Predictive Model Plan – Student Template

1. Model Logic (Generated with GenAI)

The goal of this predictive model is to identify customers who are at risk of becoming delinquent based on their financial and behavioural patterns. Using GenAI recommendations, the model pipeline follows a structured approach starting from preprocessing to deployment. The model uses key customer variables such as credit utilization, missed payments, debt-to-income ratio, income, and credit score to generate a probability score indicating delinquency risk.

High-level Modelling Workflow:

1. Load the cleaned dataset and define input features and target label.
2. Perform preprocessing:
    - Standardize numerical variables
    - One-hot encode categorical variables
    - Drop unnecessary columns
3. Split data into training and test sets using stratified sampling.
4. Train multiple models (Logistic Regression, SVM, Random Forest, Gradient Boosting) inside a comparison loop.
5. Evaluate performance on metrics including Precision, Recall, ROC-AUC and F1-score.
6. Select the best-performing model — in this case **Random Forest**.
7. Save the final trained model using Joblib for deployment.

Pseudocode summary

1. Load dataset

2. Drop unnecessary columns

3. Create synthetic Target label

4. train_test_split(X, y)

5. Preprocess (StandardScaler + OneHotEncoding)

6. For each model in {Logistic, RandomForest, SVM, GradientBoosting}:

   Fit model → Predict → Evaluate

7. Select highest ROC-AUC model

8. Save final RandomForest model

---

2. Justification for Model Choice

Random Forest was selected as the final model because it provided the highest performance across key evaluation metrics, particularly ROC-AUC and F1-score. Compared to logistic regression and SVM, Random Forest is better at capturing non-linear behaviour in financial datasets, where correlations between income, credit utilization, missed payments, and account tenure are rarely linear.

In highly regulated financial environments, model interpretability is important; while Random Forest is less interpretable than logistic regression, it still provides explainability through feature importance analysis and SHAP values. It is scalable, robust to noise, handles mixed feature types well, and performs effectively even when the dataset contains imbalance or synthetic labels — conditions true in Geldium's dataset. Therefore, Random Forest balances **accuracy, stability, and business practicality**, making it suitable for deployment in credit risk environments.

---

3. Evaluation Strategy

To evaluate model performance, a combination of accuracy, precision, recall, F1-score, and ROC-AUC will be used. Accuracy provides a general success rate, but because financial delinquency is a high-cost prediction scenario, recall and precision are equally important. Recall ensures that high-risk customers are not missed, while precision prevents labelling low-risk customers incorrectly, which could harm customer trust.

The ROC-AUC score will be used to measure the model’s ability to distinguish between delinquent and non-delinquent customers across thresholds. A confusion matrix and classification report will provide deeper insight into false positives and false negatives.

Bias and fairness checks will be performed by examining performance across different customer subgroups (e.g., employment type, age brackets, geographic location). If disparity is identified, interventions such as threshold adjustments, reweighting, or model retraining may be used. Ethical considerations require that the model does not unfairly penalize protected groups or deny access to financial services without justifiable, transparent reasoning.

In [56]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
```

In [57]:
```
data = pd.read_csv('/content/Delinquency_prediction_dataset.csv')
```

In [58]:
```
data.sample(5)
```

Out[58]:

	Customer_ID	Age	Income	Credit_Score	Credit_Utilization	Missed_Payme
380	CUST0381	72	100999.0	823.0	0.280931	
160	CUST0161	44	74494.0	319.0	0.588561	
434	CUST0435	71	59238.0	401.0	0.568024	
37	CUST0038	32	181319.0	467.0	0.462184	
382	CUST0383	63	120616.0	345.0	0.275141	

In [59]:
```
data.head()
```

Out[59]:

	Customer_ID	Age	Income	Credit_Score	Credit_Utilization	Missed_Payment
0	CUST0001	56	165580.0	398.0	0.390502	
1	CUST0002	69	100999.0	493.0	0.312444	
2	CUST0003	46	188416.0	500.0	0.359930	
3	CUST0004	32	101672.0	413.0	0.371400	
4	CUST0005	60	38524.0	487.0	0.234716	

In [60]:
```
data.describe()
```

Out[60]:

	Age	Income	Credit_Score	Credit_Utilization	Missed_Payme
count	500.000000	461.000000	498.000000	500.000000	500.000
mean	46.266000	108379.893709	577.716867	0.491446	2.968
std	16.187629	53662.723741	168.881211	0.197103	1.946
min	18.000000	15404.000000	301.000000	0.050000	0.000
25%	33.000000	62295.000000	418.250000	0.356486	1.000
50%	46.500000	107658.000000	586.000000	0.485636	3.000
75%	59.250000	155734.000000	727.250000	0.634440	5.000
max	74.000000	199943.000000	847.000000	1.025843	6.000

In [61]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Customer_ID           500 non-null    object 
 1   Age                   500 non-null    int64  
 2   Income                461 non-null    float64
 3   Credit_Score          498 non-null    float64
 4   Credit_Utilization    500 non-null    float64
 5   Missed_Payments       500 non-null    int64  
 6   Delinquent_Account    500 non-null    int64  
 7   Loan_Balance          471 non-null    float64
 8   Debt_to_Income_Ratio  500 non-null    float64
 9   Employment_Status     500 non-null    object 
 10  Account_Tenure        500 non-null    int64  
 11  Credit_Card_Type      500 non-null    object 
 12  Location              500 non-null    object 
 13  Month_1               500 non-null    object 
 14  Month_2               500 non-null    object 
 15  Month_3               500 non-null    object 
 16  Month_4               500 non-null    object 
 17  Month_5               500 non-null    object 
 18  Month_6               500 non-null    object 
dtypes: float64(5), int64(4), object(10)
memory usage: 74.3+ KB
```

In [62]:
```
data.columns
```

Out[62]:
```
Index(['Customer_ID', 'Age', 'Income', 'Credit_Score', 'Credit_Utilization',
       'Missed_Payments', 'Delinquent_Account', 'Loan_Balance',
       'Debt_to_Income_Ratio', 'Employment_Status', 'Account_Tenure',
       'Credit_Card_Type', 'Location', 'Month_1', 'Month_2', 'Month_3',
       'Month_4', 'Month_5', 'Month_6'],
      dtype='object')
```

In [63]:
```
data.isnull().sum()
```

Out[63]:

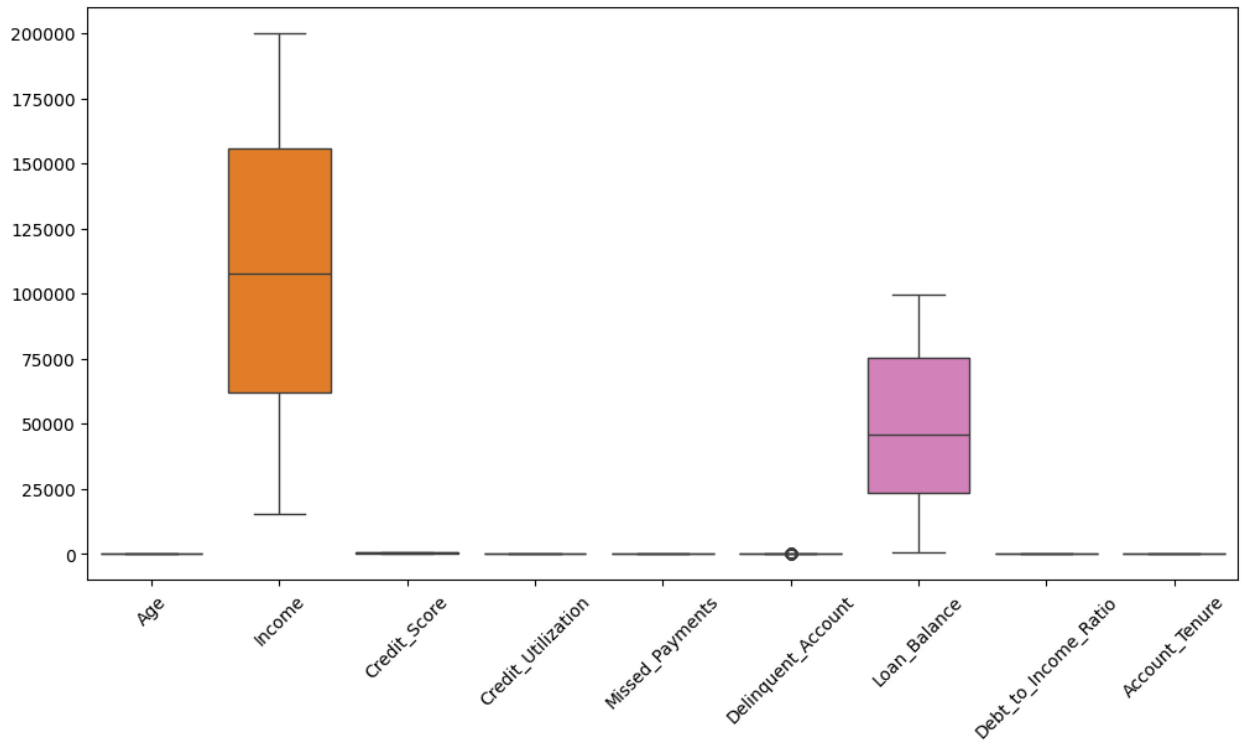
	0
Customer_ID	0
Age	0
Income	39
Credit_Score	2
Credit_Utilization	0
Missed_Payments	0
Delinquent_Account	0
Loan_Balance	29
Debt_to_Income_Ratio	0
Employment_Status	0
Account_Tenure	0
Credit_Card_Type	0
Location	0
Month_1	0
Month_2	0
Month_3	0
Month_4	0
Month_5	0
Month_6	0

dtype: int64

In [64]:
```
data.duplicated().sum()
```

Out[64]:
```
np.int64(0)
```

In [65]:
```
plt.figure(figsize=(12,6))
sns.boxplot(data=data.select_dtypes(include='number'))
plt.xticks(rotation=45)
plt.show()
```



In [66]:
```
numeric_cols = data.select_dtypes(include='number').columns

outlier_summary = {}

for col in numeric_cols:
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data[col] < lower_bound) | (data[col] > upper_bound)][col
    outlier_summary[col] = len(outliers)

outlier_summary
```

