In [6]:
```
import pandas as pd

df = pd.read_csv("C:/Users/Admin/Downloads/index_data.csv")
print(df.head())
print(df.info())
```

```
   Creditability  Account Balance  Duration of Credit (month)  \
0              1                1                          18
1              1                1                           9
2              1                2                          12
3              1                1                          12
4              1                1                          12

   Payment Status of Previous Credit  Purpose  Credit Amount  \
0                                  4        2           1049
1                                  4        0           2799
2                                  2        9            841
3                                  4        0           2122
4                                  4        0           2171

   Value Savings/Stocks  Length of current employment  Instalment per cent  \
0                     1                             2                    4
1                     1                             3                    2
2                     2                             4                    2
3                     1                             3                    3
4                     1                             3                    4

   Sex & Marital Status  ...  Duration in Current address  \
0                     2  ...                           4
1                     3  ...                           2
2                     2  ...                           4
3                     3  ...                           2
4                     3  ...                           4

   Most valuable available asset  Age (years)  Concurrent Credits  \
0                              2           21                   3
1                              1           36                   3
2                              1           23                   3
3                              1           39                   3
4                              2           38                   1

   Type of apartment  No of Credits at this Bank  Occupation  \
0                  1                           1           3
1                  1                           2           3
2                  1                           1           2
3                  1                           2           2
4                  2                           2           2

   No of dependents  Telephone  Foreign Worker
0                 1          1               1
1                 2          1               1
2                 1          1               1
3                 2          1               2
4                 1          1               2

[5 rows x 21 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Creditability                    1000 non-null   int64
```

```
 1   Account Balance                     1000 non-null   int64
 2   Duration of Credit (month)          1000 non-null   int64
 3   Payment Status of Previous Credit   1000 non-null   int64
 4   Purpose                             1000 non-null   int64
 5   Credit Amount                       1000 non-null   int64
 6   Value Savings/Stocks                1000 non-null   int64
 7   Length of current employment        1000 non-null   int64
 8   Instalment per cent                 1000 non-null   int64
 9   Sex & Marital Status                1000 non-null   int64
 10  Guarantors                          1000 non-null   int64
 11  Duration in Current address         1000 non-null   int64
 12  Most valuable available asset       1000 non-null   int64
 13  Age (years)                         1000 non-null   int64
 14  Concurrent Credits                  1000 non-null   int64
 15  Type of apartment                   1000 non-null   int64
 16  No of Credits at this Bank          1000 non-null   int64
 17  Occupation                          1000 non-null   int64
 18  No of dependents                    1000 non-null   int64
 19  Telephone                           1000 non-null   int64
 20  Foreign Worker                      1000 non-null   int64
dtypes: int64(21)
memory usage: 164.2 KB
None
```

In [10]:
```
print(df.describe(include='all'))
```

	Creditability	Account Balance	Duration of Credit (month)	\
count	1000.000000	1000.000000	1000.000000	
mean	0.700000	2.577000	20.903000	
std	0.458487	1.257638	12.058814	
min	0.000000	1.000000	4.000000	
25%	0.000000	1.000000	12.000000	
50%	1.000000	2.000000	18.000000	
75%	1.000000	4.000000	24.000000	
max	1.000000	4.000000	72.000000	

	Payment Status of Previous Credit	Purpose	Credit Amount	\
count	1000.00000	1000.000000	1000.00000	
mean	2.54500	2.828000	3271.24800	
std	1.08312	2.744439	2822.75176	
min	0.00000	0.000000	250.00000	
25%	2.00000	1.000000	1365.50000	
50%	2.00000	2.000000	2319.50000	
75%	4.00000	3.000000	3972.25000	
max	4.00000	10.000000	18424.00000	

	Value Savings/Stocks	Length of current employment	\
count	1000.000000	1000.000000	
mean	2.105000	3.384000	
std	1.580023	1.208306	
min	1.000000	1.000000	
25%	1.000000	3.000000	
50%	1.000000	3.000000	
75%	3.000000	5.000000	
max	5.000000	5.000000	

	Instalment per cent	Sex & Marital Status	...	\
count	1000.000000	1000.00000	...	
mean	2.973000	2.68200	...	
std	1.118715	0.70808	...	
min	1.000000	1.00000	...	
25%	2.000000	2.00000	...	
50%	3.000000	3.00000	...	
75%	4.000000	3.00000	...	
max	4.000000	4.00000	...	

	Duration in Current address	Most valuable available asset	\
count	1000.000000	1000.000000	
mean	2.845000	2.358000	
std	1.103718	1.050209	
min	1.000000	1.000000	
25%	2.000000	1.000000	
50%	3.000000	2.000000	
75%	4.000000	3.000000	
max	4.000000	4.000000	

	Age (years)	Concurrent Credits	Type of apartment	\
count	1000.00000	1000.000000	1000.000000	
mean	35.54200	2.675000	1.928000	
std	11.35267	0.705601	0.530186	
min	19.00000	1.000000	1.000000	
25%	27.00000	3.000000	2.000000	

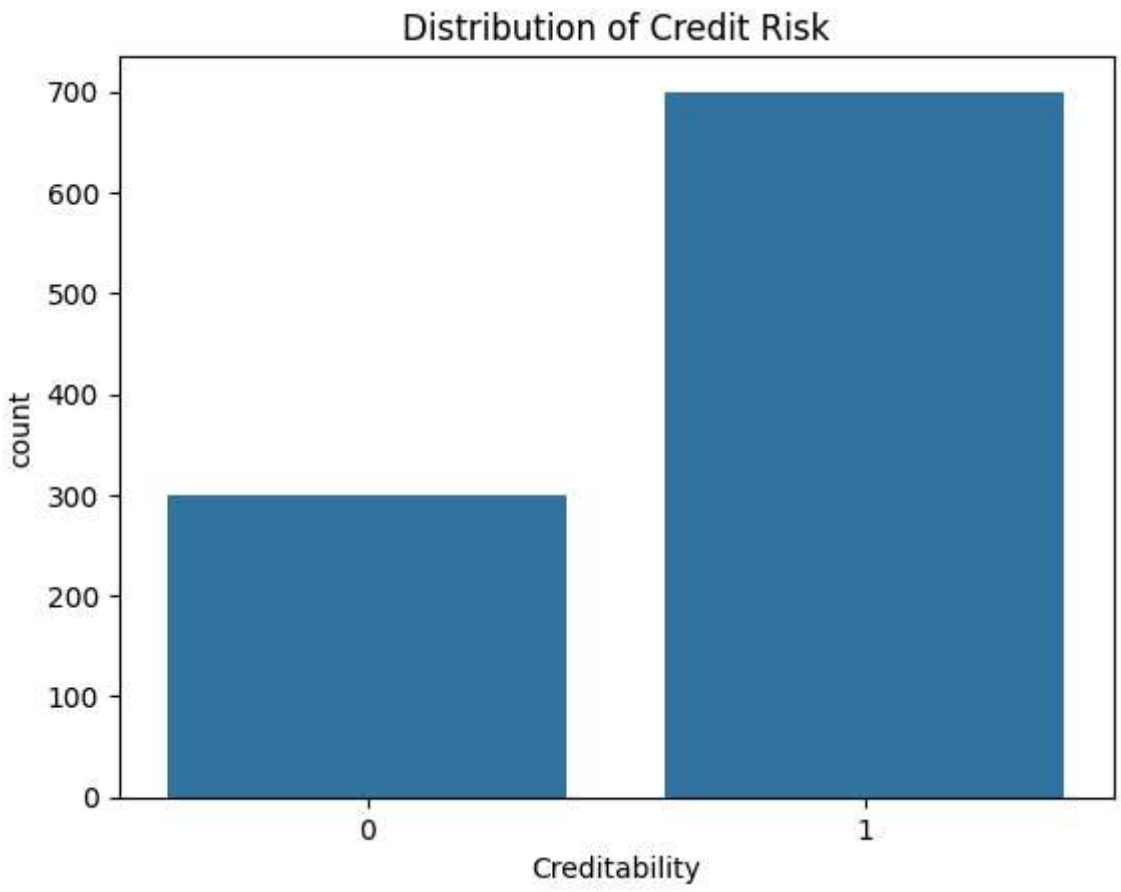
```
50%       33.00000            3.000000           2.000000
75%       42.00000            3.000000           2.000000
max       75.00000            3.000000           3.000000

       No of Credits at this Bank   Occupation  No of dependents    Telephone  \
count                 1000.000000  1000.000000      1000.000000  1000.000000
mean                     1.407000     2.904000         1.155000     1.404000
std                      0.577654     0.653614         0.362086     0.490943
min                      1.000000     1.000000         1.000000     1.000000
25%                      1.000000     3.000000         1.000000     1.000000
50%                      1.000000     3.000000         1.000000     1.000000
75%                      2.000000     3.000000         1.000000     2.000000
max                      4.000000     4.000000         2.000000     2.000000

       Foreign Worker
count     1000.000000
mean         1.037000
std          0.188856
min          1.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          2.000000

[8 rows x 21 columns]
```

In [11]:
```
print(df.isnull().sum())
```

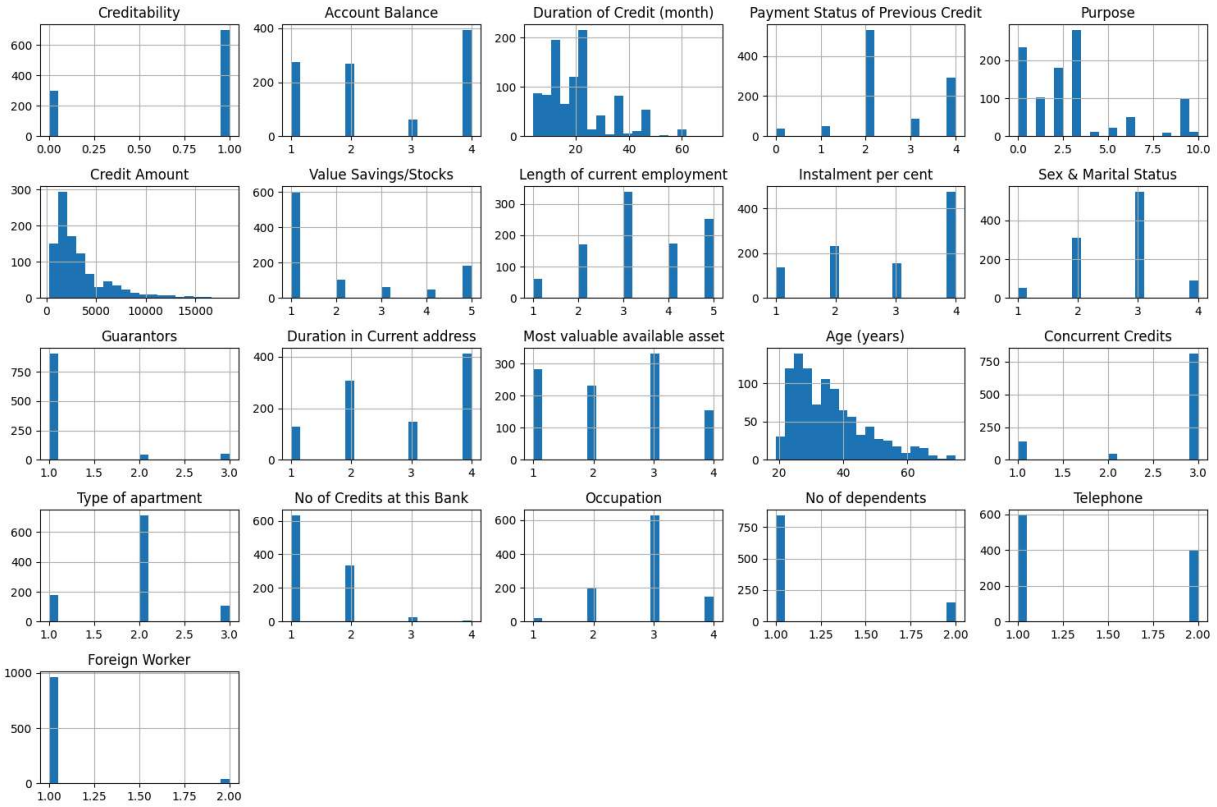
```
Creditability                        0
Account Balance                      0
Duration of Credit (month)           0
Payment Status of Previous Credit    0
Purpose                              0
Credit Amount                        0
Value Savings/Stocks                 0
Length of current employment         0
Instalment per cent                  0
Sex & Marital Status                 0
Guarantors                           0
Duration in Current address          0
Most valuable available asset        0
Age (years)                          0
Concurrent Credits                   0
Type of apartment                    0
No of Credits at this Bank           0
Occupation                           0
No of dependents                     0
Telephone                            0
Foreign Worker                       0
dtype: int64
```

In [13]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x='Creditability', data=df)
```

