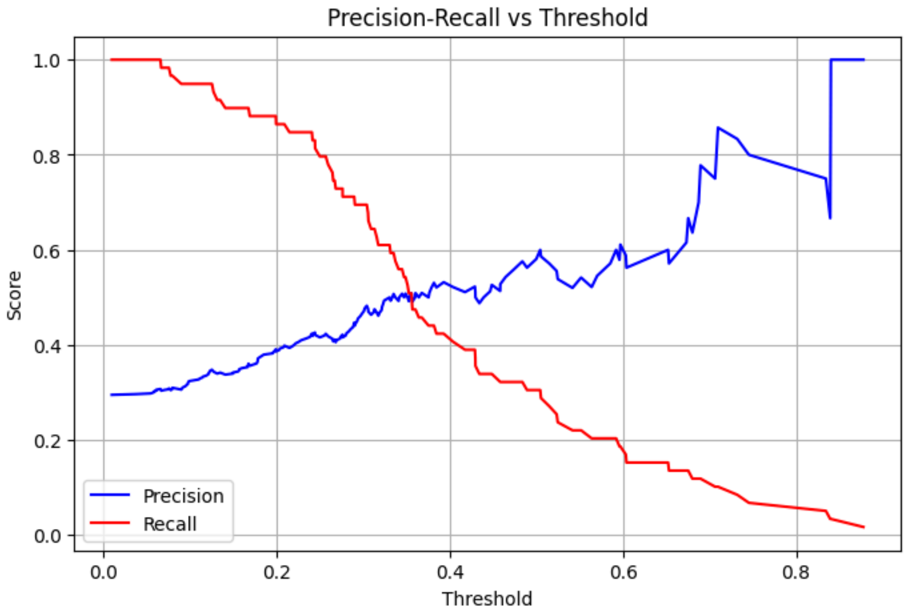
AUC of 0.7553 is moderately good and suggests decent discriminatory power.

We will run attempts to improve recall rate, as it is worse to class a customer as good when they are bad, than it is to class a customer as bad when they are good in credit risk.

In comparison to the Portuguese dataset, the recall of defaulters was at 0.94% in a credit risk business. While, the recall of non-defaulters was at 89.79%.

```
#Adjusting Classification Threshold
# Get predicted probabilities from the model
y_probs = logit_model.predict(X_train_const)

# Compute precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_train, y_probs)

# Select the final features in X_test (similar to X_train_selected)
X_test_selected = X_test[final_selected_features]
```

```
# Add constant to the selected features in X_test
X_test_const = sm.add_constant(X_test_selected)

# Get predicted probabilities using the selected features and constant term
y_probs = logit_model.predict(X_test_const)  # predicted probabilities

# Compute precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)

# Plot precision vs recall
plt.figure(figsize=(8,5))
plt.plot(thresholds, precision[:-1], label='Precision', color='b')
plt.plot(thresholds, recall[:-1], label='Recall', color='r')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.title('Precision-Recall vs Threshold')
plt.legend()
plt.grid(True)
plt.show()

# Choose new threshold, e.g., 0.35 (adjust based on plot)
new_threshold = 0.35
y_pred_new = (y_probs >= new_threshold).astype(int)

# Evaluate new performance
print("Confusion Matrix (Threshold = 0.35):")
print(confusion_matrix(y_test, y_pred_new))

print("\nClassification Report:")
print(classification_report(y_test, y_pred_new))

print("New AUC Score:", roc_auc_score(y_test, y_probs))

# ROC and AUC
#Dedenting the following block to align with the rest of the code outside of function definition
fpr, tpr, _ = roc_curve(y_test, y_probs) #Using y_test and y_probs for ROC calculation on test set.
auc_score = roc_auc_score(y_test, y_probs) #Using y_test and y_probs for AUC calculation on test set.
print("AUC Score: {:.4f}".format(auc_score))

# Plot ROC
plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr, label='AUC = {:.2f}'.format(auc_score), color='navy')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid()
plt.show()
```




```
Confusion Matrix (Threshold = 0.35):
[[110  31]
 [ 28  31]]

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       141
           1       0.50      0.53      0.51        59

    accuracy                           0.70       200
   macro avg       0.65      0.65      0.65       200
weighted avg       0.71      0.70      0.71       200

New AUC Score: 0.7213607404736145
```