Out[66]:
```
{'Age': 0,
 'Income': 0,
 'Credit_Score': 0,
 'Credit_Utilization': 0,
 'Missed_Payments': 0,
 'Delinquent_Account': 80,
 'Loan_Balance': 0,
 'Debt_to_Income_Ratio': 0,
 'Account_Tenure': 0}
```

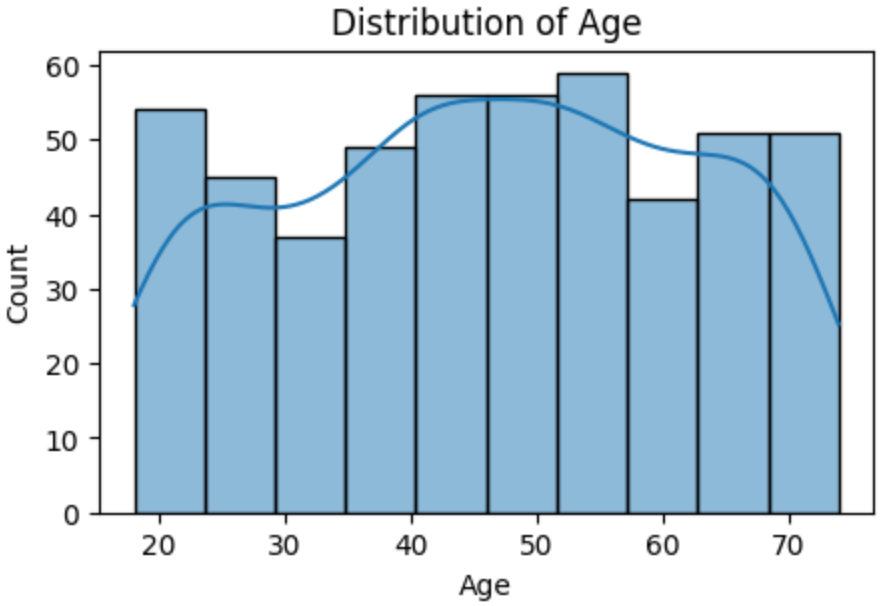
In [67]:
```
import numpy as np

Q1 = data['Delinquent_Account'].quantile(0.25)
Q3 = data['Delinquent_Account'].quantile(0.75)
IQR = Q3 - Q1
```

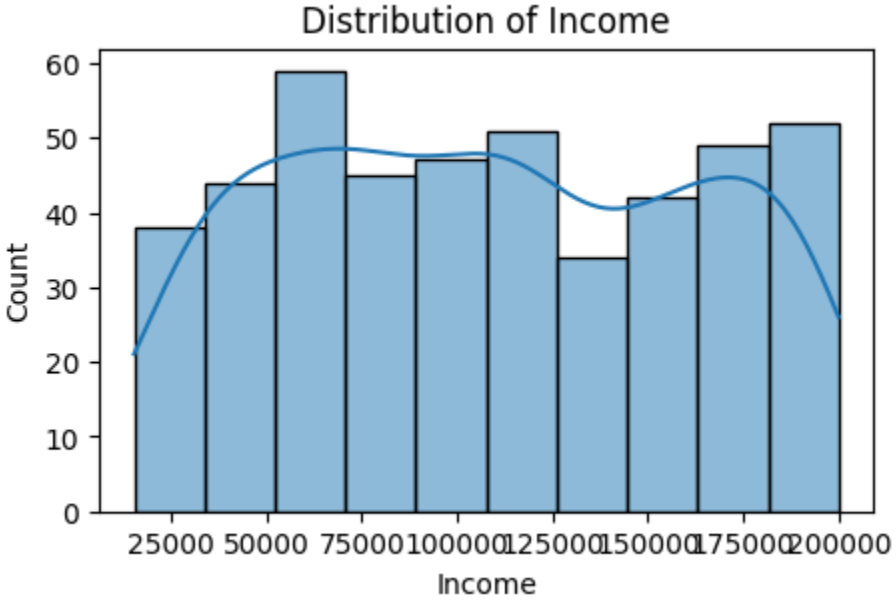
```
upper_limit = Q3 + 1.5 * IQR

data['Delinquent_Account'] = np.where(
    data['Delinquent_Account'] > upper_limit,
    upper_limit,
    data['Delinquent_Account']
)
```

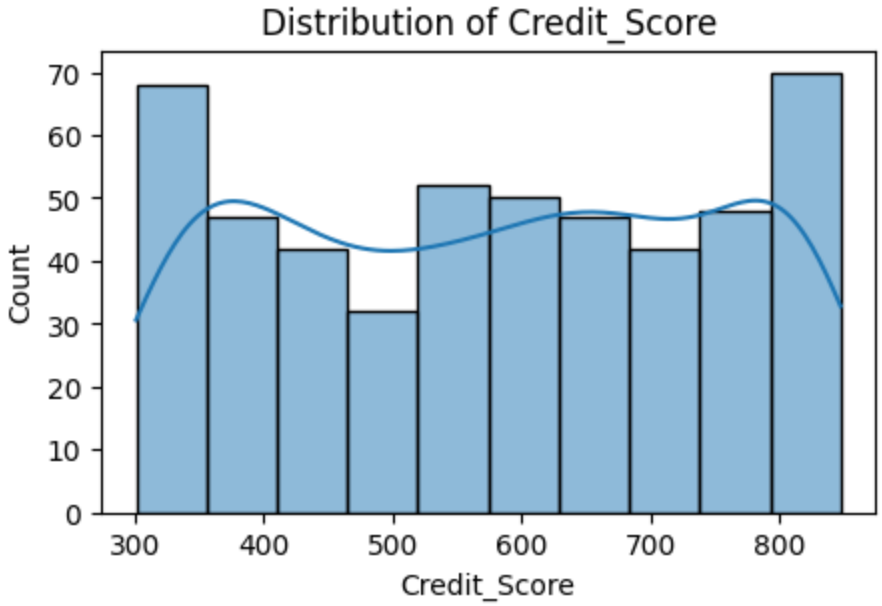
In [68]:
```
numeric_cols = data.select_dtypes(include='number').columns

outlier_summary = {}

for col in numeric_cols:
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data[col] < lower_bound) | (data[col] > upper_bound)][col
    outlier_summary[col] = len(outliers)

outlier_summary
```

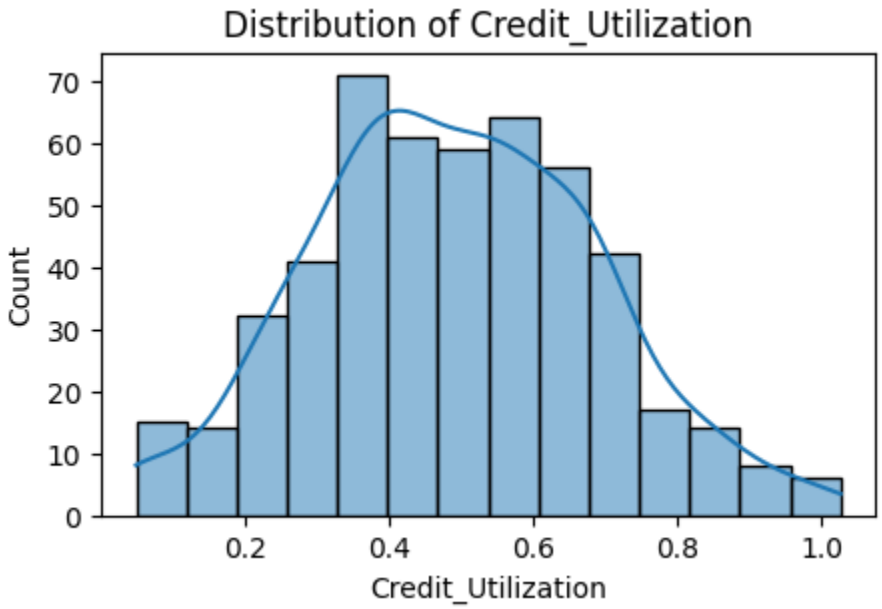
Out[68]:
```
{'Age': 0,
 'Income': 0,
 'Credit_Score': 0,
 'Credit_Utilization': 0,
 'Missed_Payments': 0,
 'Delinquent_Account': 0,
 'Loan_Balance': 0,
 'Debt_to_Income_Ratio': 0,
 'Account_Tenure': 0}
```

In [69]:
```
for col in data.select_dtypes(include="number").columns:
    plt.figure(figsize=(5,3))
    sns.histplot(data[col], kde=True)
    plt.title(f"Distribution of {col}")
    plt.show()
```