```
plt.title("Distribution of Credit Risk")
plt.show()
```

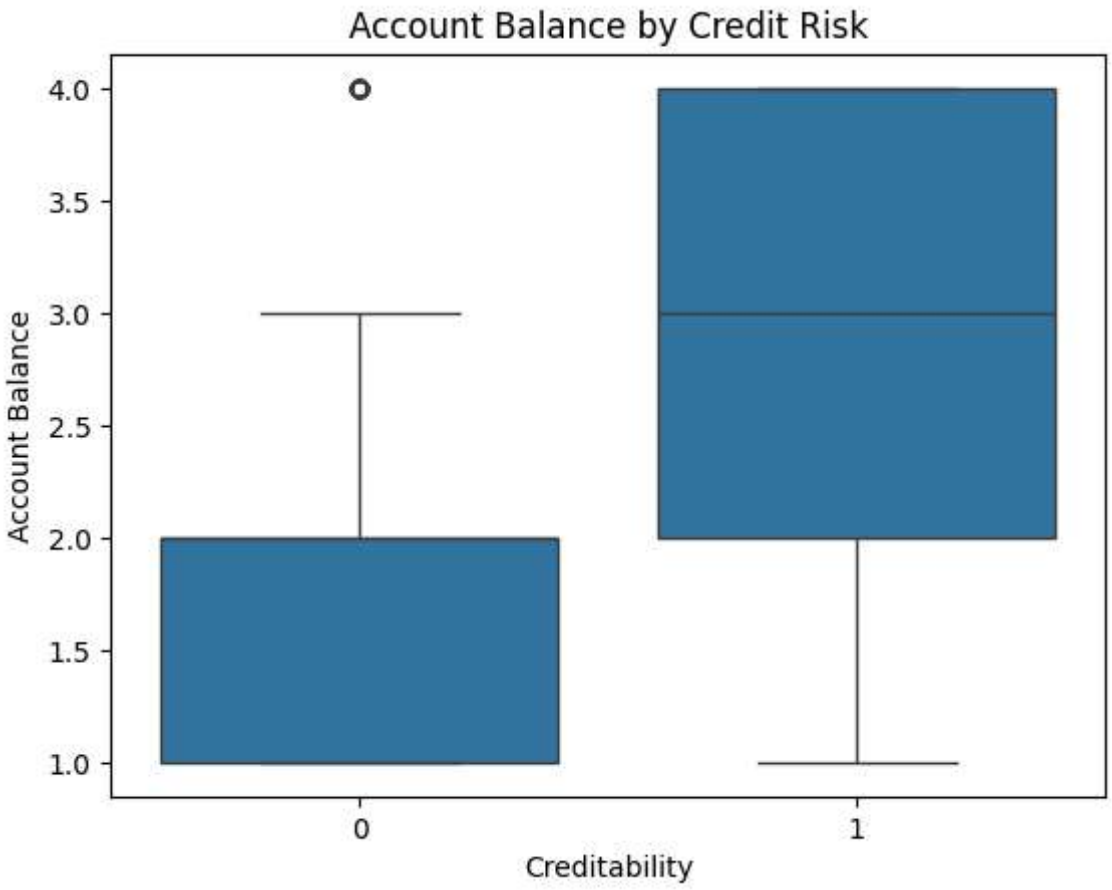


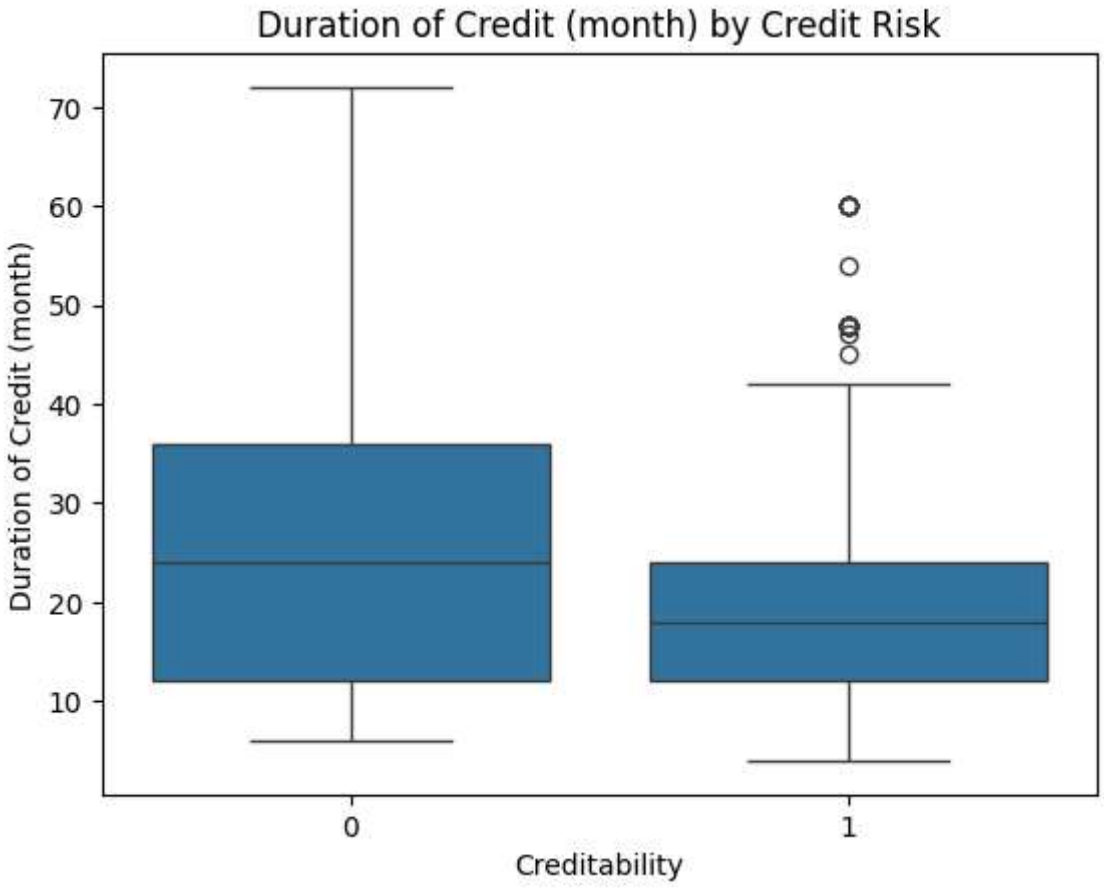
In [14]:
```
df.hist(figsize=(15,10), bins=20)
plt.tight_layout()
plt.show()
```

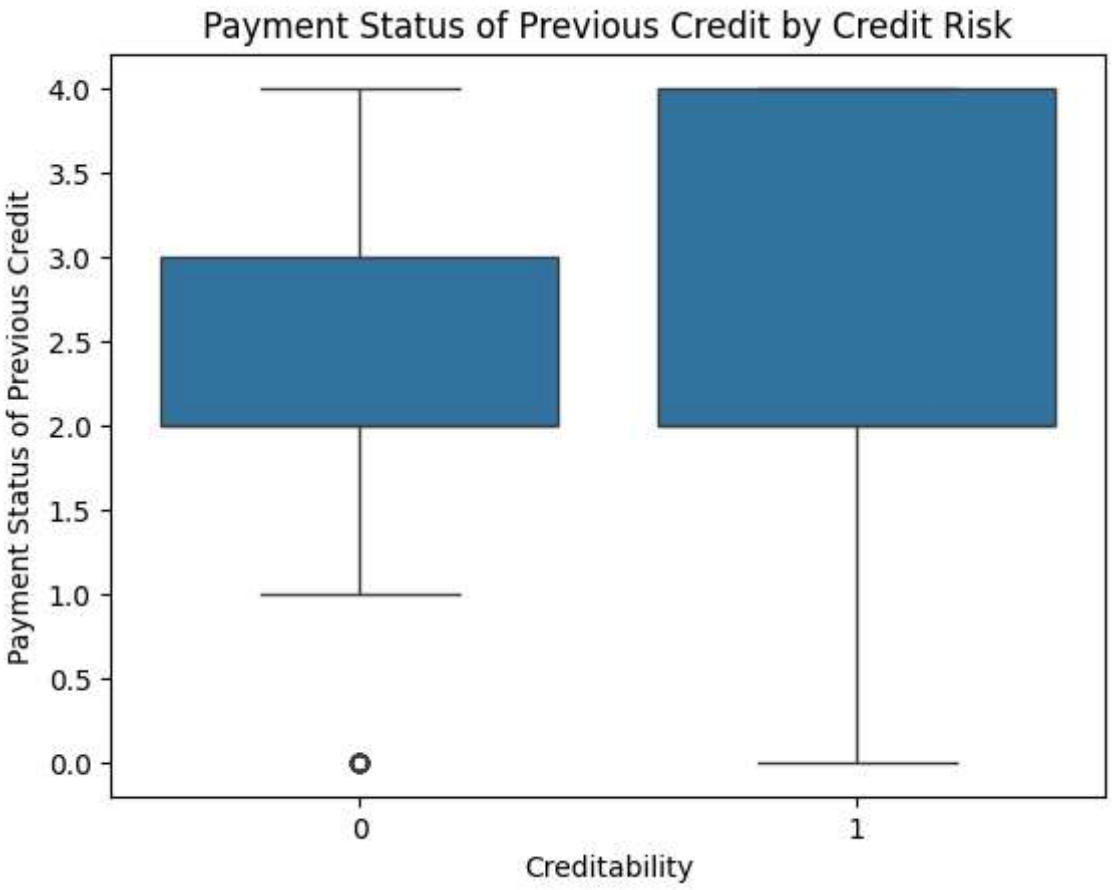


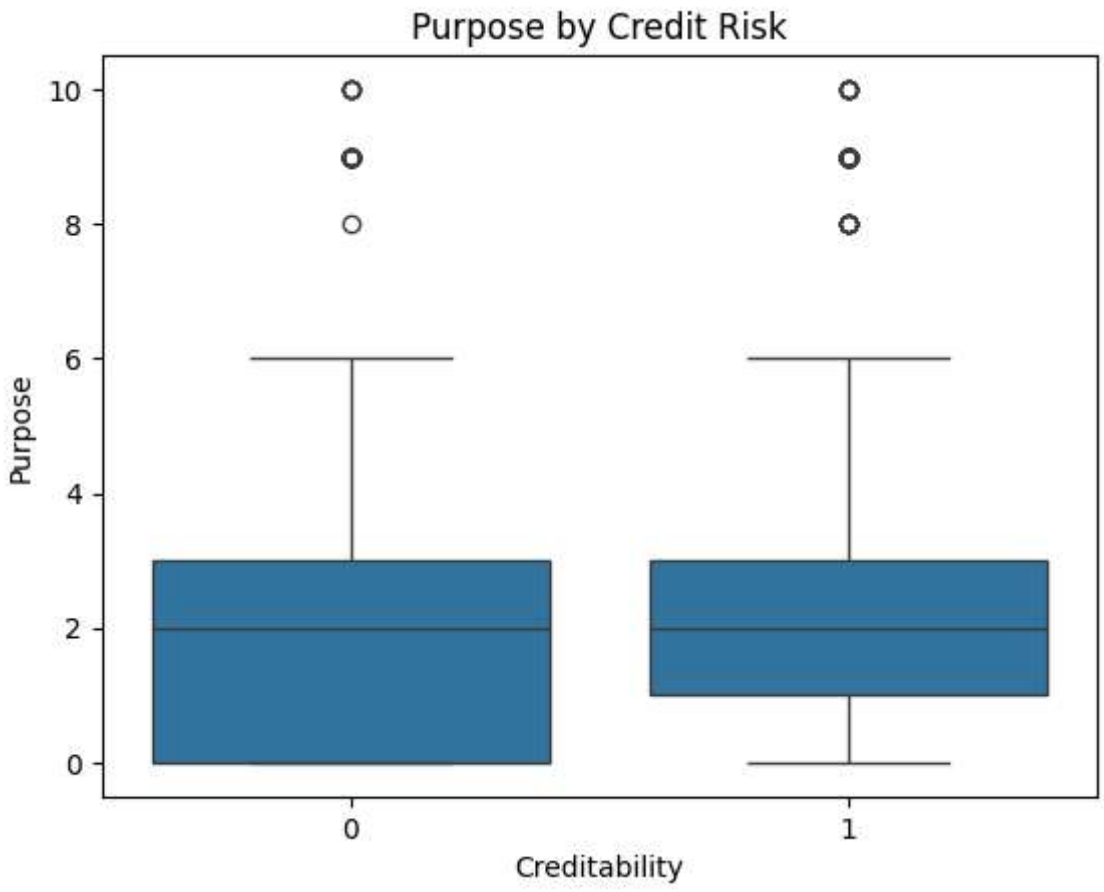
In [16]:
```
for col in df.select_dtypes(include='object').columns:
    sns.countplot(y=col, data=df)
    plt.title(f"Distribution of {col}")
    plt.show()
```

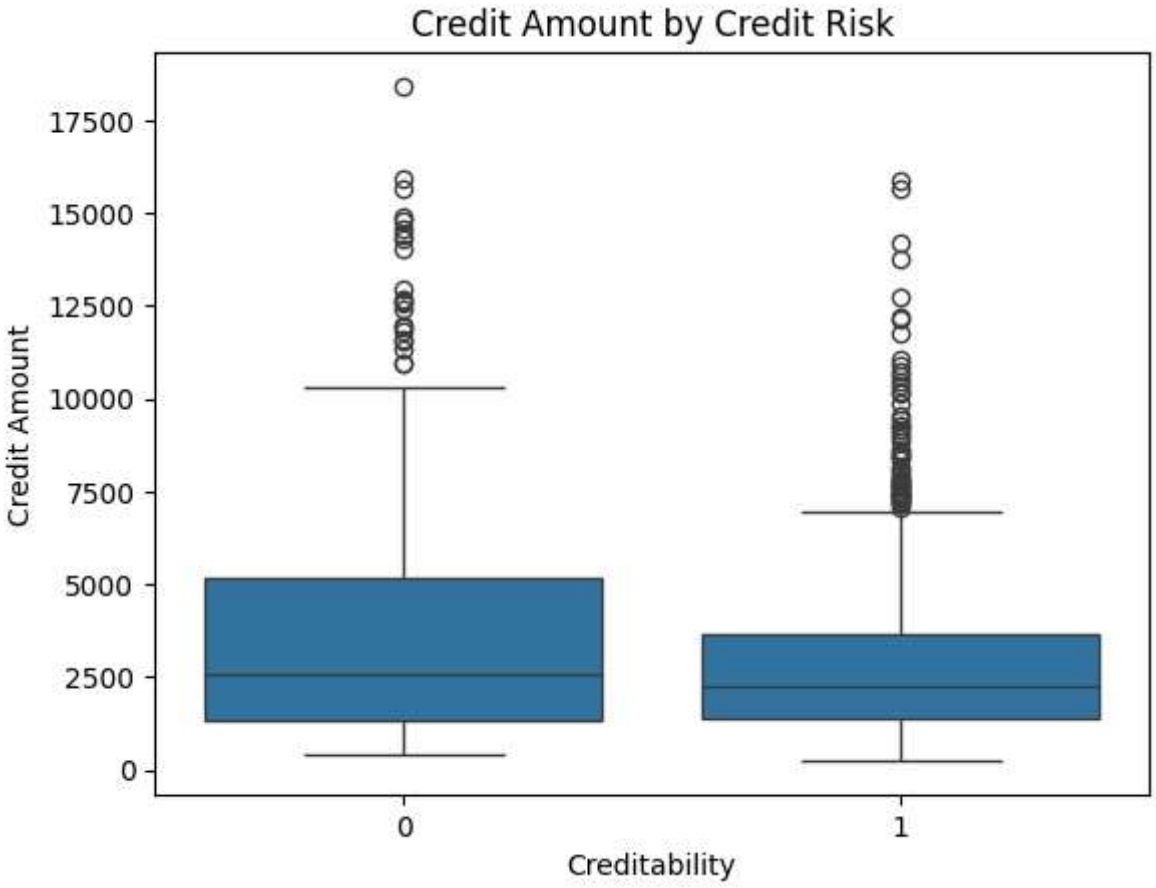
In [17]:
```
import seaborn as sns