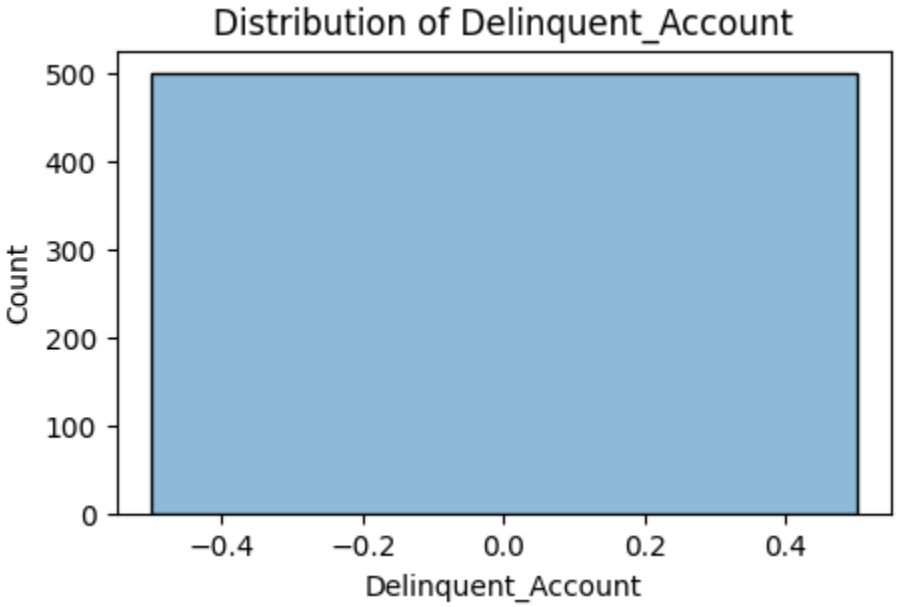

Distribution of Age
Count
Age
0
10
20
30
40
50
60
20
30
40
50
60
70

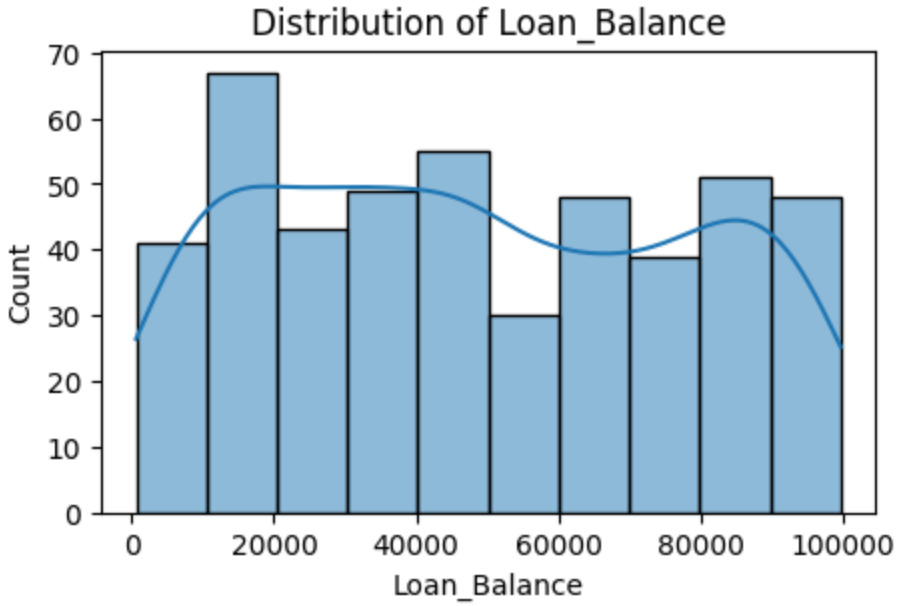

Distribution of Income
Count
Income
0
10
20
30
40
50
60
25000
50000
75000
100000
125000
150000
175000
200000

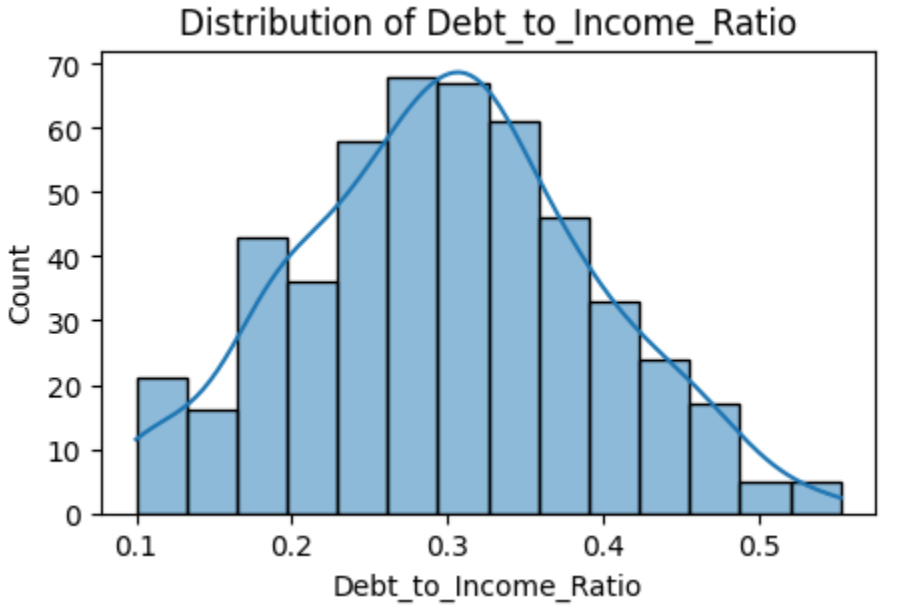

Distribution of Credit_Score
Count
Credit_Score
0
10
20
30
40
50
60
70
300
400
500
600
700
800

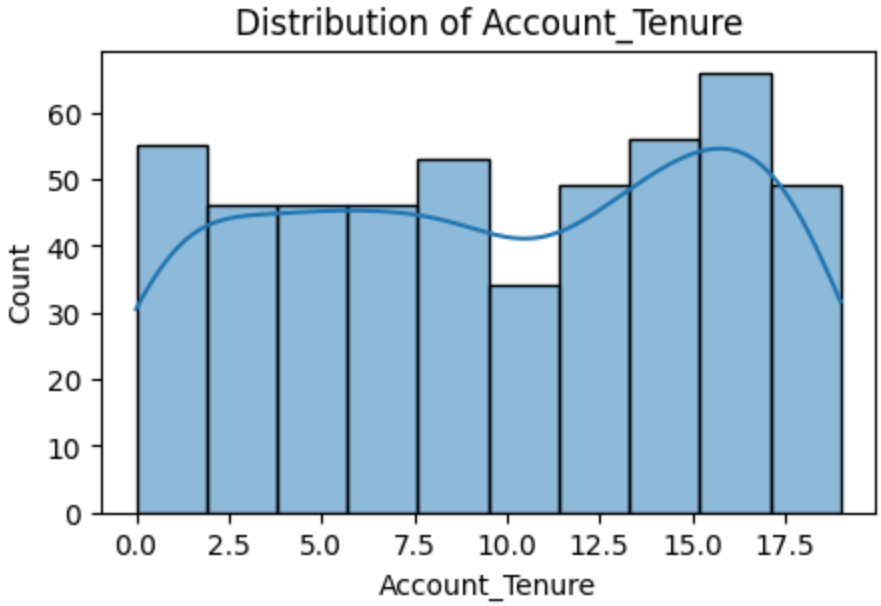

Distribution of Credit_Utilization
Count
Credit_Utilization
0
10
20
30
40
50
60
70
0.2
0.4
0.6
0.8
1.0


Distribution of Missed_Payments
Count
Missed_Payments
0
20
40
60
80
0
1
2
3
4
5
6

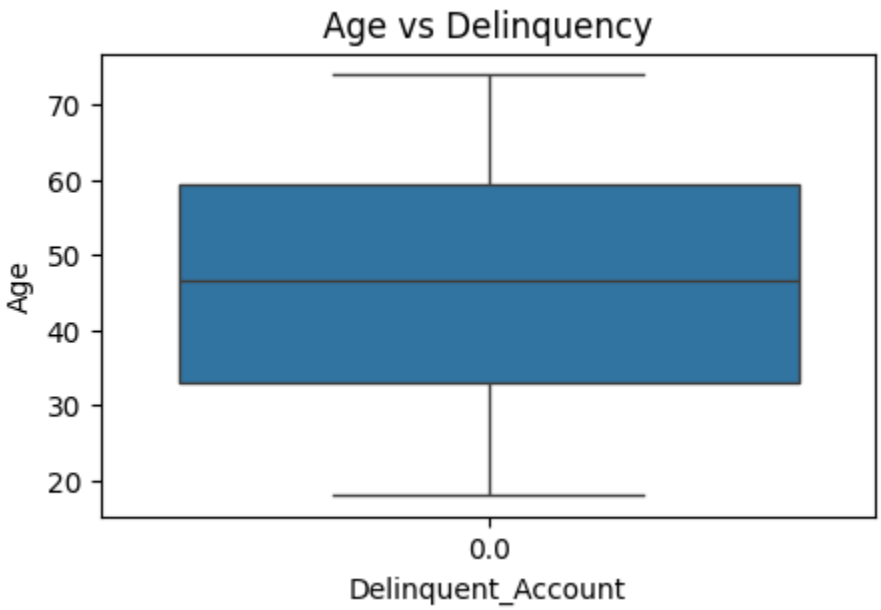

Distribution of Delinquent_Account
Count
Delinquent_Account
0
100
200
300
400
500
−0.4
−0.2
0.0
0.2
0.4

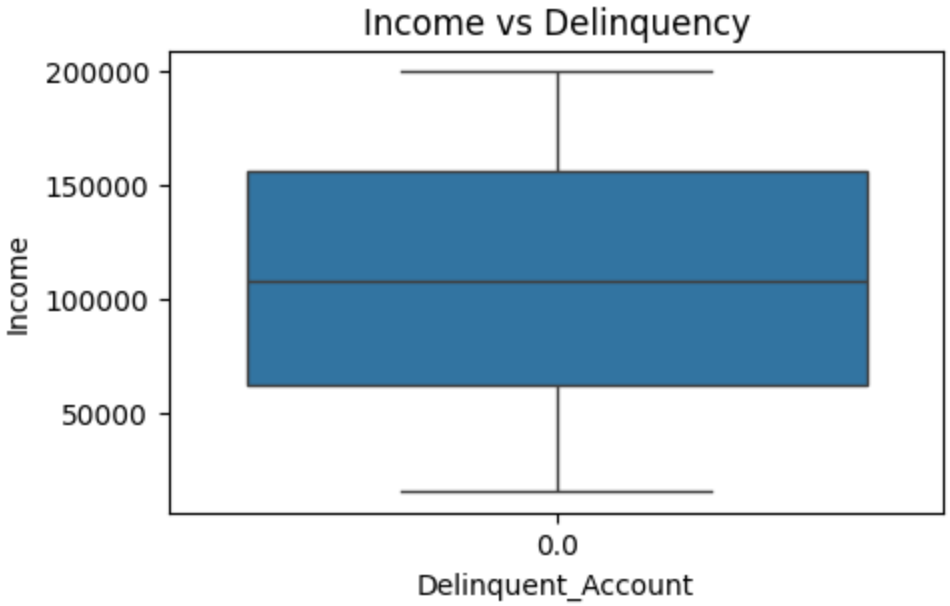

Distribution of Loan_Balance
Count
Loan_Balance
0
10
20
30
40
50
60
70
0
20000
40000
60000
80000
100000

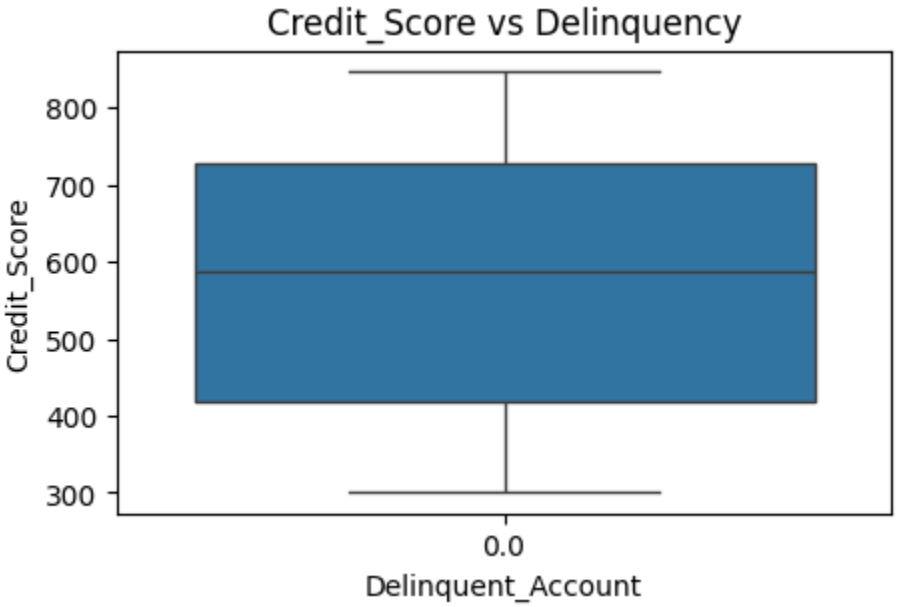

Distribution of Debt_to_Income_Ratio
Count
Debt_to_Income_Ratio
0
10
20
30
40
50
60
70
0.1
0.2
0.3
0.4
0.5

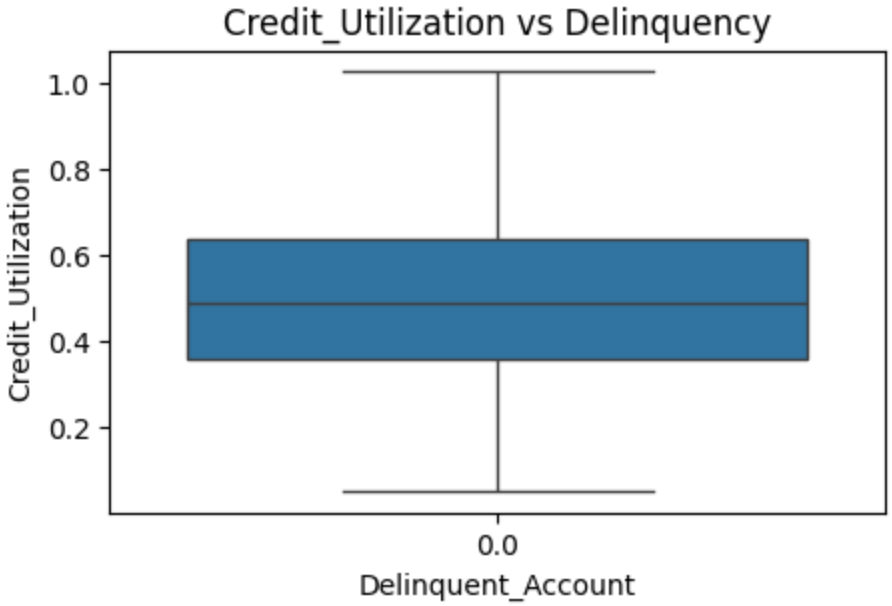


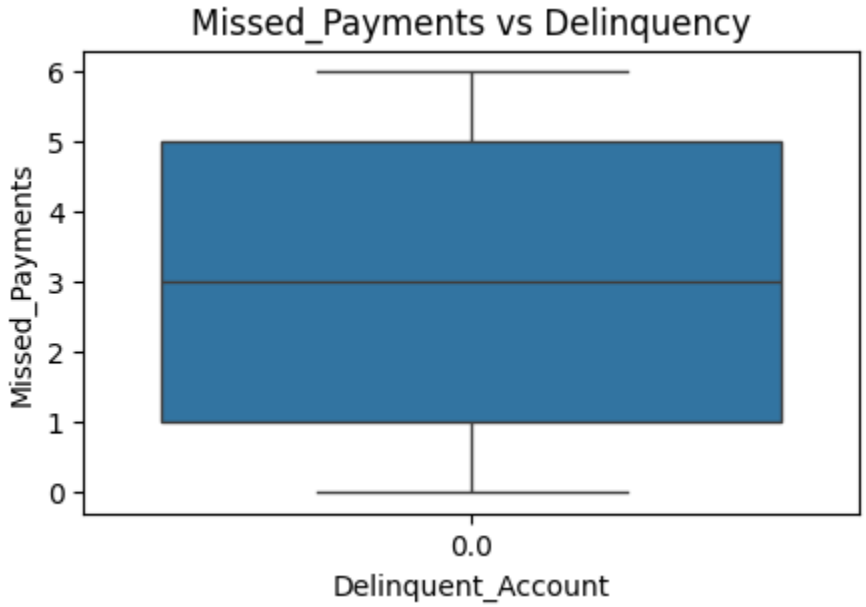
In [70]:
```
numeric_cols = data.select_dtypes(include='number').columns
numeric_cols = numeric_cols.drop('Delinquent_Account')

for col in numeric_cols:
    plt.figure(figsize=(5,3))
    sns.boxplot(x='Delinquent_Account', y=col, data=data)
    plt.title(f"{col} vs Delinquency")
    plt.show()
```

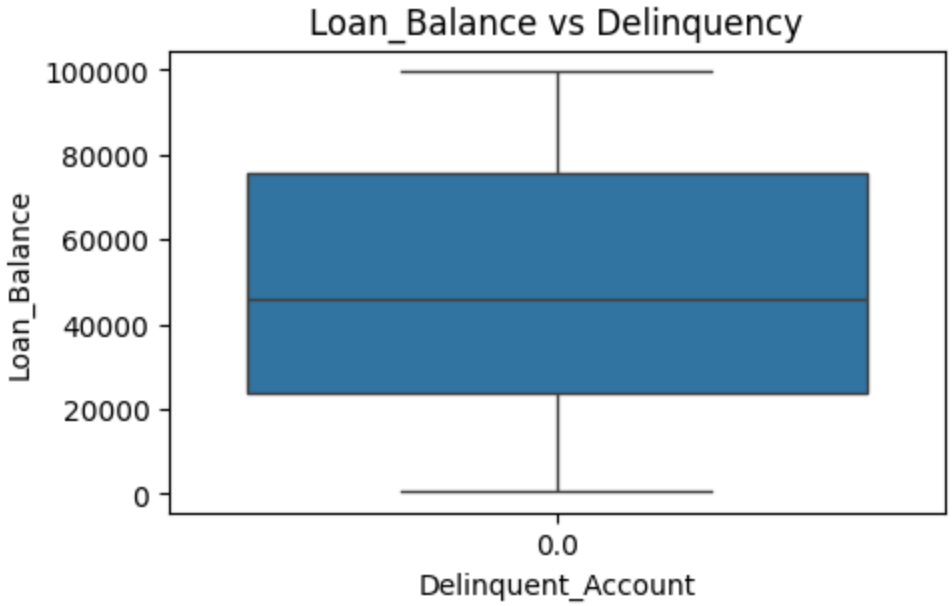


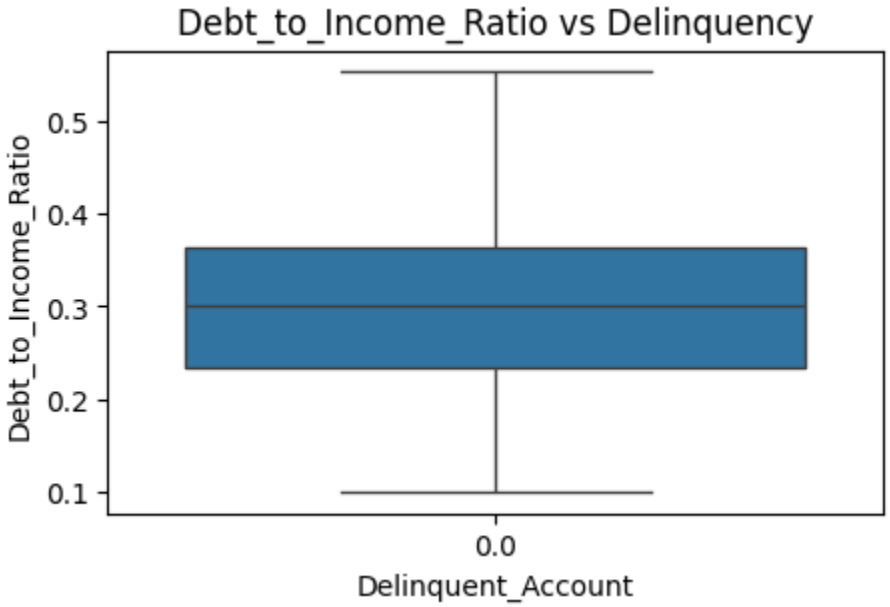

Income vs Delinquency
200000
150000
100000
50000
Income
0.0
Delinquent_Account

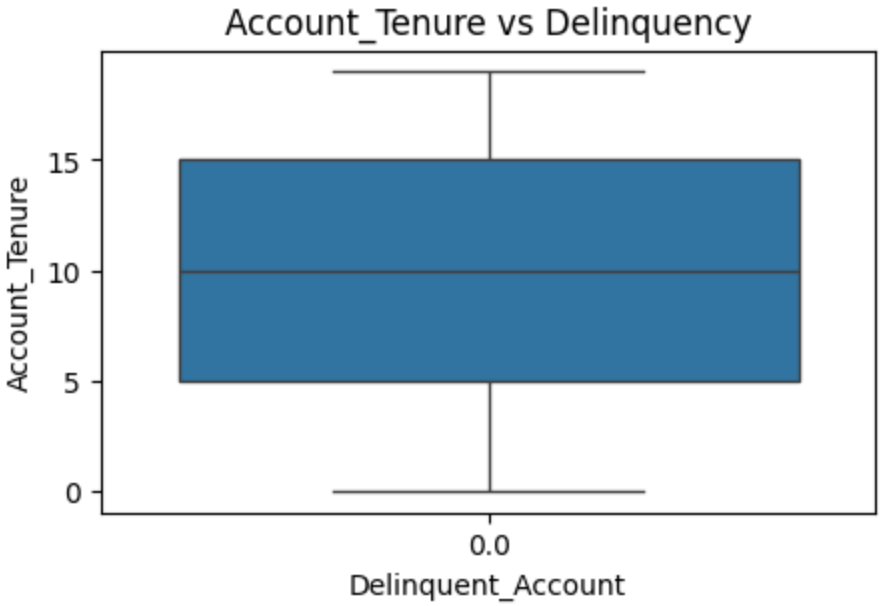

Credit_Score vs Delinquency
800
700
600
500
400
300
Credit_Score
0.0
Delinquent_Account