for col in df.select_dtypes(include='number').columns:
    if col != 'Creditability':
        sns.boxplot(x='Creditability', y=col, data=df)
        plt.title(f"{col} by Credit Risk")
        plt.show()
```

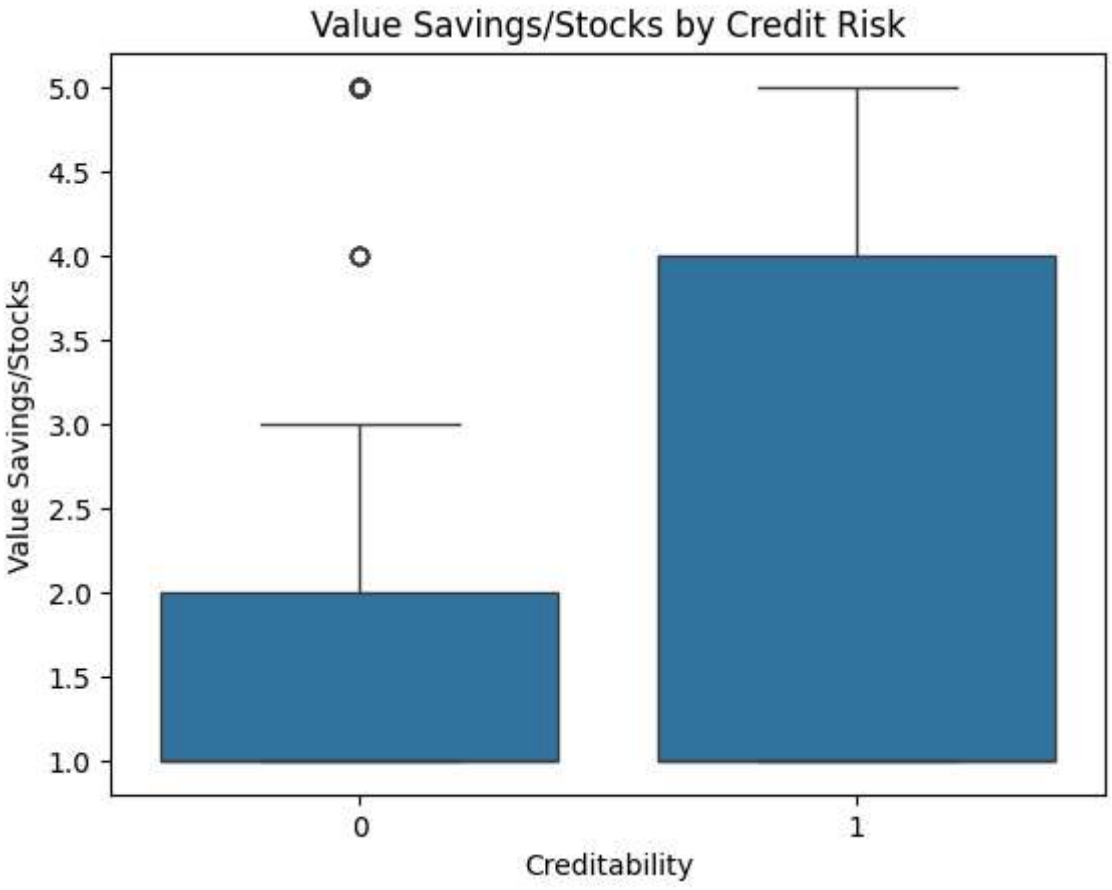

Account Balance by Credit Risk
Account Balance
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
Creditability

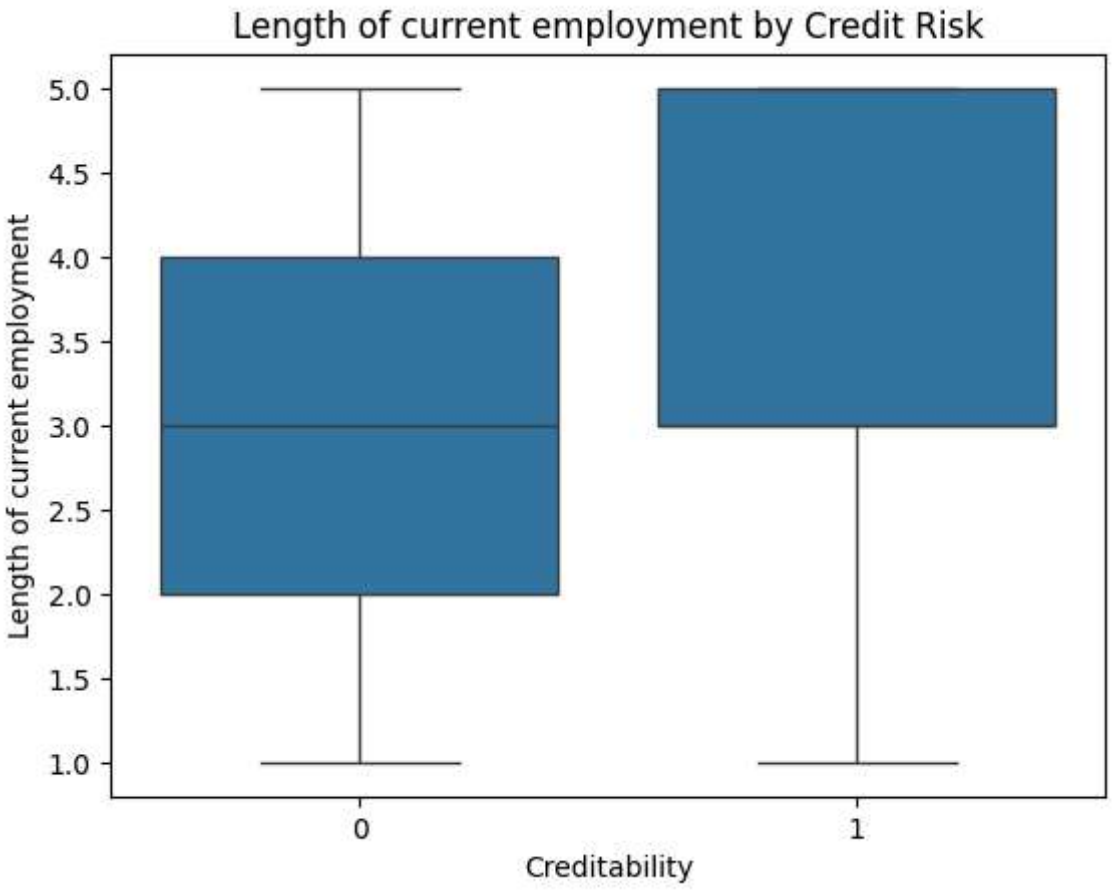

Duration of Credit (month) by Credit Risk
Duration of Credit (month)
70
60
50
40
30
20
10
0
1
Creditability

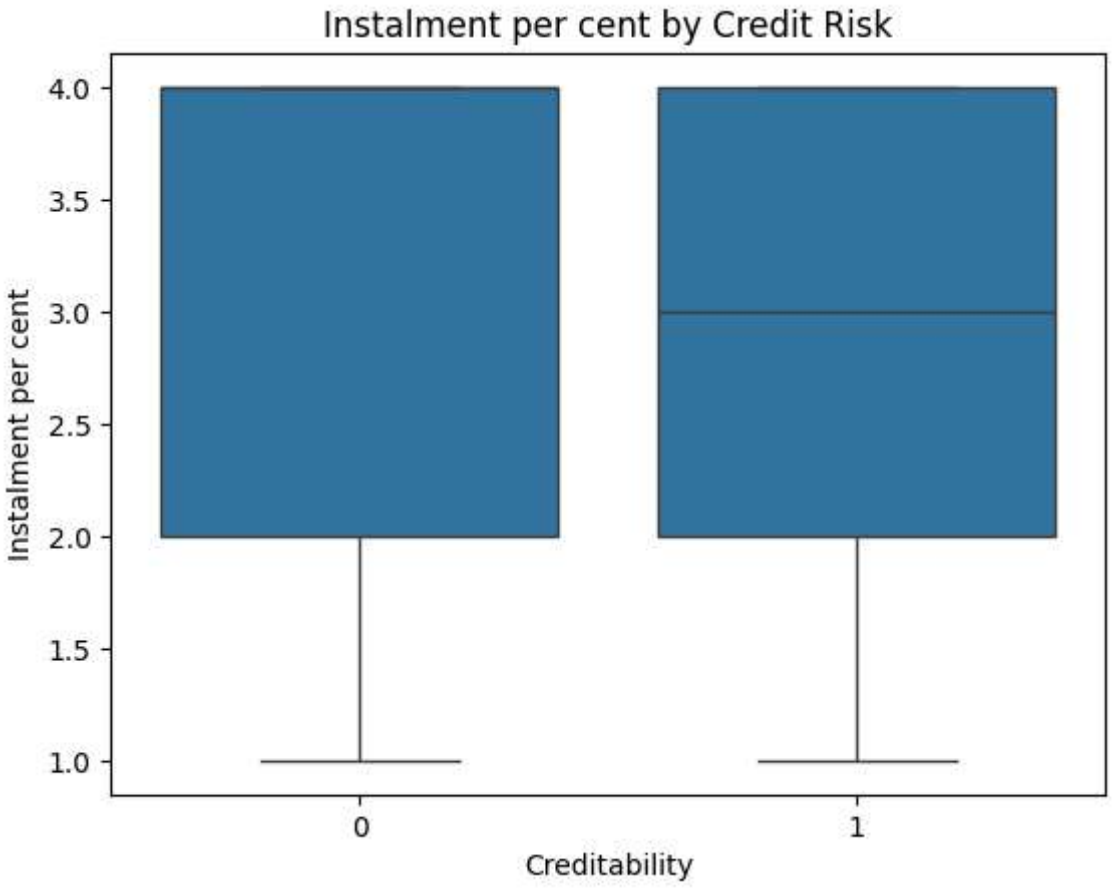

Payment Status of Previous Credit by Credit Risk
Payment Status of Previous Credit
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
Creditability

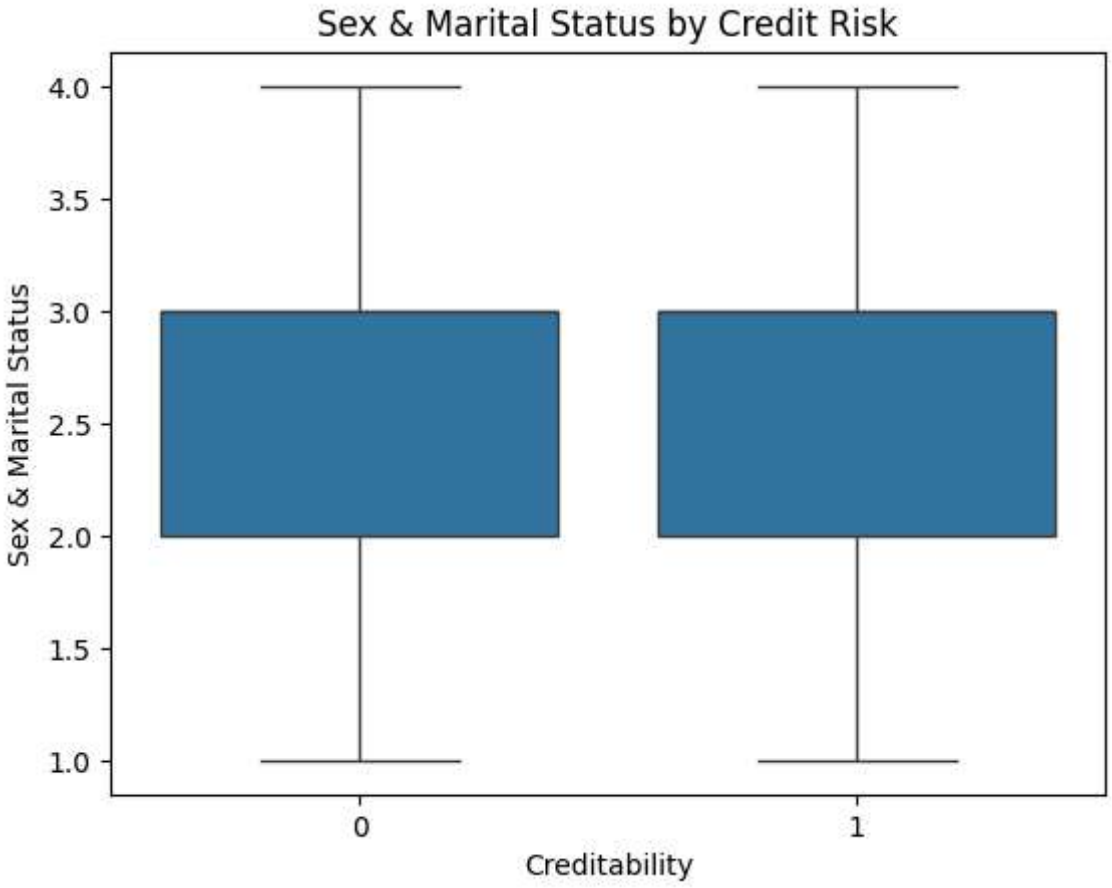

Purpose by Credit Risk
Purpose
10
8
6
4
2
0
0
1
Creditability

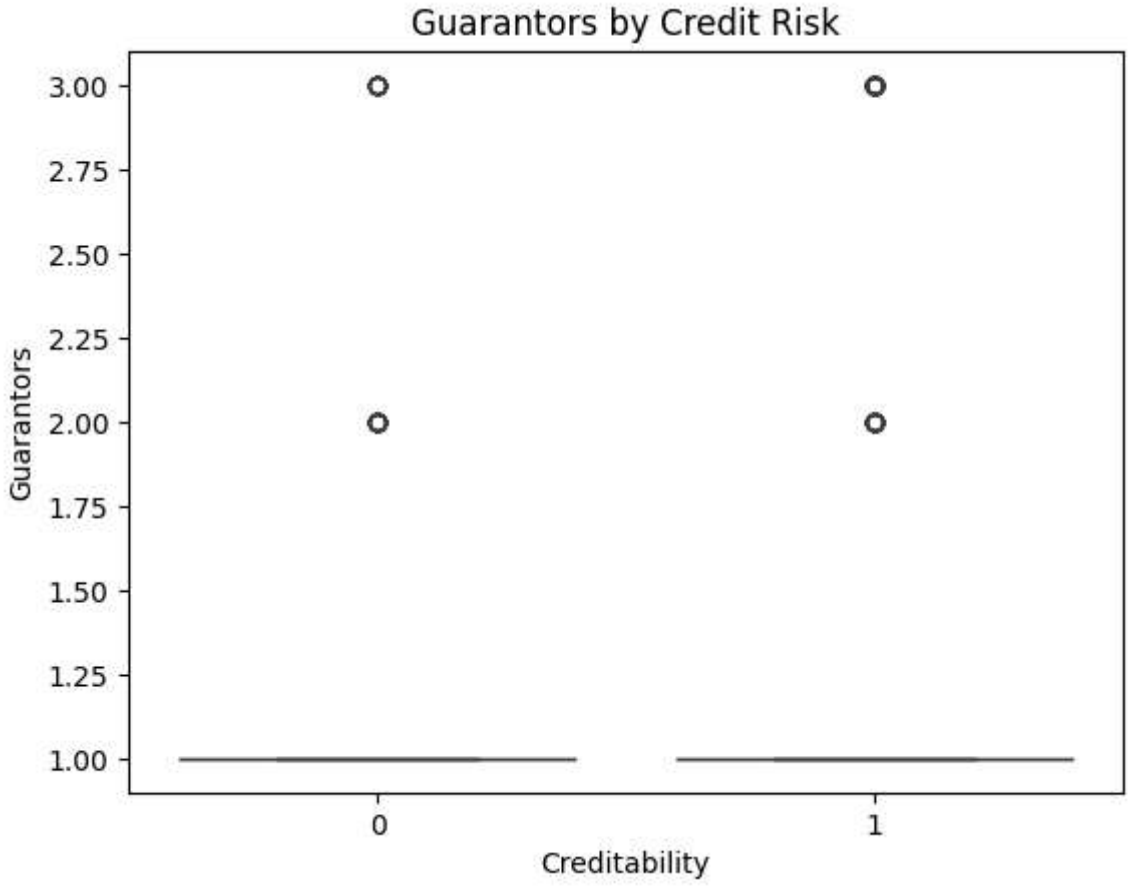

Credit Amount by Credit Risk