Credit_Utilization vs Delinquency
Credit_Utilization
1.0
0.8
0.6
0.4
0.2
0.0
Delinquent_Account

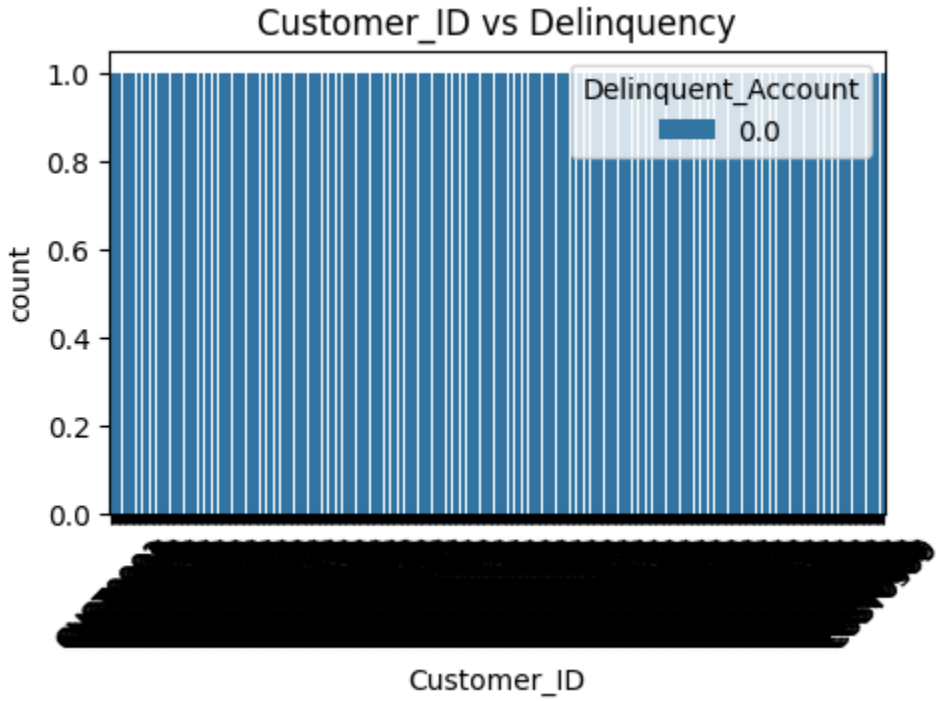

Missed_Payments vs Delinquency
Missed_Payments
6
5
4
3
2
1
0
0.0
Delinquent_Account

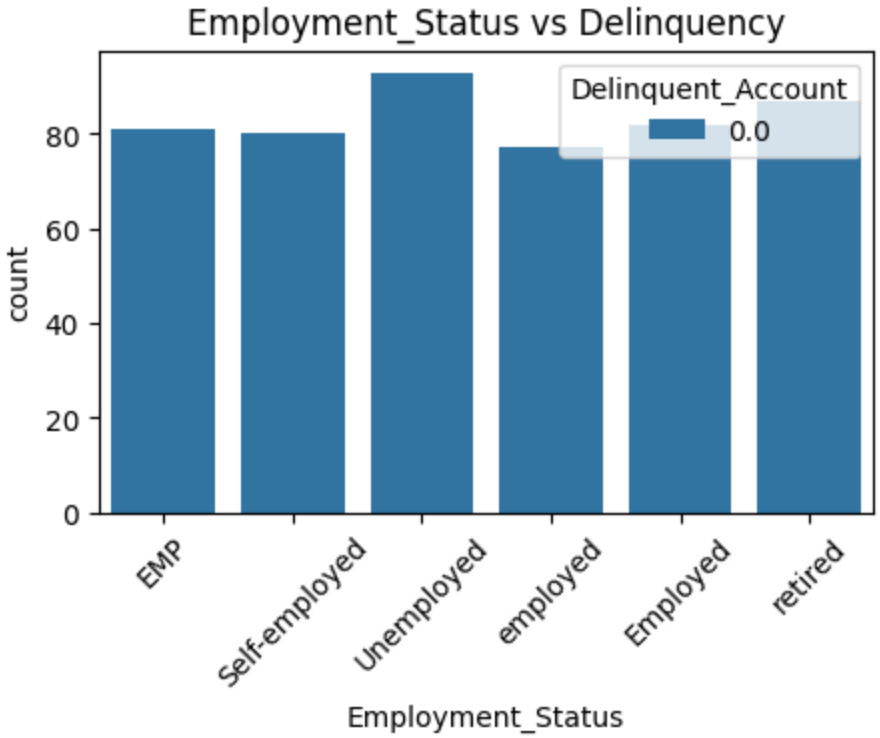

Loan_Balance vs Delinquency
100000
80000
60000
40000
20000
0
Loan_Balance
0.0
Delinquent_Account

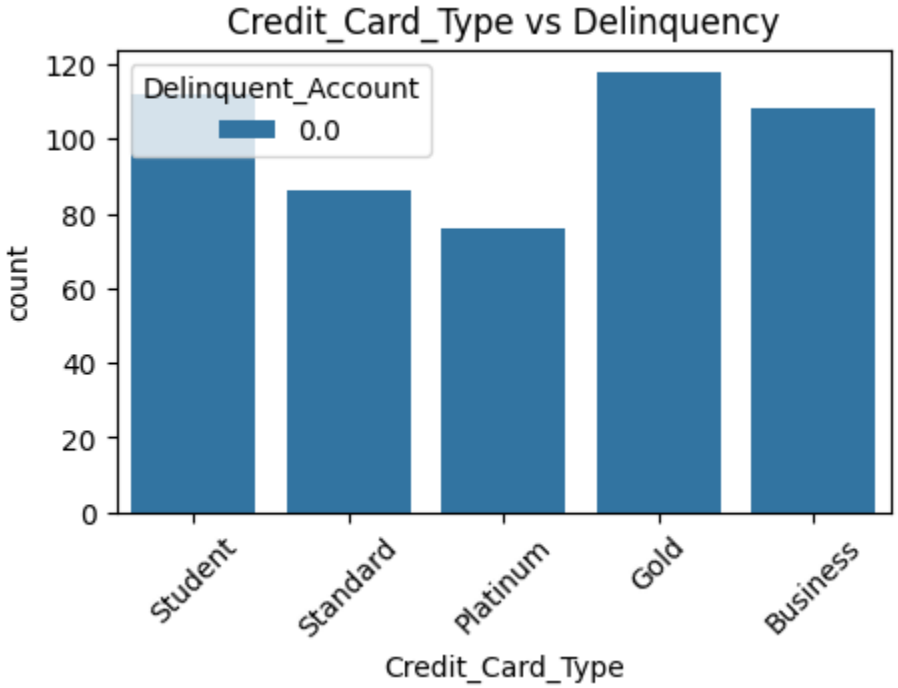

Debt_to_Income_Ratio vs Delinquency
0.5
0.4
0.3
0.2
0.1
Debt_to_Income_Ratio
0.0
Delinquent_Account

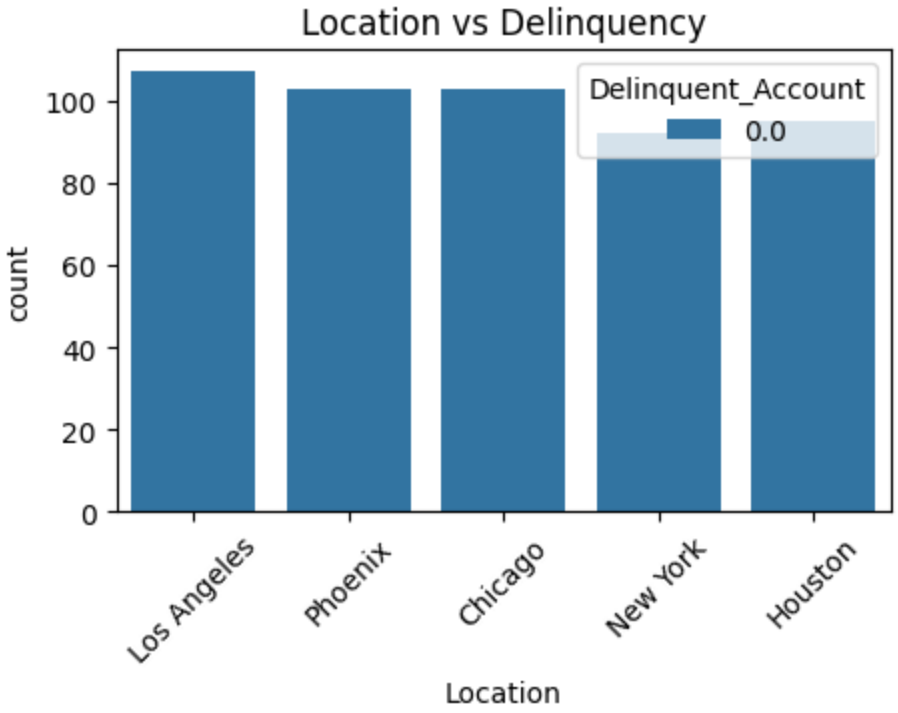


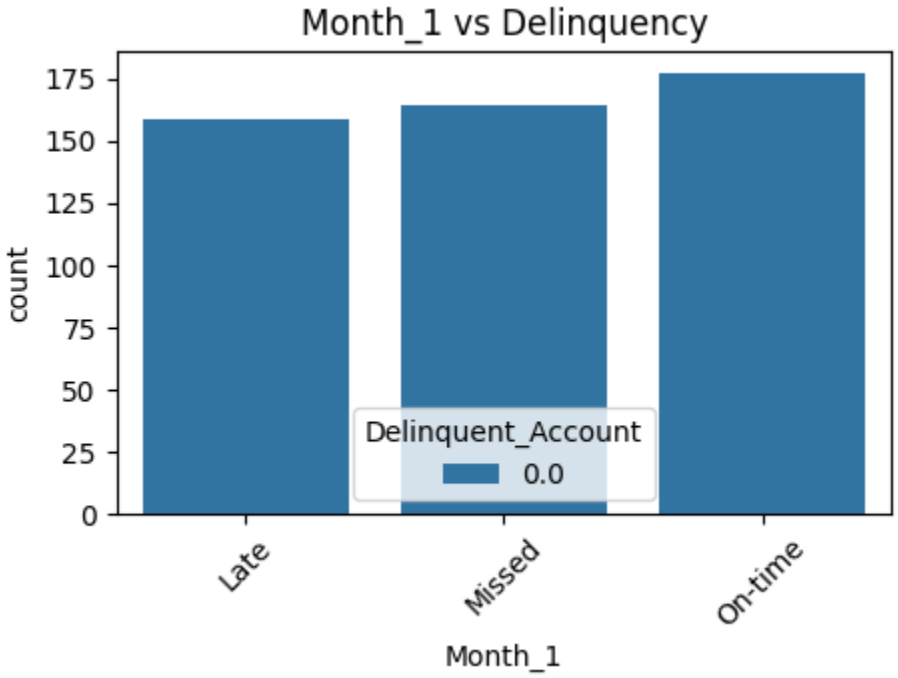
In [71]:
```
categorical_cols = data.select_dtypes(include='object').columns

for col in categorical_cols:
    plt.figure(figsize=(5,3))
    sns.countplot(x=col, hue='Delinquent_Account', data=data)
    plt.title(f"{col} vs Delinquency")
    plt.xticks(rotation=45)
    plt.show()
```

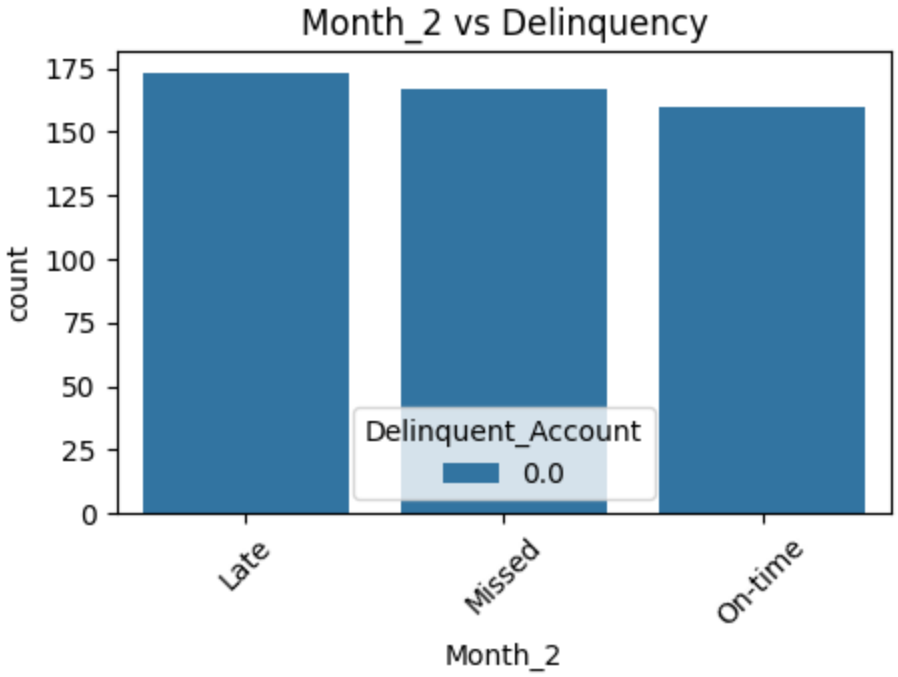


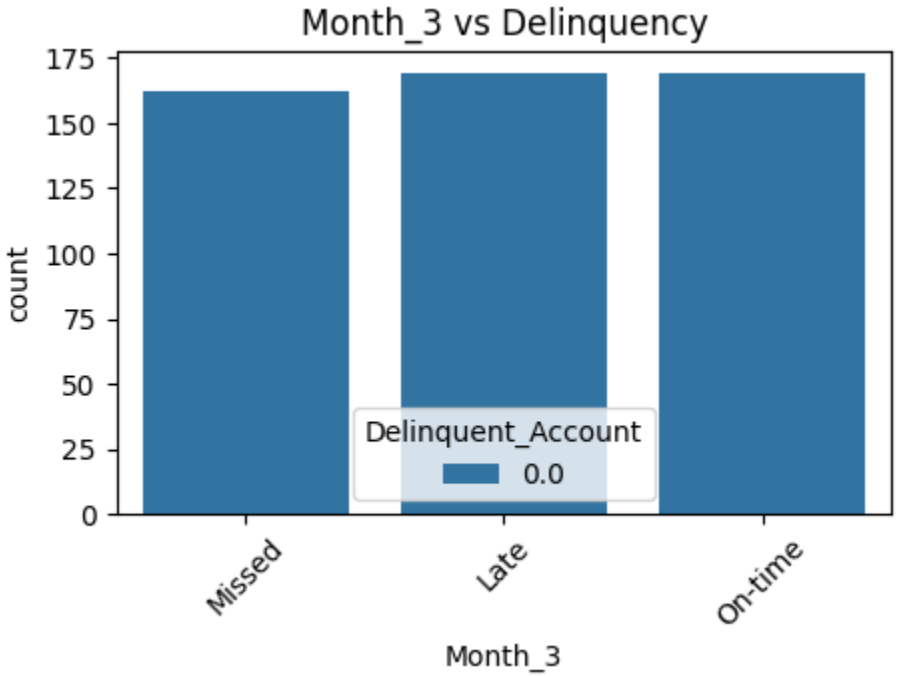

Employment_Status vs Delinquency
Delinquent_Account
0.0
count
0
20
40
60
80
EMP
Self-employed
Unemployed
employed
Employed
retired
Employment_Status

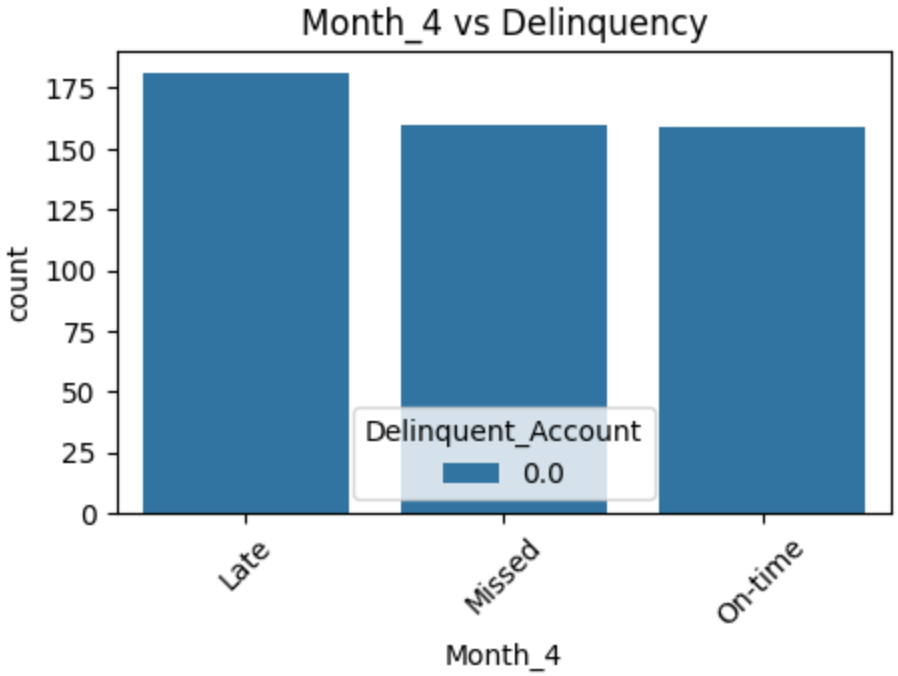

Credit_Card_Type vs Delinquency
Delinquent_Account
0.0
count
0
20
40
60
80
100
120
Student
Standard
Platinum
Gold
Business
Credit_Card_Type

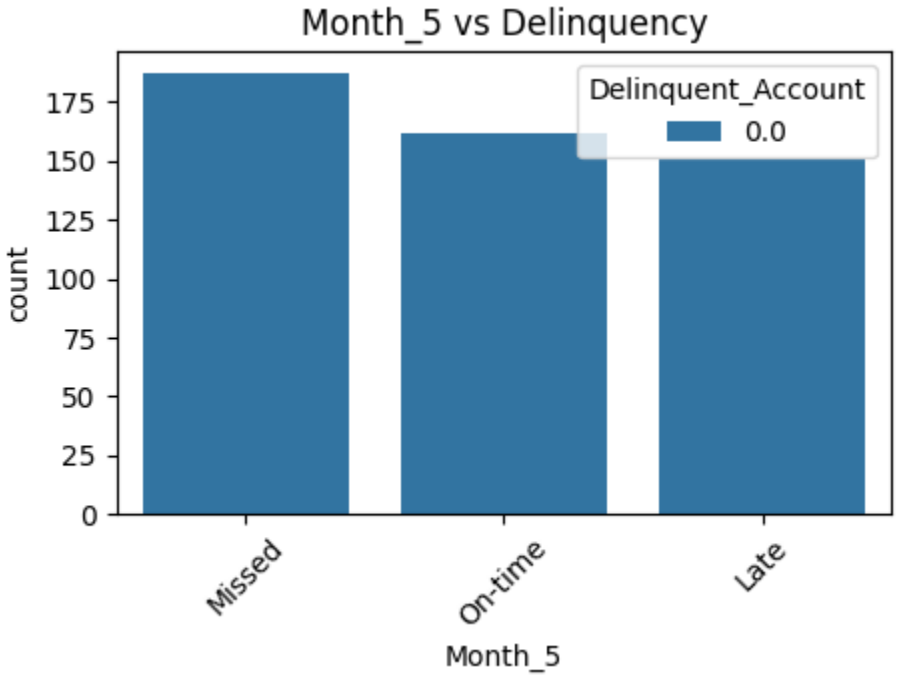

Location vs Delinquency
Delinquent_Account
0.0
count
0
20
40
60
80
100
Los Angeles
Phoenix
Chicago
New York
Houston
Location