Credit Amount
17500
15000
12500
10000
7500
5000
2500
0
0
1
Creditability

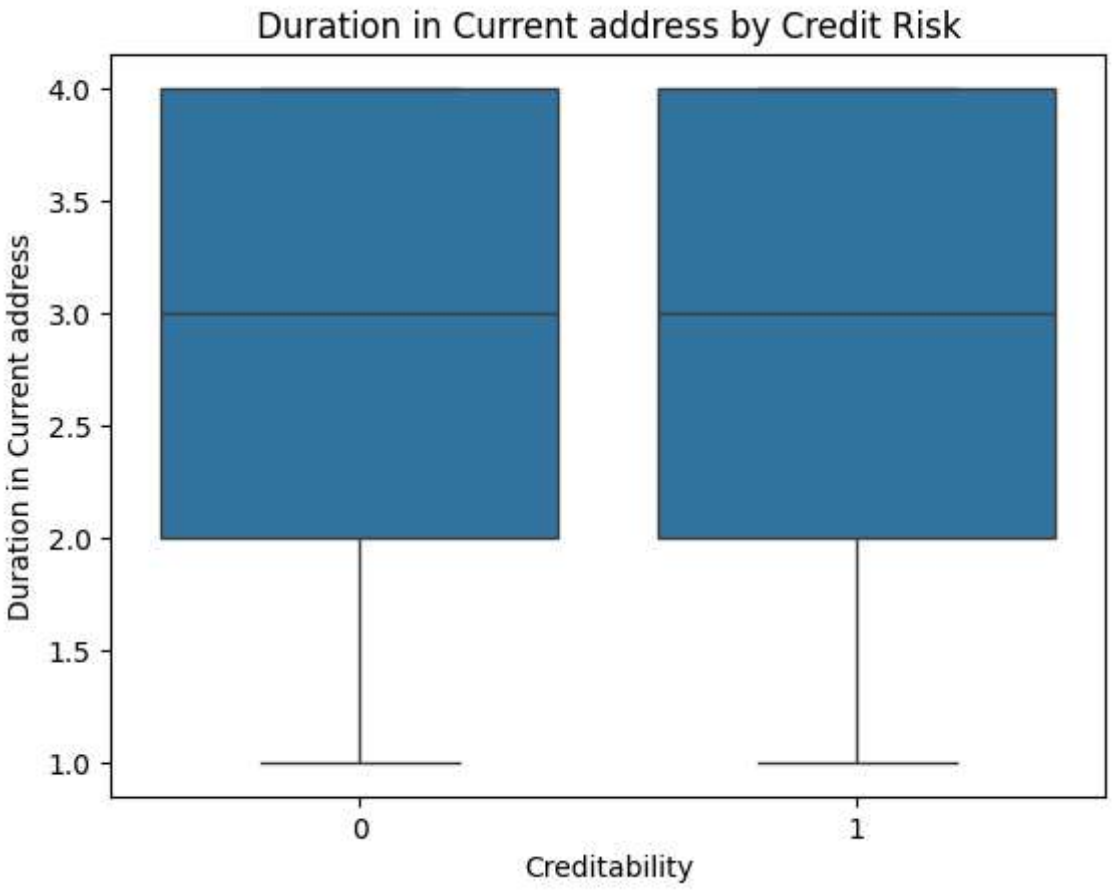

Value Savings/Stocks by Credit Risk
Value Savings/Stocks
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
Creditability

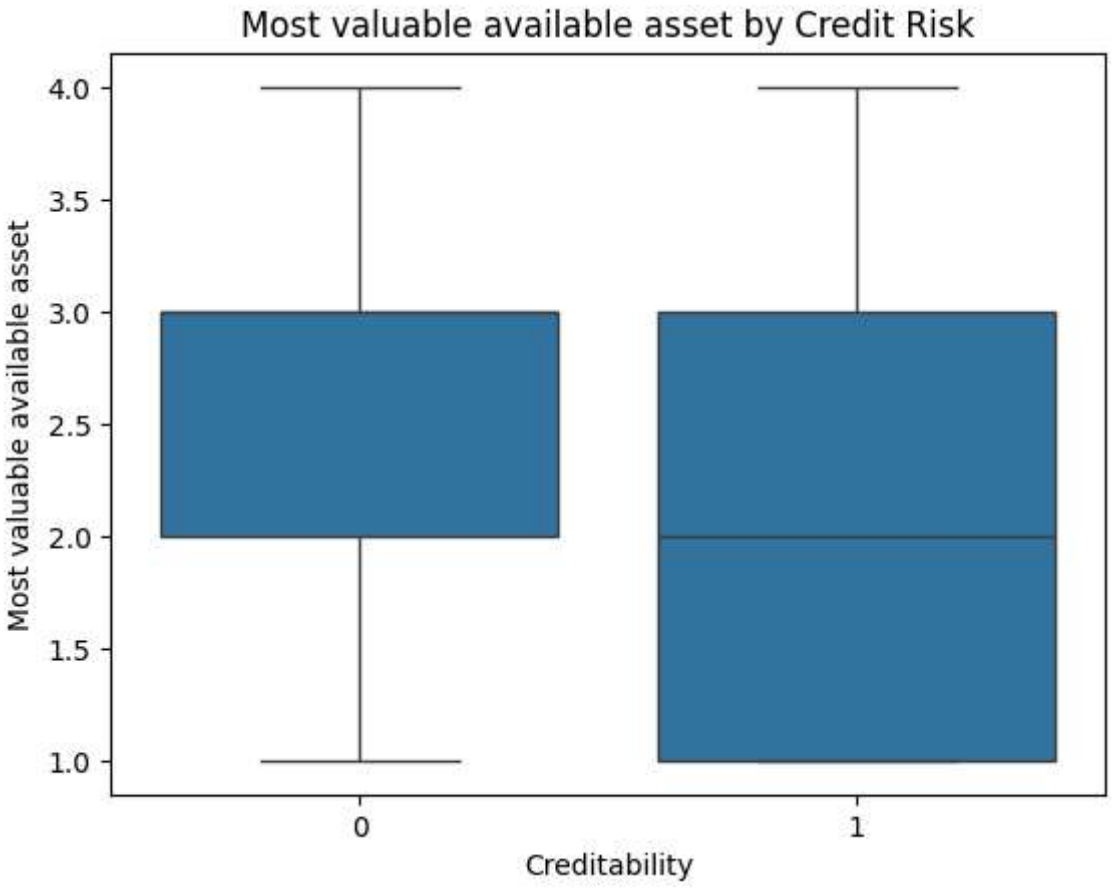

Length of current employment by Credit Risk
Length of current employment
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
Creditability

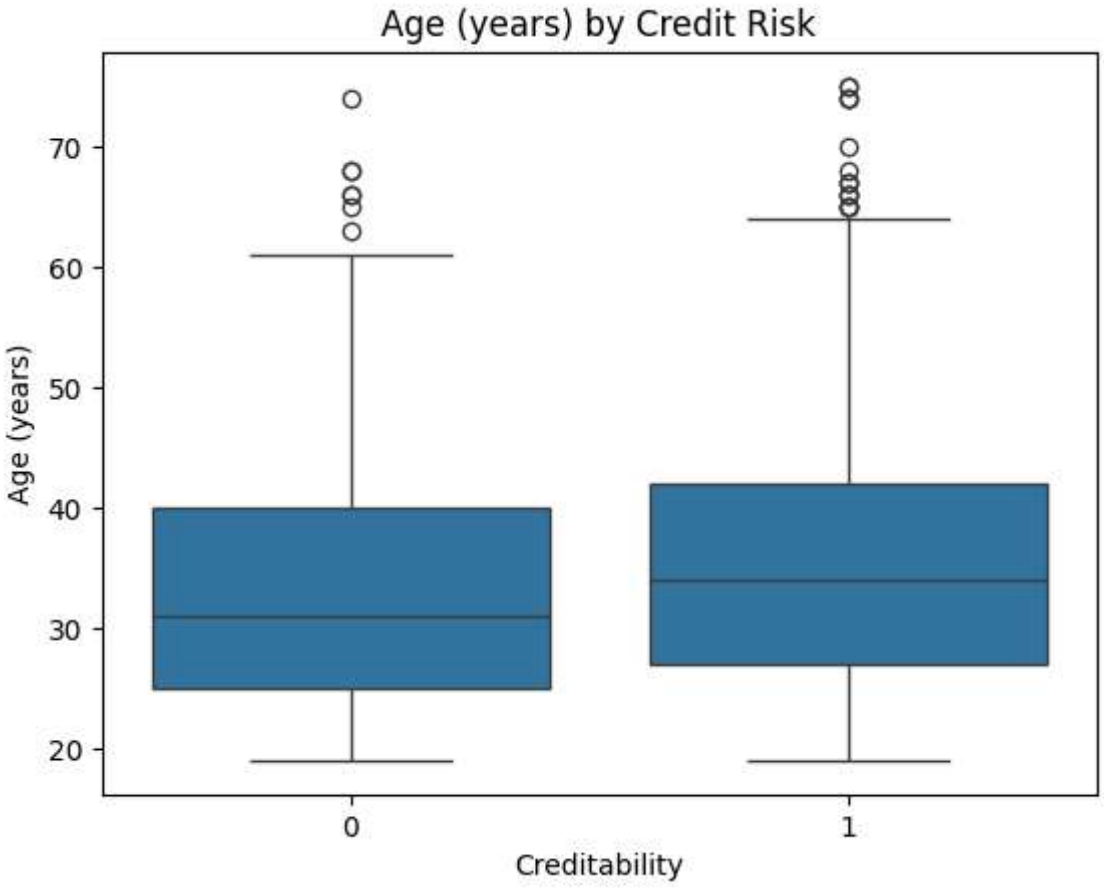

Instalment per cent by Credit Risk
Instalment per cent
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
Creditability

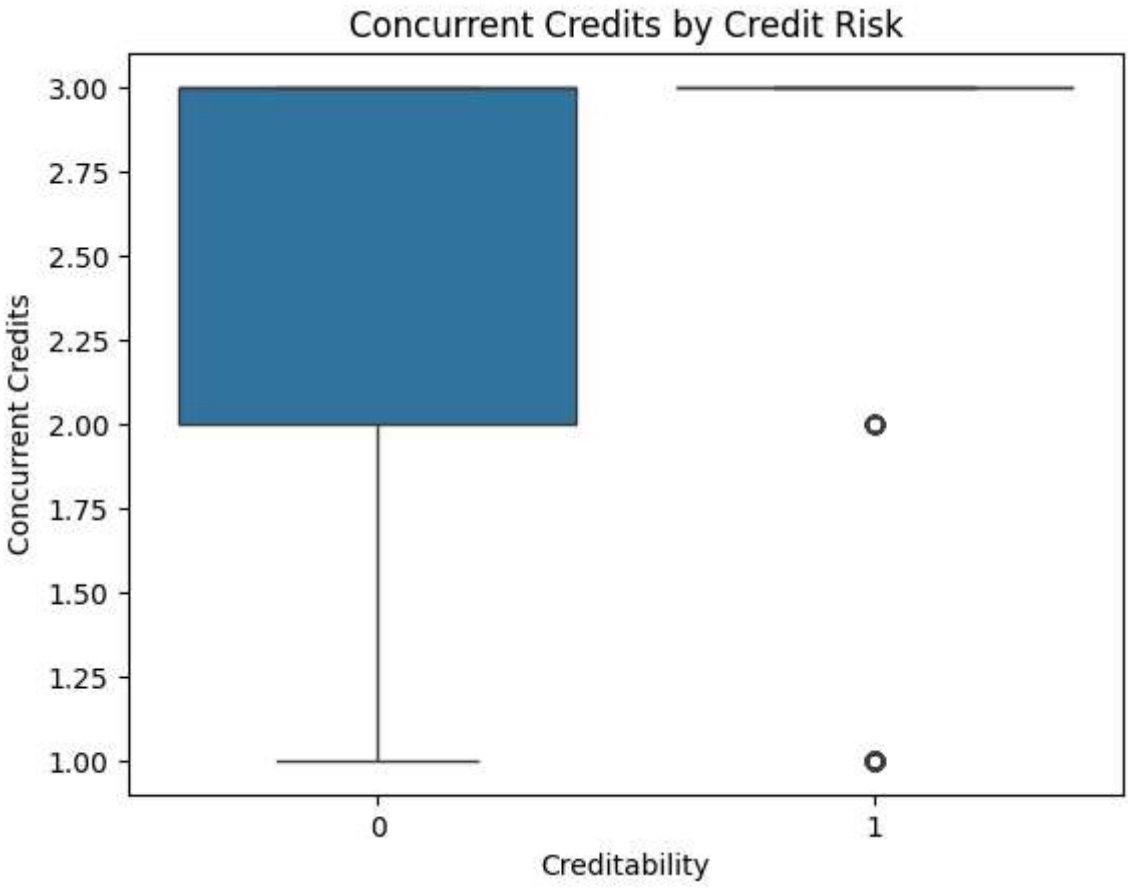

Sex & Marital Status by Credit Risk
Sex & Marital Status
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
Creditability

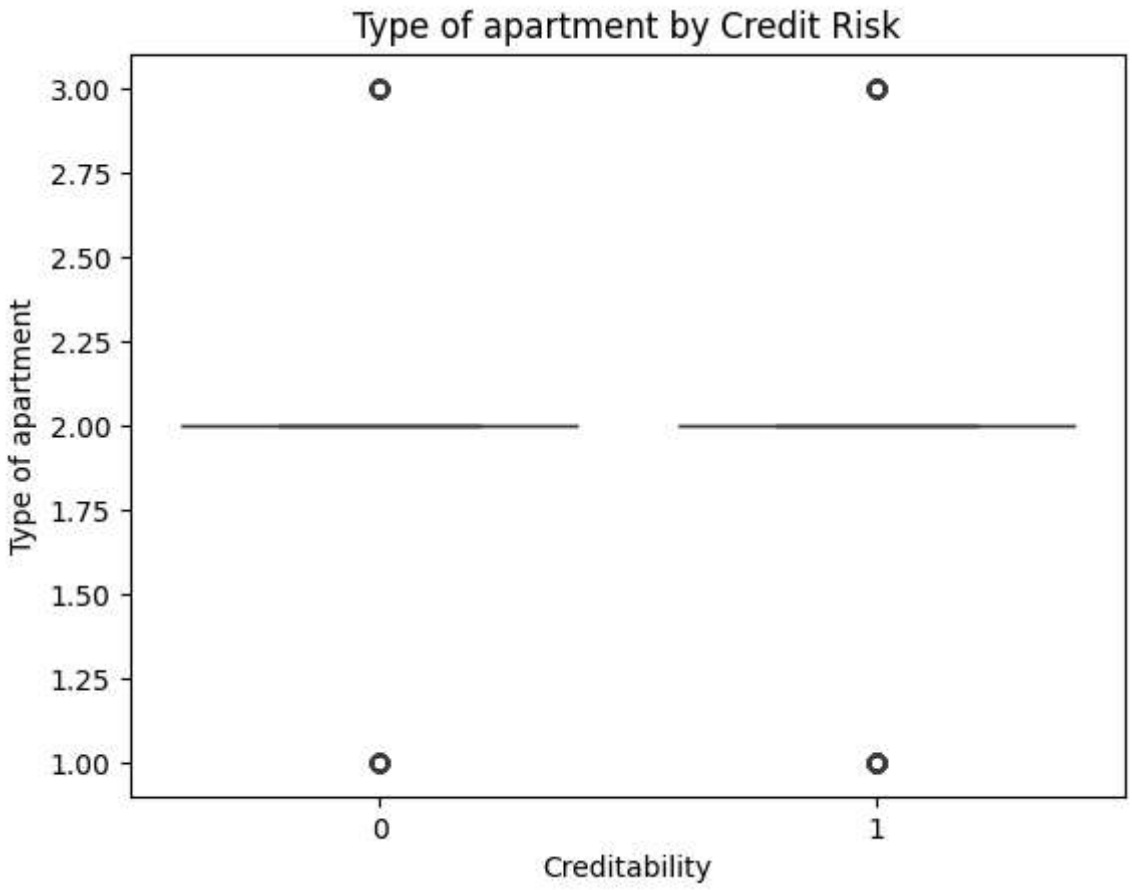

Guarantors by Credit Risk
Guarantors
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0
1
Creditability

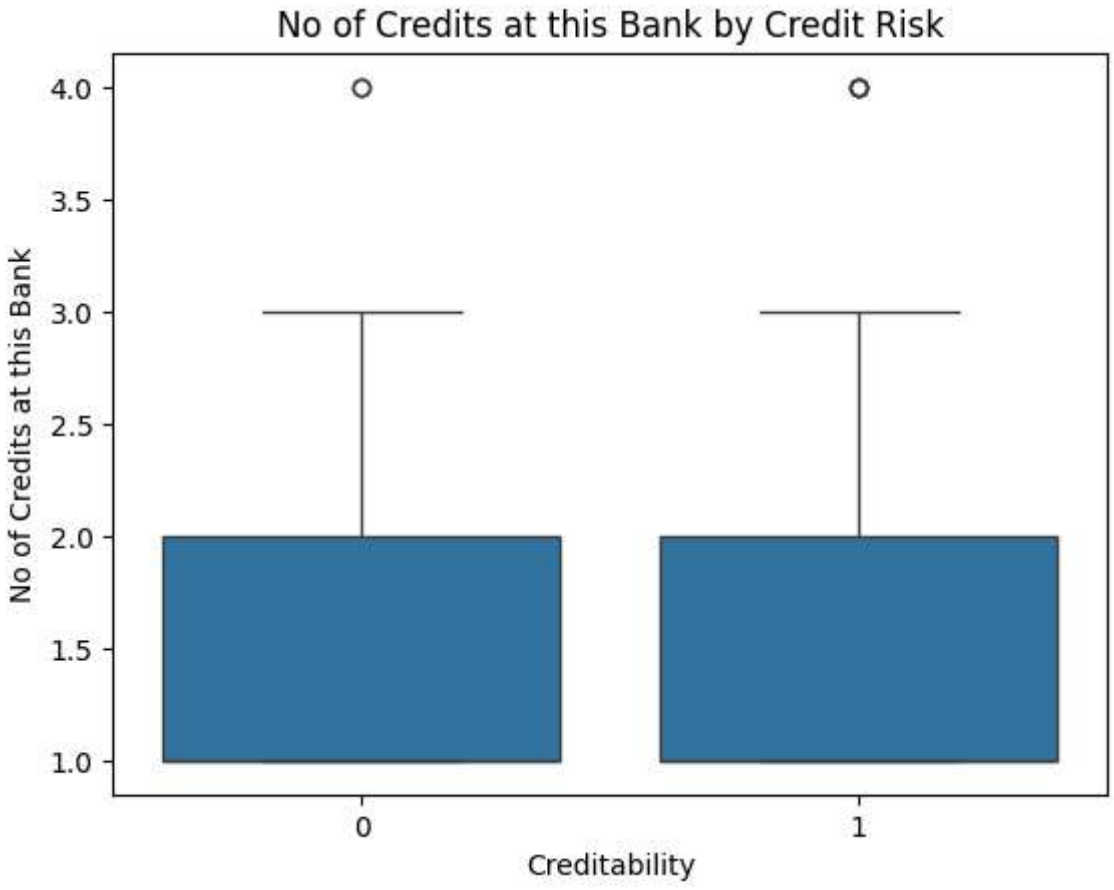

Duration in Current address by Credit Risk
Duration in Current address
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
Creditability

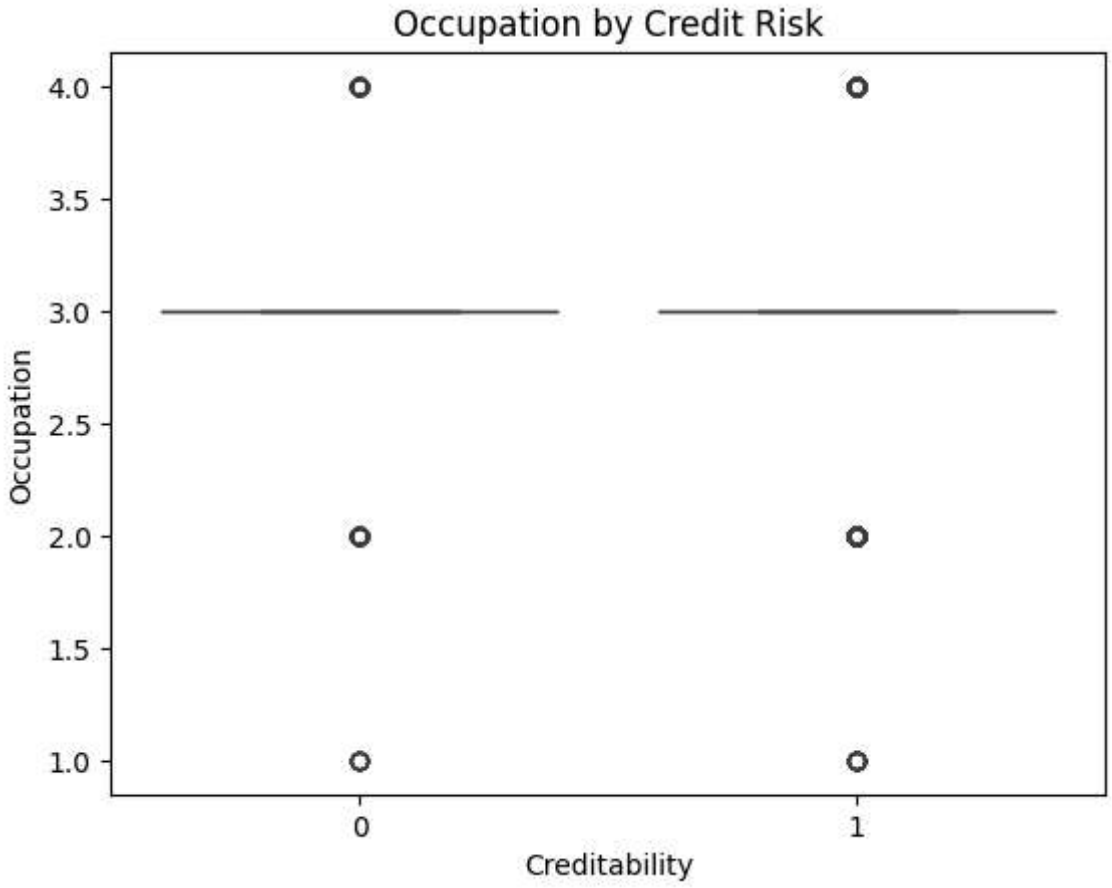

Most valuable available asset by Credit Risk
Most valuable available asset
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
Creditability

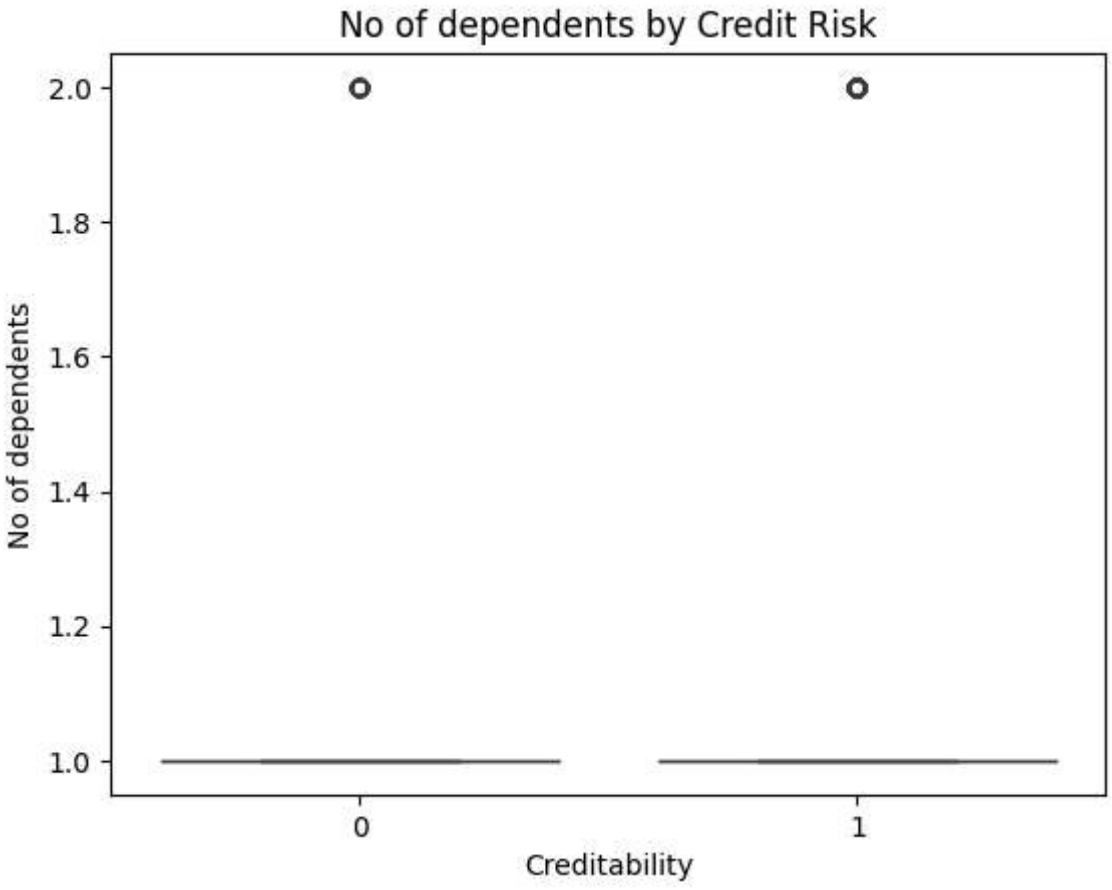

Age (years) by Credit Risk
Age (years)
70
60
50
40
30
20
0
1
Creditability

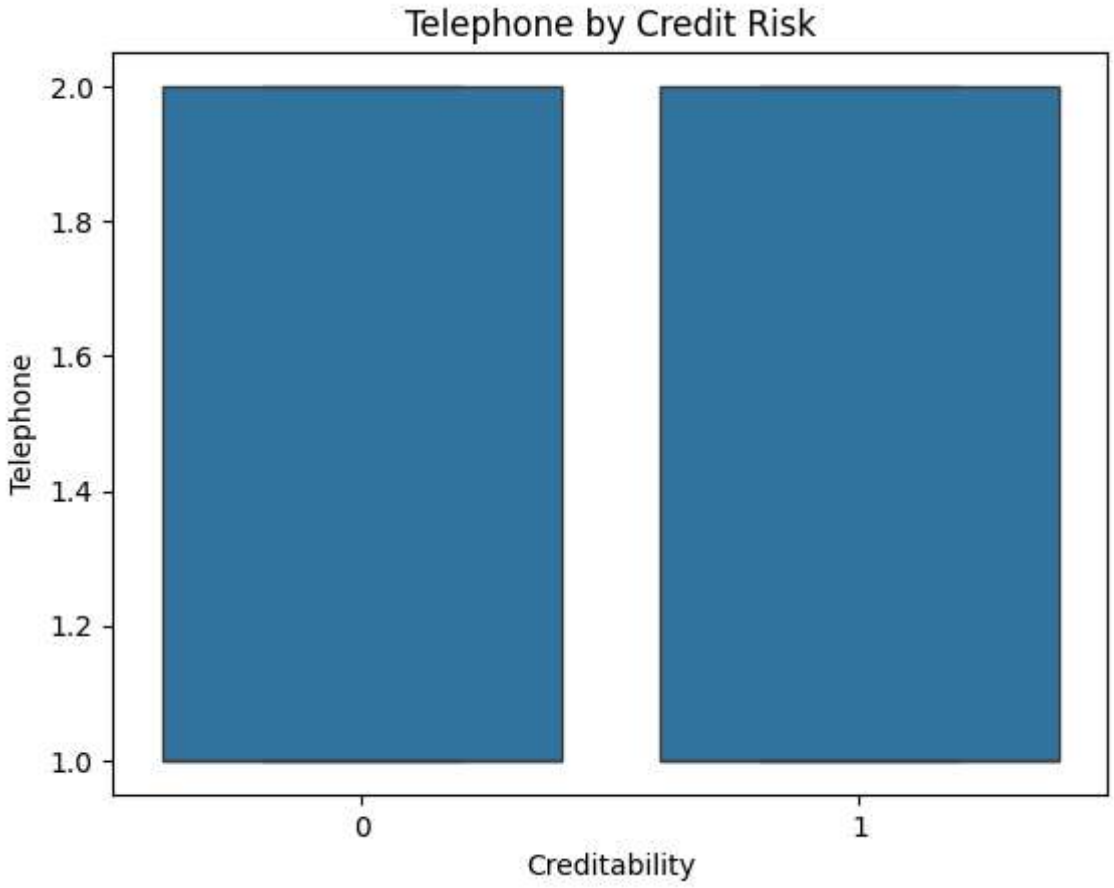

Concurrent Credits by Credit Risk
Concurrent Credits
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0
1
Creditability

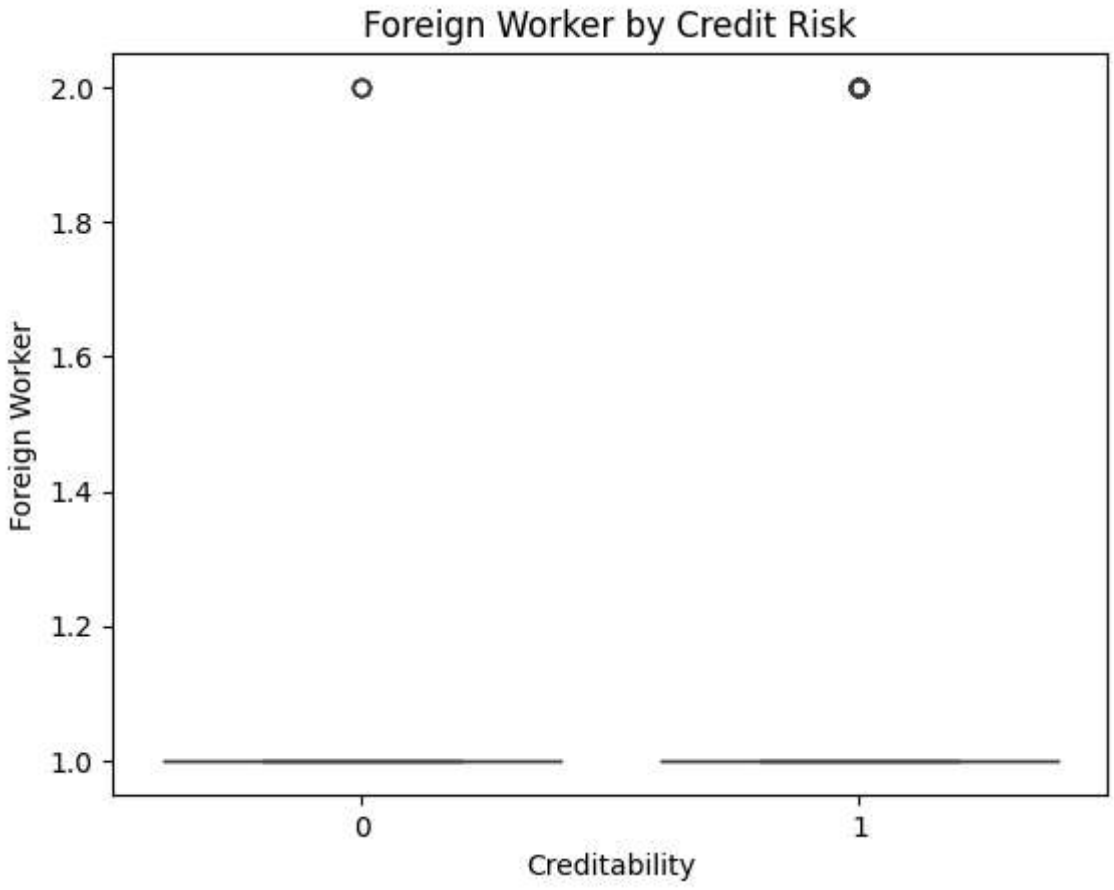

Type of apartment by Credit Risk
Type of apartment
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0
1
Creditability