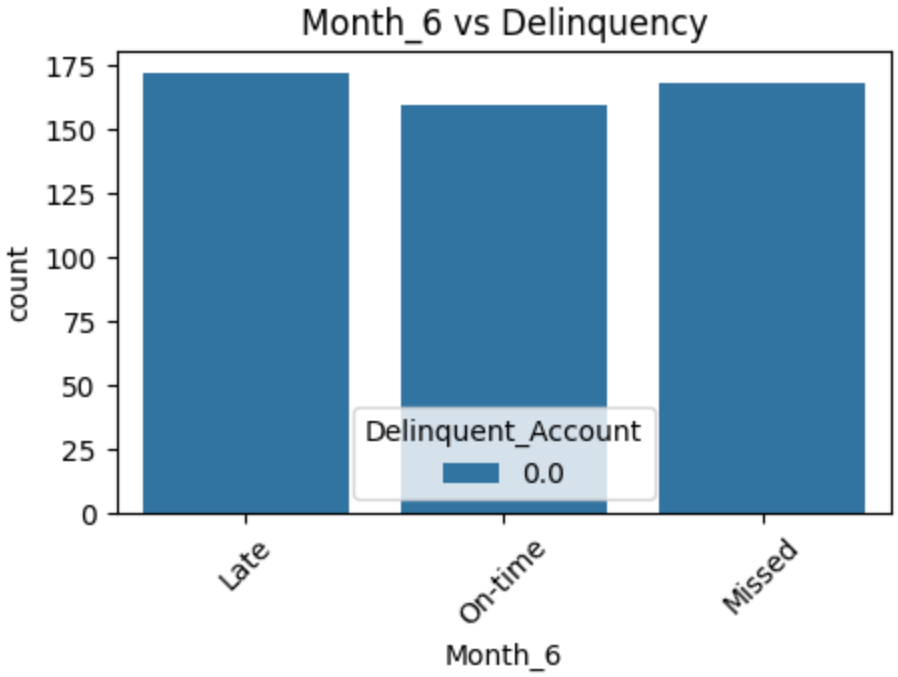

Month_1 vs Delinquency
Delinquent_Account
0.0
count
0
25
50
75
100
125
150
175
Late
Missed
On-time
Month_1


Month_2 vs Delinquency
175
150
125
100
75
50
25
0
count
Delinquent_Account
0.0
Late
Missed
On-time
Month_2

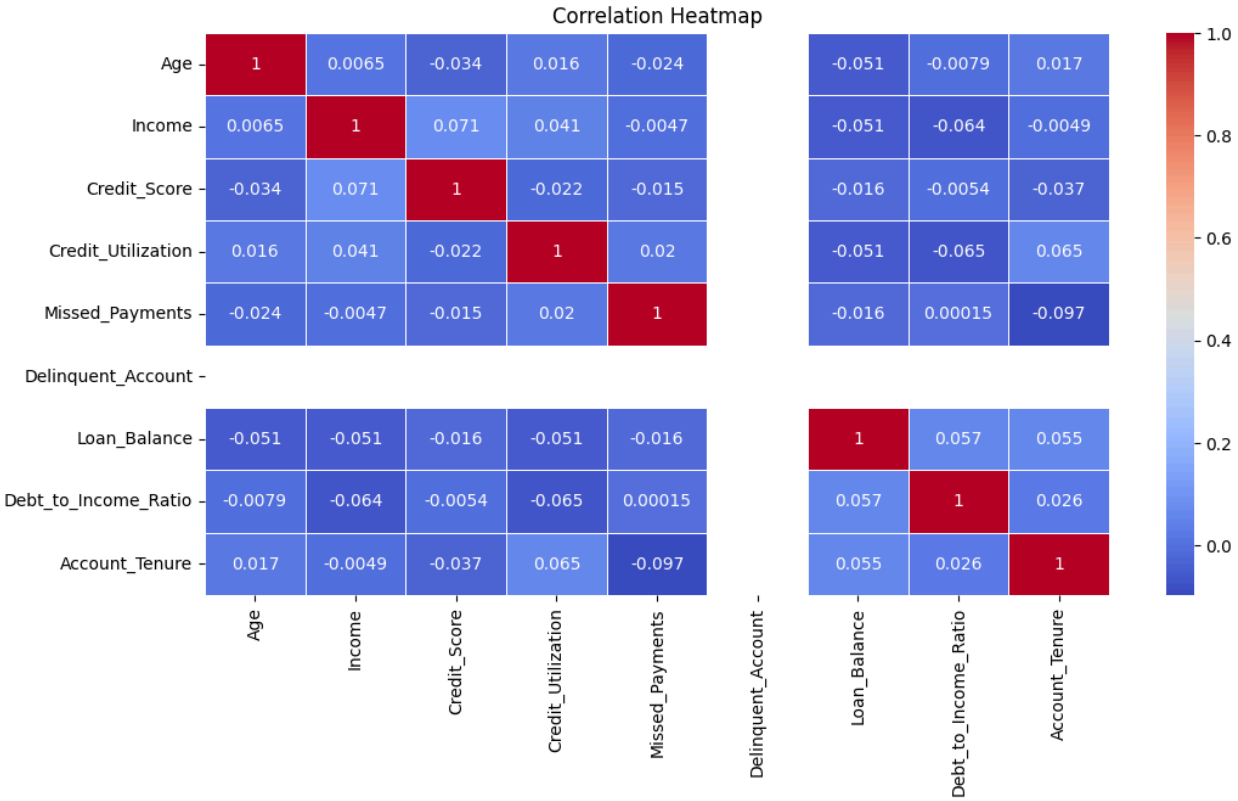

Month_3 vs Delinquency
175
150
125
100
75
50
25
0
count
Delinquent_Account
0.0
Missed
Late
On-time
Month_3


Month_4 vs Delinquency
count
175
150
125
100
75
50
25
0
Delinquent_Account
0.0
Late
Missed
On-time
Month_4


Month_5 vs Delinquency
count
175
150
125
100
75
50
25
0
Delinquent_Account
0.0
Missed
On-time
Late
Month_5



In [72]:
```
corr = data.corr(numeric_only=True)

plt.figure(figsize=(12,6))
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```



In [84]:
```
data['Income'] = data['Income'].fillna(data['Income'].median())
data['Credit_Score'] = data['Credit_Score'].fillna(data['Credit_Score'].median
data['Loan_Balance'] = data['Loan_Balance'].fillna(data['Loan_Balance'].median

data.isnull().sum()
```

Out[84]:

	0
Customer_ID	0
Age	0
Income	0
Credit_Score	0
Credit_Utilization	0
Missed_Payments	0
Delinquent_Account	0
Loan_Balance	0
Debt_to_Income_Ratio	0
Employment_Status	0
Account_Tenure	0
Credit_Card_Type	0
Location	0
Month_1	0
Month_2	0
Month_3	0
Month_4	0
Month_5	0
Month_6	0

dtype: int64

In [85]:
```
data.head()
```

Out[85]:

	Customer_ID	Age	Income	Credit_Score	Credit_Utilization	Missed_Payment
0	CUST0001	56	165580.0	398.0	0.390502	
1	CUST0002	69	100999.0	493.0	0.312444	
2	CUST0003	46	188416.0	500.0	0.359930	
3	CUST0004	32	101672.0	413.0	0.371400	
4	CUST0005	60	38524.0	487.0	0.234716	

In [87]:

```
data['Target'] = (
    (data['Missed_Payments'] >= 3) |
    (data['Credit_Utilization'] > 0.75) |
    (data['Debt_to_Income_Ratio'] > 0.45)
).astype(int)
data['Target']
```

Out[87]:

	Target
0	1
1	1
2	0
3	1
4	1
...	...
495	0
496	0
497	0
498	0
499	1

500 rows × 1 columns

dtype: int64

In [89]:

```
cols_to_drop = [
    'Customer_ID',
    'Delinquent_Account',
    'Month_1','Month_2','Month_3',
    'Month_4','Month_5','Month_6'
]

data = data.drop(columns=cols_to_drop, errors='ignore')
```

```
data.head()
```

Out[89]:

	Age	Income	Credit_Score	Credit_Utilization	Missed_Payments	Loan_Balan
0	56	165580.0	398.0	0.390502	3	1631(
1	69	100999.0	493.0	0.312444	6	1740[
2	46	188416.0	500.0	0.359930	0	1376[
3	32	101672.0	413.0	0.371400	3	8877[
4	60	38524.0	487.0	0.234716	2	1331(

In [92]:
```
X = data.drop(['Target'], axis=1)
y = data['Target']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

In [94]:
```
numeric_features = X_train.select_dtypes(include=['int64','float64']).columns.
categorical_features = X_train.select_dtypes(include=['object']).columns.tolis

print("Numeric Features:", numeric_features)
print("Categorical Features:", categorical_features)
```

```
Numeric Features: ['Age', 'Income', 'Credit_Score', 'Credit_Utilization', 'Miss
ed_Payments', 'Loan_Balance', 'Debt_to_Income_Ratio', 'Account_Tenure']
Categorical Features: ['Employment_Status', 'Credit_Card_Type', 'Location']
```

In [97]:
```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)
```

In [105…
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifie
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "SVM": SVC(probability=True),
    "KNN": KNeighborsClassifier()
}

results = []
```

```
for model_name, clf in models.items():
    pipeline = Pipeline(steps=[
        ('preprocessing', preprocessor),
        ('classifier', clf)
    ])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    y_prob = pipeline.predict_proba(X_test)[:,1]