No of Credits at this Bank by Credit Risk
No of Credits at this Bank
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
Creditability


Occupation by Credit Risk
Occupation
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
Creditability

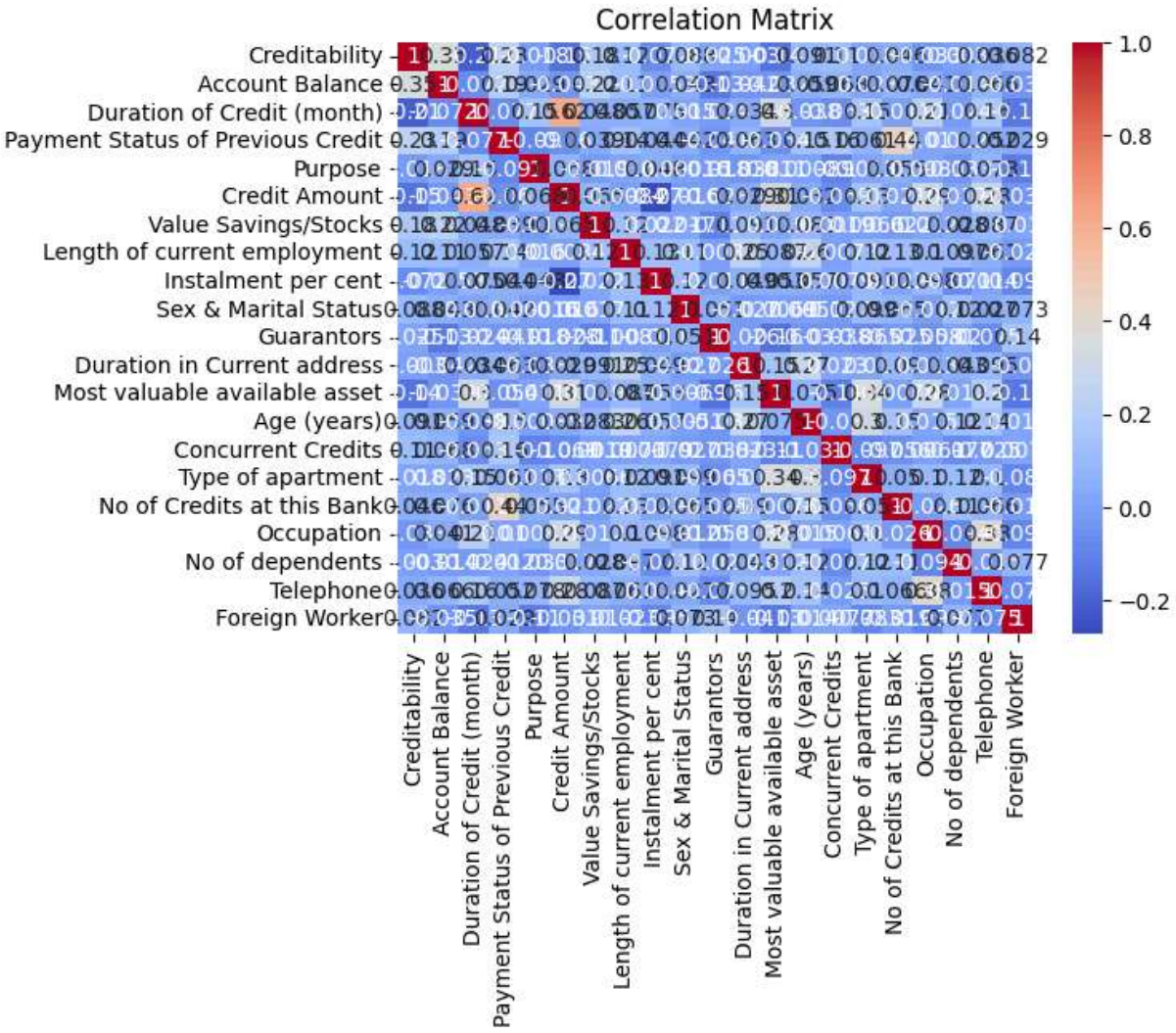

No of dependents by Credit Risk
No of dependents
2.0
1.8
1.6
1.4
1.2
1.0
0
1
Creditability


Telephone by Credit Risk
Telephone
2.0
1.8
1.6
1.4
1.2
1.0
0
1
Creditability


Foreign Worker by Credit Risk
Foreign Worker
2.0
1.8
1.6
1.4
1.2
1.0
0
1
Creditability

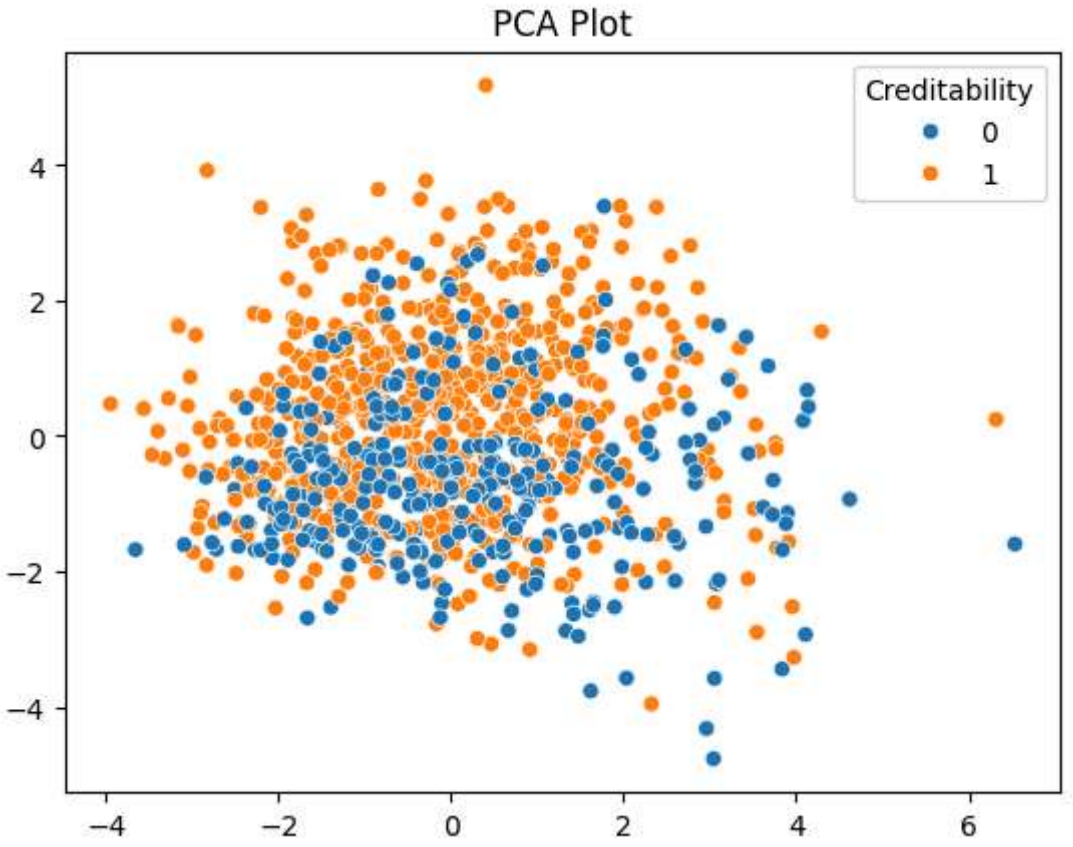
In [19]:
```
for col in df.select_dtypes(include='object').columns:
    ct = pd.crosstab(df[col], df['CreditRisk'], normalize='index')
    ct.plot(kind='bar', stacked=True)
    plt.title(f"Creditability by {col}")
    plt.ylabel("Proportion")
    plt.show()
```

In [20]:
```
import numpy as np

corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



In [23]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

X = df.drop("Creditability", axis=1).select_dtypes(include='number')
X_scaled = StandardScaler().fit_transform(X)

pca = PCA(n_components=2)
components = pca.fit_transform(X_scaled)
```

```
sns.scatterplot(x=components[:,0], y=components[:,1], hue=df['Creditability'])
plt.title("PCA Plot")
plt.show()
```



In [83]:
```
print("Class balance in full target y:")
print(y.value_counts())
```

```
Class balance in full target y:
Creditability
1    1000
Name: count, dtype: int64
```

In [85]:
```
print(df['Creditability'].describe())
```

```
count    1000.0
mean        1.0
std         0.0
min         1.0
25%         1.0
50%         1.0
75%         1.0
max         1.0
Name: Creditability, dtype: float64
```

In [87]:
```
import pandas as pd

df = pd.read_csv("C:/Users/Admin/Downloads/index_data.csv")
print(df['Creditability'].value_counts())
```

```
Creditability
1    700
0    300
Name: count, dtype: int64
```

In [88]:
```
X = df.drop(columns=['Creditability'])
y = df['Creditability']
```

In [89]:
```
X = pd.get_dummies(X, drop_first=True)
```

In [90]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [91]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42, stratify=y
)
```

In [92]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

Out[92]:

▾ LogisticRegression

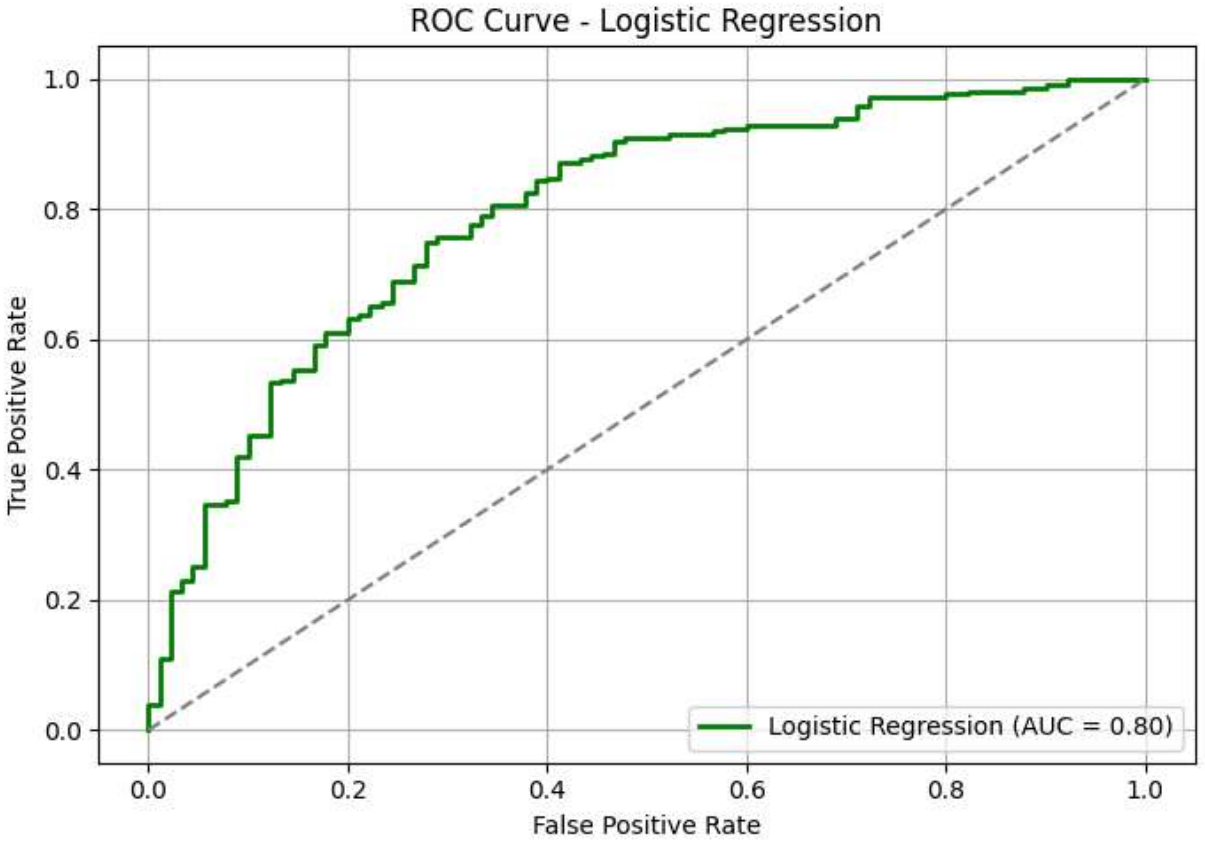
▶ Parameters

In [93]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, model.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.70      0.48      0.57        90
           1       0.80      0.91      0.86       210

    accuracy                           0.78       300
   macro avg       0.75      0.70      0.71       300
weighted avg       0.77      0.78      0.77       300
```

In [119…
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Get probability scores for the positive class
y_score_lr = model.predict_proba(X_test)[:, 1]

# Calculate False Positive Rate and True Positive Rate
fpr, tpr, _ = roc_curve(y_test, y_score_lr)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr, color='green', lw=2, label=f'Logistic Regression (AUC = {roc_auc
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [95]:
```
import pandas as pd

# Load the correct dataset (use lrgc_data.csv if you know it's clean)
df = pd.read_csv("C:/Users/Admin/Downloads/index_data.csv")

# Confirm target balance
print(df['Creditability'].value_counts())  # Expect both 0 and 1
```

```
Creditability
1    700
0    300
Name: count, dtype: int64
```

In [96]:
```
# Separate features and target
X = df.drop(columns=['Creditability'])
y = df['Creditability']

# Encode categorical variables
X = pd.get_dummies(X, drop_first=True)

# Scale numerical features
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [97]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42, stratify=y
)
```

In [98]:
```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

Out[98]:

▾ RandomForestClassifier

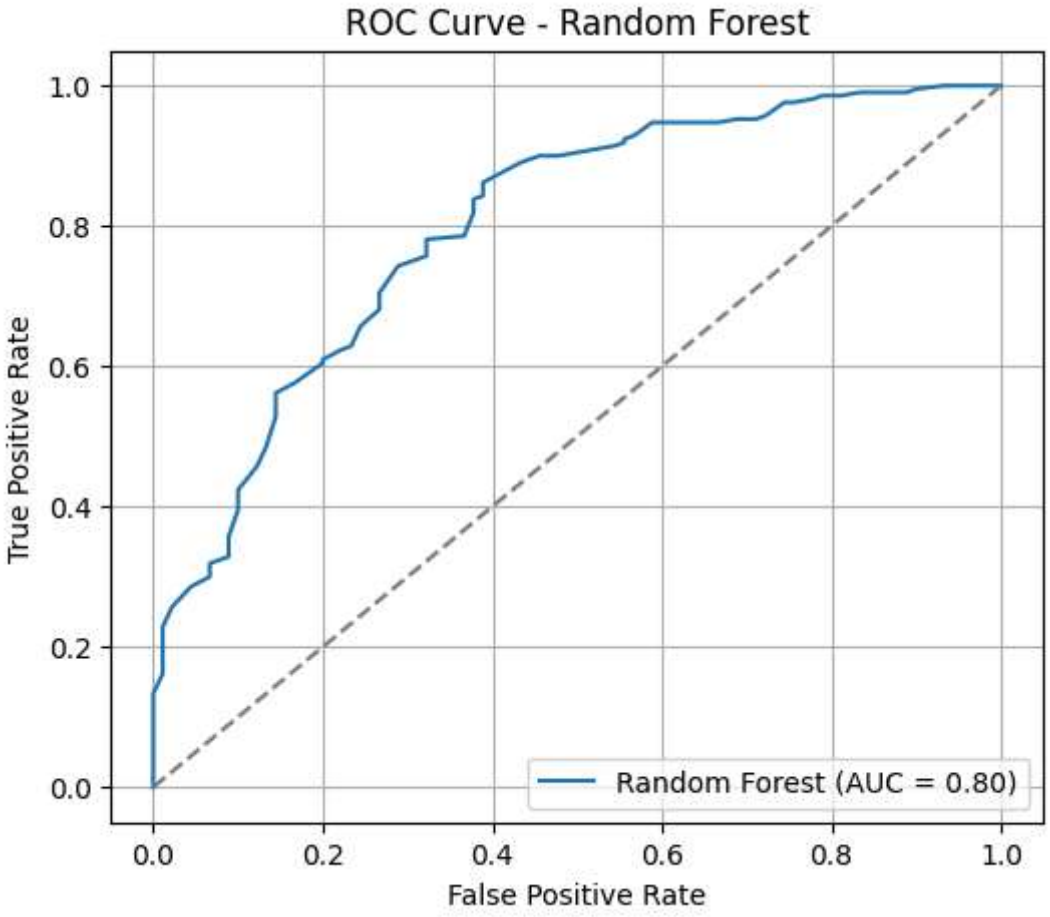
▶ Parameters

In [99]:
```
from sklearn.metrics import classification_report, confusion_matrix

y_pred_rf = rf_model.predict(X_test)

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))

print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))
```

```
Confusion Matrix:
[[ 40  50]
 [ 16 194]]

Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.44      0.55        90
           1       0.80      0.92      0.85       210

    accuracy                           0.78       300
   macro avg       0.75      0.68      0.70       300
weighted avg       0.77      0.78      0.76       300
```

In [100…
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Predict probabilities
y_score_rf = rf_model.predict_proba(X_test)[:, 1]

fpr, tpr, _ = roc_curve(y_test, y_score_rf)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, label=f'Random Forest (AUC = {roc_auc:.2f})')
```

```
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



In [101…
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from xgboost import XGBClassifier
```

In [103…
```
df = pd.read_csv("C:/Users/Admin/Downloads/index_data.csv")
```

In [104…
```
target_column = 'Creditability' if 'Creditability' in df.columns else df.columns[-1
```

In [105…
```
X = df.drop(columns=[target_column])
y = df[target_column]
```

In [106…
```
X = pd.get_dummies(X, drop_first=True)
```

In [139…
```
X = X.loc[y.notna()]
y = y.loc[y.notna()]
```

In [108…
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [109…
```
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42, stratify=y
)
```

In [110…
```
xgb_model = XGBClassifier(
    use_label_encoder=False,
    eval_metric='logloss',
    n_estimators=50,
    random_state=42
)
```

In [138…
```
xgb_model.fit(X_train, y_train)
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\xgboost\tra
ining.py:183: UserWarning: [14:23:07] WARNING: C:\actions-runner\_work\xgboost\xgboo
st\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

Out[138…

▾ XGBClassifier

▶ Parameters

In [114…
```
y_pred = xgb_model.predict(X_test)
```

In [115…
```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[ 44  46]
 [ 27 183]]

Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.49      0.55        90
           1       0.80      0.87      0.83       210

    accuracy                           0.76       300
   macro avg       0.71      0.68      0.69       300
weighted avg       0.75      0.76      0.75       300
```

In [116…
```
from sklearn.metrics import accuracy_score, classification_report

# Predictions
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Training performance
```

```
print("🔹 Training Accuracy:", accuracy_score(y_train, y_pred_train))
print("🔹 Training Report:\n", classification_report(y_train, y_pred_train))

# Testing performance
print("🔸 Testing Accuracy:", accuracy_score(y_test, y_pred_test))
print("🔸 Testing Report:\n", classification_report(y_test, y_pred_test))
```

```
🔹 Training Accuracy: 0.78
🔹 Training Report:
              precision    recall  f1-score   support

           0       0.69      0.49      0.57       210
           1       0.81      0.90      0.85       490

    accuracy                           0.78       700
   macro avg       0.75      0.70      0.71       700
weighted avg       0.77      0.78      0.77       700

🔸 Testing Accuracy: 0.7833333333333333
🔸 Testing Report:
              precision    recall  f1-score   support

           0       0.70      0.48      0.57        90
           1       0.80      0.91      0.86       210

    accuracy                           0.78       300
   macro avg       0.75      0.70      0.71       300
weighted avg       0.77      0.78      0.77       300
```

In [121…

```
print("🔹 y_train sample:")
print(y_train.head())

print("\n🔸 y_test sample:")
print(y_test.head())
```

```
🔹 y_train sample:
247    1
667    1
191    1
181    1
368    1
Name: Creditability, dtype: int64

🔸 y_test sample:
521    0
124    1
51     1
366    1
602    1
Name: Creditability, dtype: int64
```

In [122…
```
print("🔹 y_train class distribution:")
print(y_train.value_counts())

print("\n🔸 y_test class distribution:")
print(y_test.value_counts())
```

```
🔹 y_train class distribution:
Creditability
1    490
0    210
Name: count, dtype: int64

🔸 y_test class distribution:
Creditability
1    210
0     90
Name: count, dtype: int64
```

In [123…
```
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

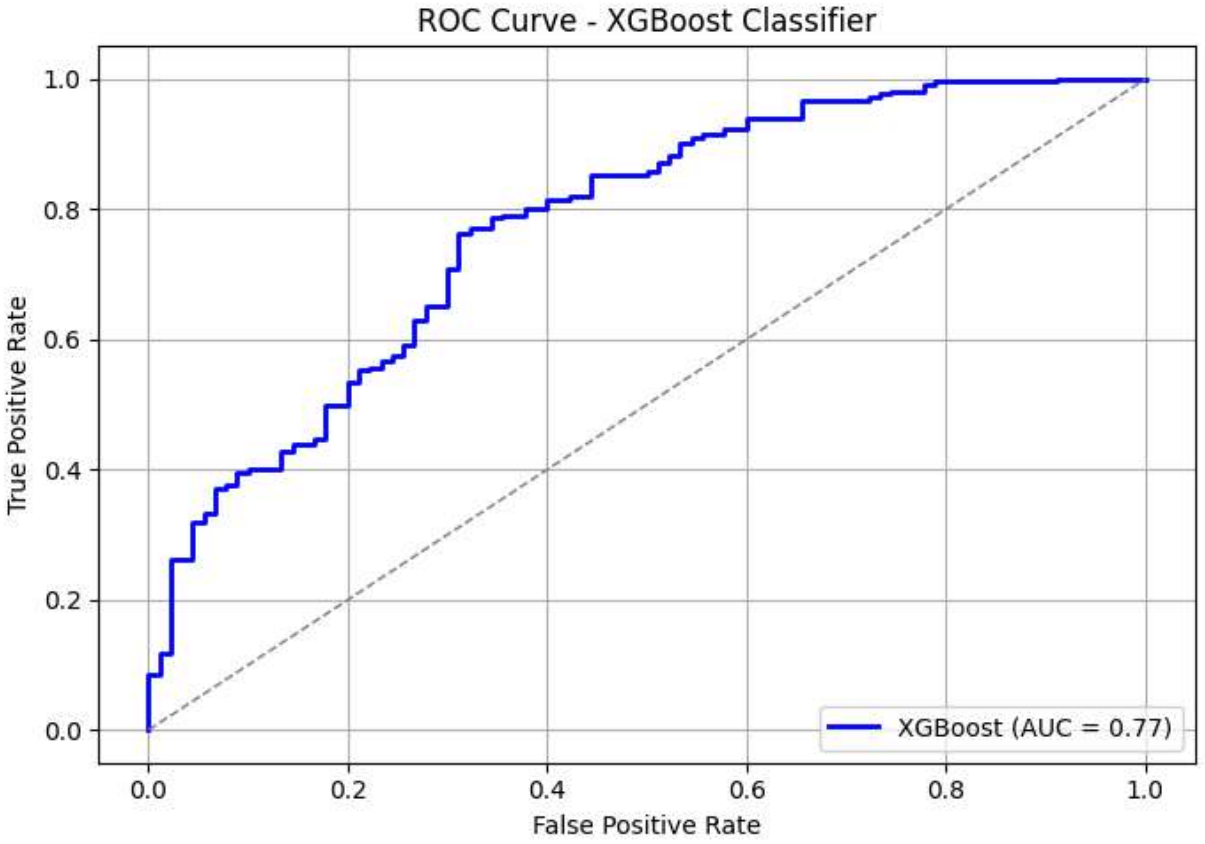
```
y_train shape: (700,)
y_test shape: (300,)
```

In [124…
```
print("y_train class distribution:")
print(y_train.value_counts())
print("\ny_test class distribution:")
print(y_test.value_counts())
```