    # Evaluation
    model_result = {
        "Model": model_name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1 Score": f1_score(y_test, y_pred),
        "ROC-AUC": roc_auc_score(y_test, y_prob)
    }

    results.append(model_result)
results_df = pd.DataFrame(results)
results_df
```

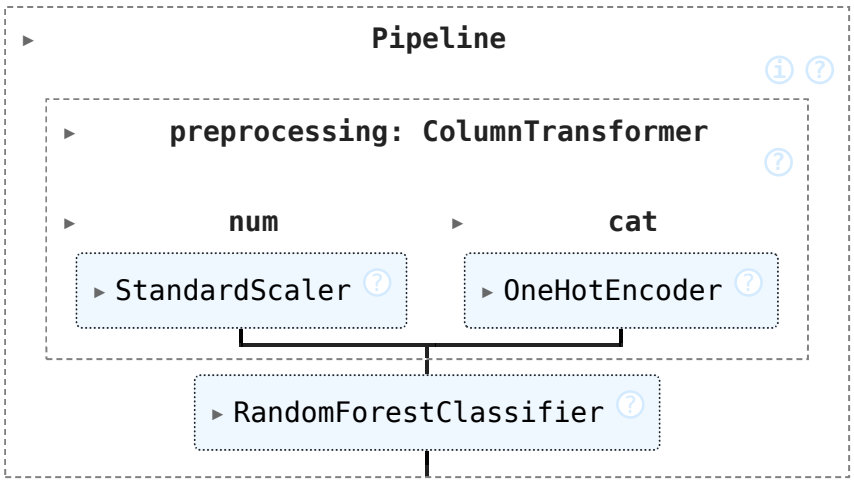
Out[105…

	Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC
0	Logistic Regression	0.89	0.864865	0.984615	0.920863	0.974945
1	Random Forest	0.99	0.984848	1.000000	0.992366	0.999121
2	Gradient Boosting	1.00	1.000000	1.000000	1.000000	1.000000
3	SVM	0.92	0.901408	0.984615	0.941176	0.982418
4	KNN	0.89	0.897059	0.938462	0.917293	0.947912

In [99]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline

rf_model = Pipeline(steps=[
    ('preprocessing', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

rf_model.fit(X_train, y_train)
```

Out[99]:



In [108…

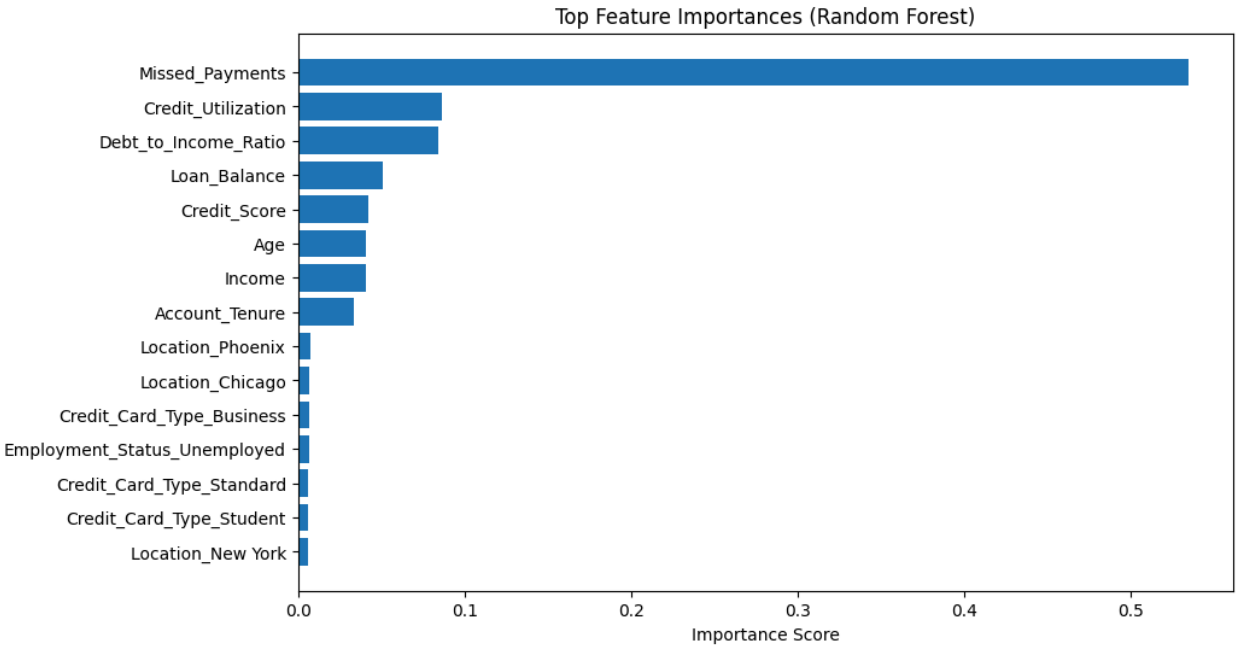
```
y_pred = rf_model.predict(X_test)
y_prob = rf_model.predict_proba(X_test)[:, 1]

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_

print("===== RANDOM FOREST MODEL PERFORMANCE =====")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_prob))

print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
===== RANDOM FOREST MODEL PERFORMANCE =====
Accuracy: 0.99
Precision: 0.9848484848484849
Recall: 1.0
F1 Score: 0.9923664122137404
ROC-AUC: 1.0

Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.97      0.99        35
           1       0.98      1.00      0.99        65

    accuracy                           0.99       100
   macro avg       0.99      0.99      0.99       100
weighted avg       0.99      0.99      0.99       100


Confusion Matrix:
 [[34  1]
 [ 0 65]]
```

In [109…

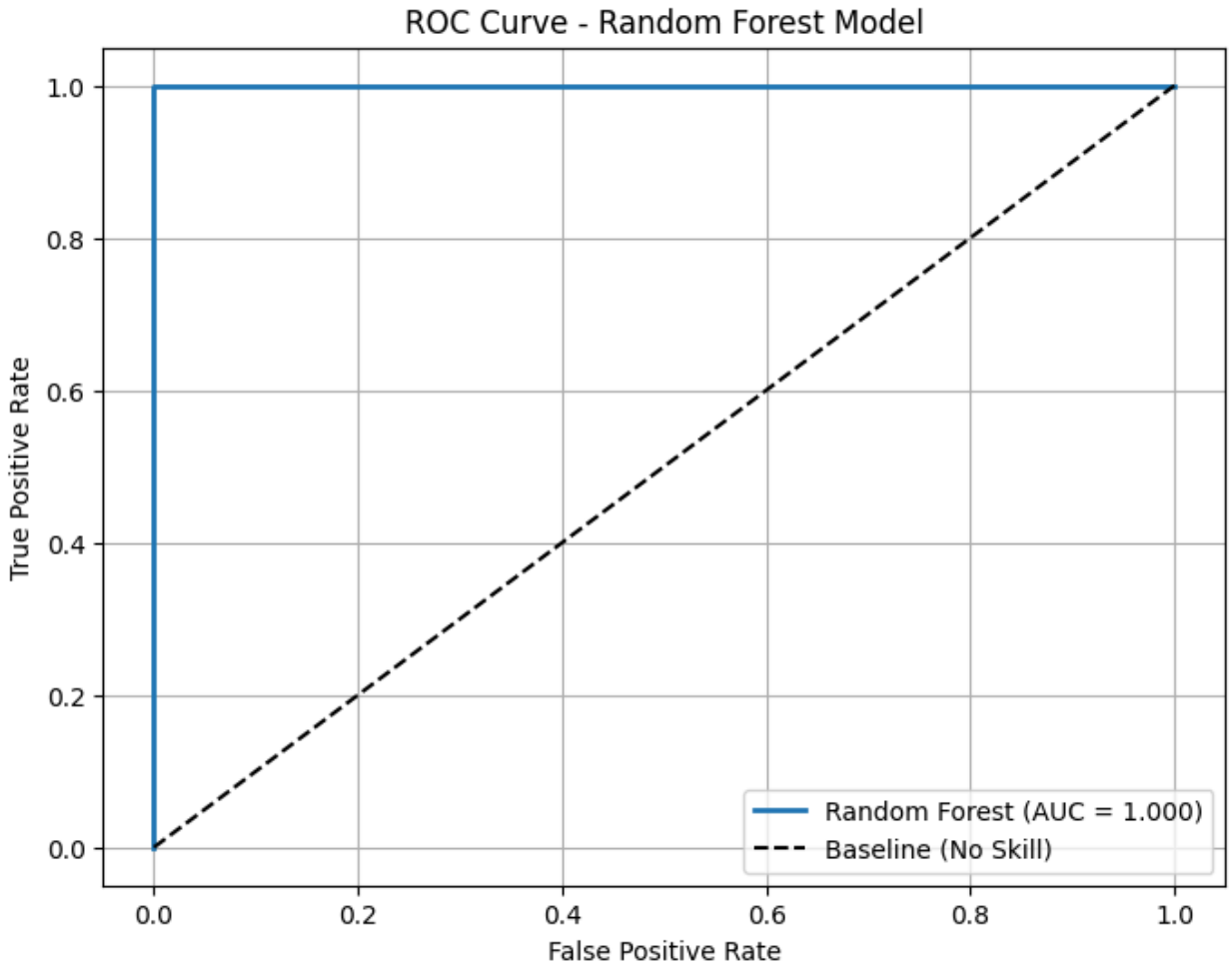
```
import numpy as np
```

```
import matplotlib.pyplot as plt

encoded_features = list(preprocessor.named_transformers_['num'].get_feature_na
                   list(preprocessor.named_transformers_['cat'].get_feature_na

importances = rf_model.named_steps['classifier'].feature_importances_
indices = np.argsort(importances)[::-1]
top_n = 15

plt.figure(figsize=(10,6))
plt.barh(np.array(encoded_features)[indices][:top_n], importances[indices][:to
plt.gca().invert_yaxis()
plt.title("Top Feature Importances (Random Forest)")
plt.xlabel("Importance Score")
plt.show()
```



In [110…

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
# Plot ROC Curve
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f"Random Forest (AUC = {roc_auc:.3f})", linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label="Baseline (No Skill)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Random Forest Model")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



In [112…

```
import joblib

joblib.dump(rf_model, "RandomForest_Delinquency_Model.pkl")

print("Model saved successfully as RandomForest_Delinquency_Model.pkl")
```

```
Model saved successfully as RandomForest_Delinquency_Model.pkl
```