```
y_train class distribution:
Creditability
1    490
0    210
Name: count, dtype: int64

y_test class distribution:
Creditability
1    210
0     90
Name: count, dtype: int64
```

In [117…
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Predict probabilities for test set
y_score = xgb_model.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC
fpr, tpr, _ = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'XGBoost (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('ROC Curve - XGBoost Classifier')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [135…

```
from sklearn.metrics import confusion_matrix

print("Logistic Regression")
print(confusion_matrix(y_test, y_pred_lr))

print("\nRandom Forest")
print(confusion_matrix(y_test, y_pred_rf))

print("\nXGBoost")
print(confusion_matrix(y_test, y_pred_xgb))
```

```
Logistic Regression
[[ 43  47]
 [ 18 192]]

Random Forest
[[ 40  50]
 [ 16 194]]

XGBoost
[[ 43  47]
 [ 28 182]]
```

In [126…
```
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train, y_train)
```

Out[126…

LogisticRegression

▼ Parameters

penalty	'l2'
dual	False
tol	0.0001
C	1.0
fit_intercept	True
intercept_scaling	1
class_weight	None
random_state	None
solver	'lbfgs'
max_iter	1000
multi_class	'deprecated'
verbose	0
warm_start	False
n_jobs	None
l1_ratio	None

In [134…
```
y_pred_lr = lr_model.predict(X_test)
```

In [133…
```
y_pred_xgb = xgb_model.predict(X_test)
```

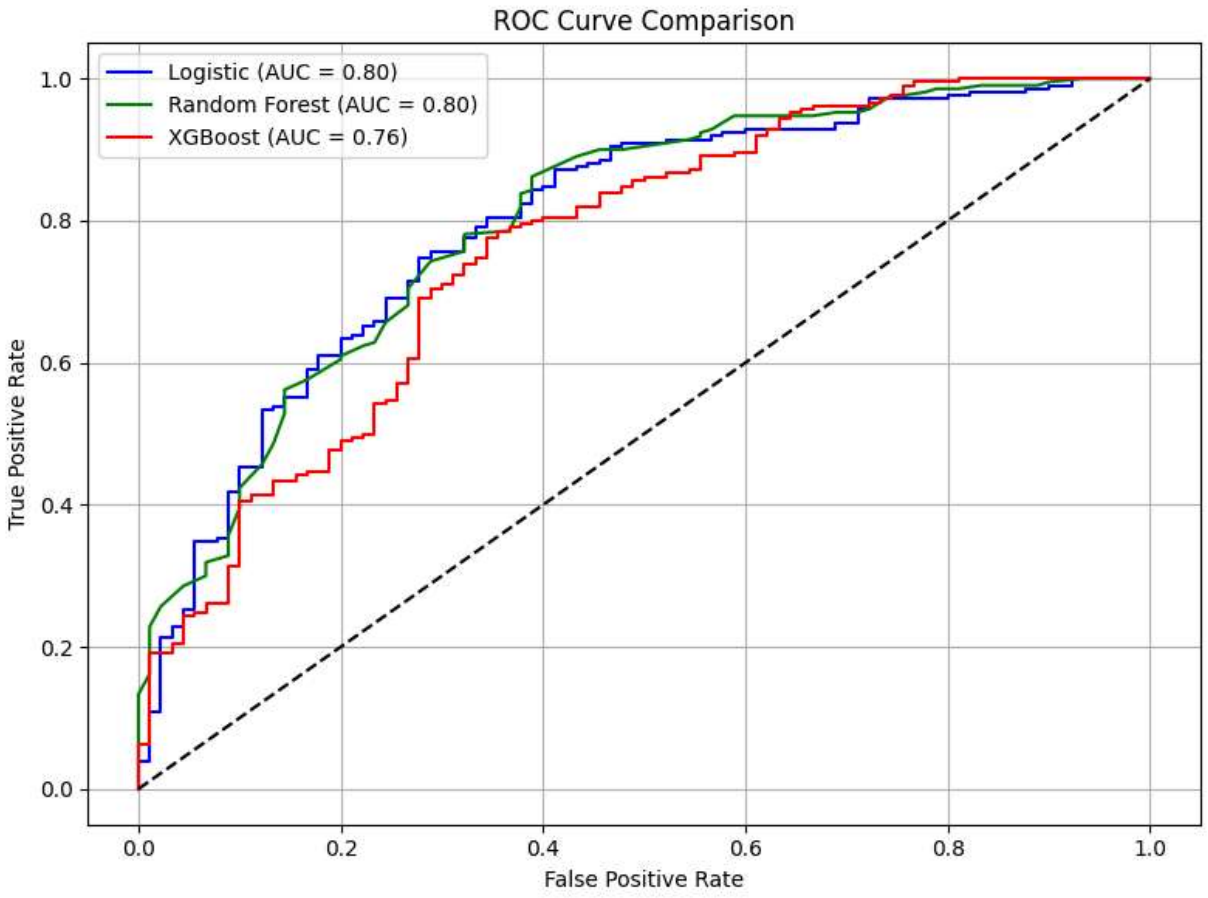
In [136…
```
from sklearn.metrics import classification_report

print("Logistic Regression:\n", classification_report(y_test, y_pred_lr))
print("Random Forest:\n", classification_report(y_test, y_pred_rf))
print("XGBoost:\n", classification_report(y_test, y_pred_xgb))
```

```
Logistic Regression:
              precision    recall  f1-score   support

           0       0.70      0.48      0.57        90
           1       0.80      0.91      0.86       210

    accuracy                           0.78       300
   macro avg       0.75      0.70      0.71       300
weighted avg       0.77      0.78      0.77       300

Random Forest:
              precision    recall  f1-score   support

           0       0.71      0.44      0.55        90
           1       0.80      0.92      0.85       210

    accuracy                           0.78       300
   macro avg       0.75      0.68      0.70       300
weighted avg       0.77      0.78      0.76       300

XGBoost:
              precision    recall  f1-score   support

           0       0.61      0.48      0.53        90
           1       0.79      0.87      0.83       210

    accuracy                           0.75       300
   macro avg       0.70      0.67      0.68       300
weighted avg       0.74      0.75      0.74       300
```

In [137…

```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Get probabilities
lr_probs = lr_model.predict_proba(X_test)[:, 1]
rf_probs = rf_model.predict_proba(X_test)[:, 1]
xgb_probs = xgb_model.predict_proba(X_test)[:, 1]

# Calculate ROC curve and AUC
fpr_lr, tpr_lr, _ = roc_curve(y_test, lr_probs)
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_probs)
fpr_xgb, tpr_xgb, _ = roc_curve(y_test, xgb_probs)

auc_lr = auc(fpr_lr, tpr_lr)
auc_rf = auc(fpr_rf, tpr_rf)
auc_xgb = auc(fpr_xgb, tpr_xgb)

# Plot
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, label=f"Logistic (AUC = {auc_lr:.2f})", color='blue')
plt.plot(fpr_rf, tpr_rf, label=f"Random Forest (AUC = {auc_rf:.2f})", color='green'
plt.plot(fpr_xgb, tpr_xgb, label=f"XGBoost (AUC = {auc_xgb:.2f})", color='red')
plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC Curve Comparison')
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [140…
```
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (700, 20)
X_test shape: (300, 20)
y_train shape: (700,)
y_test shape: (300,)
```

In [141…
```
print("y_train class distribution:\n", y_train.value_counts(normalize=True))
print("y_test class distribution:\n", y_test.value_counts(normalize=True))
```

```
y_train class distribution:
 Creditability
1    0.7
0    0.3
Name: proportion, dtype: float64
y_test class distribution:
 Creditability
1    0.7
0    0.3
Name: proportion, dtype: float64
```

In [142…

```
import pandas as pd
X_train_df = pd.DataFrame(X_train)
X_test_df = pd.DataFrame(X_test)

print("\nX_train summary:\n", X_train_df.describe())
print("\nX_test summary:\n", X_test_df.describe())
```

X_train summary:

	0	1	2	3	4	5
count	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000
mean	-0.011251	-0.003212	-0.020454	-0.024790	-0.035779	-0.012212
std	1.007111	0.998880	0.993524	0.994291	0.969851	0.997579
min	-1.254566	-1.402415	-2.350870	-1.030963	-1.070856	-0.699707
25%	-1.254566	-0.738668	-0.503428	-0.666407	-0.683894	-0.699707
50%	-0.459026	-0.240857	-0.503428	-0.301852	-0.360909	-0.699707
75%	1.132053	0.256953	1.344014	0.062704	0.210539	0.566731
max	1.132053	3.243815	1.344014	2.614592	4.492104	1.833169

	6	7	8	9	10	11
count	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000
mean	-0.009226	-0.021847	-0.016956	0.037400	0.011007	0.003266
std	1.009840	1.006961	0.992341	1.049240	1.012296	0.996993
min	-1.973997	-1.764514	-2.376626	-0.303686	-1.672459	-1.293723
25%	-0.317959	-0.870183	-0.963650	-0.303686	-0.765977	-1.293723
50%	-0.317959	0.024147	0.449326	-0.303686	0.140505	-0.341055
75%	1.338078	0.918477	0.449326	-0.303686	1.046987	0.611613
max	1.338078	0.918477	1.862303	3.885083	1.046987	1.564281

	12	13	14	15	16	17
count	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000
mean	0.013924	-0.011141	0.017253	-0.017073	-0.001749	0.005921
std	1.016168	1.008277	1.002401	0.988305	1.005136	1.006330
min	-1.457831	-2.375050	-1.751205	-0.704926	-2.914492	-0.428290
25%	-0.752799	0.460831	0.135869	-0.704926	0.146949	-0.428290
50%	-0.224024	0.460831	0.135869	-0.704926	0.146949	-0.428290
75%	0.569138	0.460831	0.135869	1.027079	0.146949	-0.428290
max	3.477397	0.460831	2.022944	4.491089	1.677670	2.334869

	18	19
count	700.000000	700.000000
mean	-0.011063	0.038597
std	0.998485	1.090667
min	-0.823318	-0.196014
25%	-0.823318	-0.196014
50%	-0.823318	-0.196014
75%	1.214598	-0.196014
max	1.214598	5.101669

X_test summary:

	0	1	2	3	4	5
count	300.000000	300.000000	300.000000	300.000000	300.000000	300.000000
mean	0.026253	0.007495	0.047726	0.057843	0.083485	0.028495
std	0.986069	1.005904	1.016664	1.014156	1.065616	1.008381
min	-1.254566	-1.402415	-2.350870	-1.030963	-1.061640	-0.699707
25%	-0.657911	-0.738668	-0.503428	-0.666407	-0.649159	-0.699707
50%	-0.459026	-0.240857	-0.503428	0.062704	-0.297110	-0.699707
75%	1.132053	0.568085	1.344014	0.062704	0.399634	0.566731
max	1.132053	4.239436	1.344014	2.614592	5.370764	1.833169

	6	7	8	9	10	11
count	300.000000	300.000000	300.000000	300.000000	300.000000	300.000000
mean	0.021528	0.050977	0.039563	-0.087266	-0.025684	-0.007621
std	0.979690	0.985040	1.019862	0.871709	0.973600	1.010269

```
min     -1.973997   -1.764514   -2.376626   -0.303686   -1.672459   -1.293723
25%     -0.317959   -0.870183   -0.963650   -0.303686   -0.765977   -1.293723
50%     -0.317959    0.024147    0.449326   -0.303686    0.140505   -0.341055
75%      1.338078    0.918477    0.449326   -0.303686    1.046987    0.611613
max      1.338078    0.918477    1.862303    3.885083    1.046987    1.564281

               12          13          14          15          16          17  \
count  300.000000  300.000000  300.000000  300.000000  300.000000  300.000000
mean    -0.032490    0.025996   -0.040258    0.039836    0.004082   -0.013816
std      0.963830    0.983290    0.996556    1.028967    0.991256    0.988293
min     -1.369702   -2.375050   -1.751205   -0.704926   -2.914492   -0.428290
25%     -0.752799    0.460831    0.135869   -0.704926    0.146949   -0.428290
50%     -0.224024    0.460831    0.135869   -0.704926    0.146949   -0.428290
75%      0.481009    0.460831    0.135869    1.027079    0.146949   -0.428290
max      3.477397    0.460831    2.022944    4.491089    1.677670    2.334869

               18          19
count  300.000000  300.000000
mean     0.025814   -0.090061
std      1.006385    0.742915
min     -0.823318   -0.196014
25%     -0.823318   -0.196014
50%     -0.823318   -0.196014
75%      1.214598   -0.196014
max      1.214598    5.101669
```

In [143…

```python
from sklearn.metrics import accuracy_score

train_preds = model.predict(X_train)
test_preds = model.predict(X_test)

print("Train Accuracy:", accuracy_score(y_train, train_preds))
print("Test Accuracy :", accuracy_score(y_test, test_preds))
```

```
Train Accuracy: 0.78
Test Accuracy : 0.7833333333333333
```

In [ ]: