```
import pandas as pd
```

```
df = pd.read_csv(
    "/content/logistic_regression.csv",
    on_bad_lines='skip',  # Skip problematic rows
    quoting=3             # Treat all quote characters as regular characters
)
df.head(5)
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	open_acc	pub_re
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...	16.0	0.
1	Mendozaberg	OK 22690"	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Nal
2	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...	17.0	0.
3	Loganmouth	SD 05113"	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Nal
4	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...	13.0	0.

5 rows × 27 columns

There are about 687568 rows with 27 columns

```
df.columns
```

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'application_type',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8019 entries, 0 to 8018
Data columns (total 27 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   loan_amnt             8019 non-null   object
 1   term                  7562 non-null   object
 2   int_rate              3879 non-null   float64
 3   installment           3879 non-null   float64
 4   grade                 3879 non-null   object
 5   sub_grade             3879 non-null   object
 6   emp_title             3648 non-null   object
 7   emp_length            3693 non-null   object
 8   home_ownership        3879 non-null   object
 9   annual_inc            3879 non-null   float64
 10  verification_status   3879 non-null   object
 11  issue_d               3878 non-null   object
 12  loan_status           3878 non-null   object
 13  purpose               3878 non-null   object
 14  title                 3862 non-null   object
 15  dti                   3878 non-null   float64
 16  earliest_cr_line      3878 non-null   object
 17  open_acc              3878 non-null   float64
 18  pub_rec               3878 non-null   float64
 19  revol_bal             3878 non-null   float64
 20  revol_util            3876 non-null   float64
 21  total_acc             3878 non-null   float64
 22  initial_list_status   3878 non-null   object
 23  application_type      3878 non-null   object
```

```
 24  mort_acc              3554 non-null   float64
 25  pub_rec_bankruptcies  3875 non-null   float64
 26  address               3878 non-null   object
dtypes: float64(11), object(16)
memory usage: 1.7+ MB
```

seems there are lot of null values,in most of the columns, except loan_amt

```
df.isna().sum()
```

	0
loan_amnt	0
term	457
int_rate	4140
installment	4140
grade	4140
sub_grade	4140
emp_title	4371
emp_length	4326
home_ownership	4140
annual_inc	4140
verification_status	4140
issue_d	4141
loan_status	4141
purpose	4141
title	4157
dti	4141
earliest_cr_line	4141
open_acc	4141
pub_rec	4141
revol_bal	4141
revol_util	4143
total_acc	4141
initial_list_status	4141
application_type	4141
mort_acc	4465
pub_rec_bankruptcies	4144
address	4141

dtype: int64

```
for g in df.columns:
    if df[g].dtype in ['float64', 'int64']:
        # Use median if the column is numerical
        df[g] = df[g].fillna(df[g].median())
    else:
        # Use mode if the column is categorical or object
        if not df[g].mode().empty:  # Check if mode exists
            df[g] = df[g].fillna(df[g].mode()[0])
        else:
          print("no Mode for "+ g)
          df[g] = df[g].fillna("unKnown")
print(df.isna().sum())
```

```
loan_amnt               0
term                    0
int_rate                0
installment             0
grade                   0
sub_grade               0
emp_title               0
emp_length              0
home_ownership          0
annual_inc              0
verification_status     0
issue_d                 0
loan_status             0
purpose                 0
title                   0
dti                     0
earliest_cr_line        0
open_acc                0
pub_rec                 0
revol_bal               0
revol_util              0
total_acc               0
initial_list_status     0
application_type        0
mort_acc                0
pub_rec_bankruptcies    0
address                 0
dtype: int64
```

```
print(df.describe())
```

```
          int_rate  installment    annual_inc          dti     open_acc  \
count  8019.000000  8019.000000  8.019000e+03  8019.000000  8019.000000
mean     13.480335   403.794382  6.862200e+04    17.006115    11.113605
std       3.078912   174.073247  4.169159e+04     5.665633     3.460202
min       5.320000    24.320000  4.200000e+03     0.000000     1.000000
25%      13.330000   378.580000  6.400000e+04    16.740000    11.000000
50%      13.330000   378.580000  6.400000e+04    16.740000    11.000000
75%      13.330000   378.580000  6.400000e+04    16.740000    11.000000
max      28.990000  1309.490000  2.500000e+06    43.690000    42.000000

           pub_rec      revol_bal   revol_util    total_acc     mort_acc  \
count  8019.000000    8019.000000  8019.000000  8019.000000  8019.000000
mean      0.082679   12975.690984    54.052862    24.529492     1.319741
std       0.349283   13059.144464    16.872191     8.093213     1.424853
min       0.000000       0.000000     0.000000     2.000000     0.000000
25%       0.000000   10726.000000    54.600000    24.000000     1.000000
50%       0.000000   10726.000000    54.600000    24.000000     1.000000
75%       0.000000   10726.000000    54.600000    24.000000     1.000000
max       8.000000  382666.000000   106.500000    84.000000    13.000000

       pub_rec_bankruptcies
count           8019.000000
mean               0.057863
std                0.251499
min                0.000000
25%                0.000000
50%                0.000000
75%                0.000000
max                4.000000
```

1.Most borrowers have no public records or bankruptcies, indicating strong credit profiles

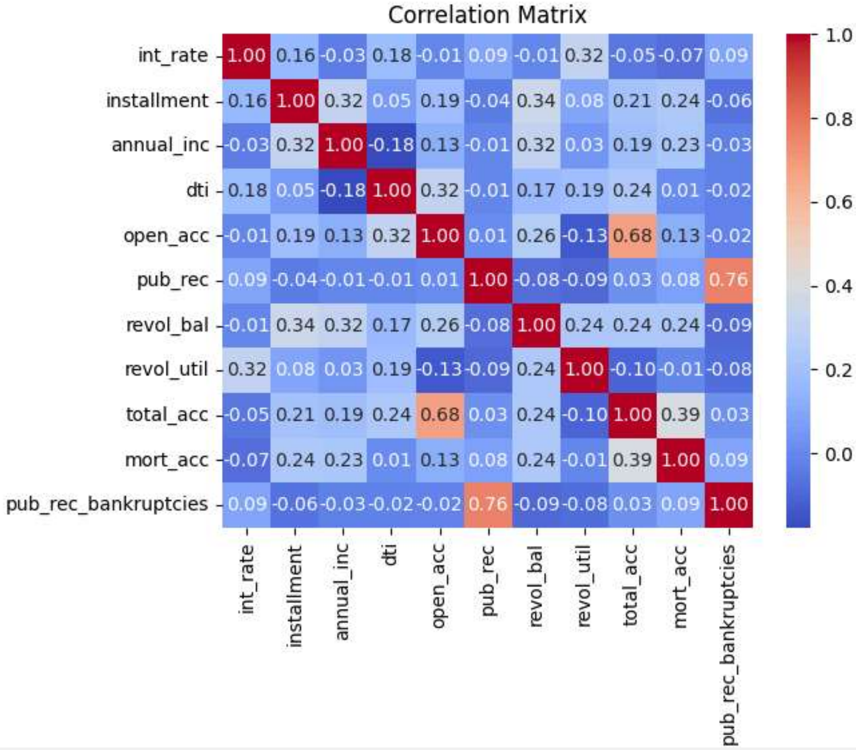
2.Both installment and int_rate distributions are relatively symmetric, with median interest rates at 13.33% and median installments at $375

While the median revol_bal $(10, 947) and revol_util(55.2$1.7M

Start coding or generate with AI.

```
# Select only numeric columns (int and float)
import seaborn as sns
import matplotlib.pyplot as plt
numeric_df = df.select_dtypes(include=['float64', 'int64'])

# Drop rows with NaN values (if any)
numeric_df = numeric_df.dropna()
```

```
# Compute correlation matrix for numeric columns
corr_matrix = numeric_df.corr()

# Plot the heatmap
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



1.The int_rate (interest rate) has weak correlations with most other columns, with the highest correlation being 0.17 with installment 2.There is a noticeable positive correlation of 0.62 between revol_util (revolving utilization) and pub_rec_bankruptcies (public record bankruptcies). This suggests that as revolving utilization increases, the likelihood of bankruptcies in the public record also increases, which makes sense from a financial behavior perspective.

Start coding or generate with AI.

```
df['loan_status'].value_counts()
```

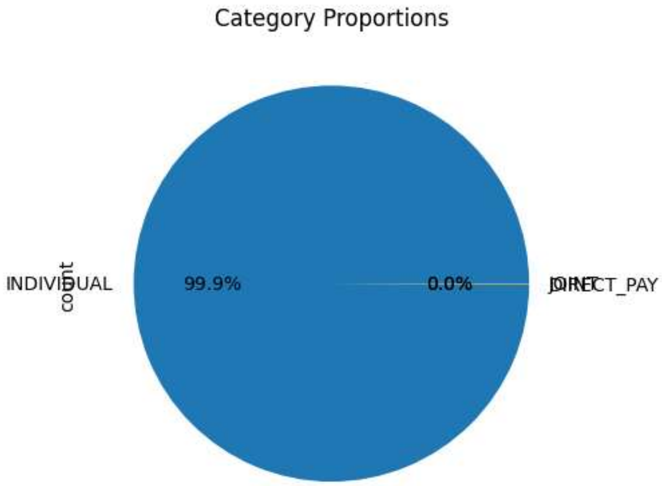
	count
loan_status	
Fully Paid	7282
Charged Off	737

dtype: int64

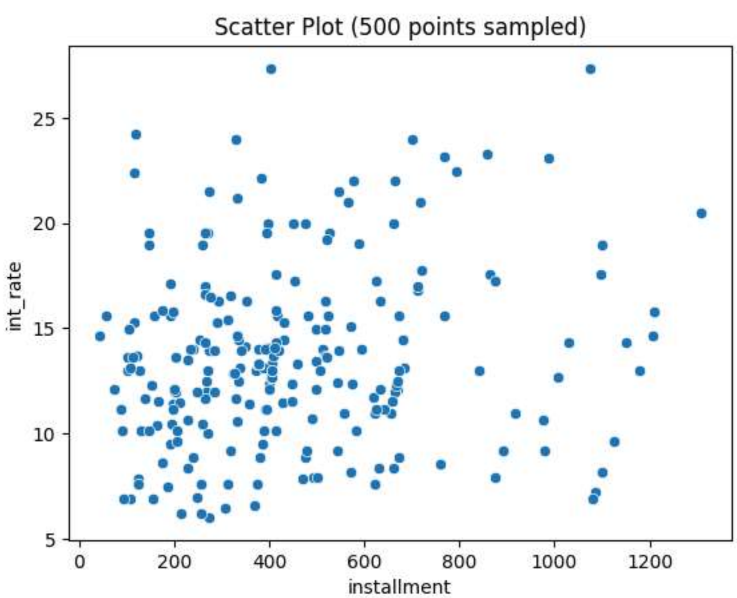
seems most of them are fully paid

```
Percent_of_cust_fully_paid = (df['loan_status'].value_counts()['Fully Paid']/df.shape[0])*100
```

```
# Pie chart for categorical variable
df['application_type'].value_counts().plot.pie(autopct='%1.1f%%')
plt.title('Category Proportions')
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample 500 random points from the dataframe
sampled_df = df.sample(n=500, random_state=42)

# Scatter plot
sns.scatterplot(x=sampled_df['installment'], y=sampled_df['int_rate'])
plt.title('Scatter Plot (500 points sampled)')
plt.show()
```




```
# Ordinal encoding for grade
grade_mapping = {'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7}
df['grade'] = df['grade'].map(grade_mapping)

# Ordinal encoding for sub_grade
sub_grade_mapping = {'A1': 1, 'A2': 2, 'A3': 3, 'A4': 4, 'A5': 5, 'B1': 6, 'B2': 7, 'B3': 8, 'B4': 9, 'B5': 10,
                     'C1': 11, 'C2': 12, 'C3': 13, 'C4': 14, 'C5': 15, 'D1': 16, 'D2': 17, 'D3': 18, 'D4': 19, 'D5': 20,
                     'E1': 21, 'E2': 22, 'E3': 23, 'E4': 24, 'E5': 25, 'F1': 26, 'F2': 27, 'F3': 28, 'F4': 29, 'F5': 30,
                     'G1': 31, 'G2': 32, 'G3': 33, 'G4': 34, 'G5': 35}
df['sub_grade'] = df['sub_grade'].map(sub_grade_mapping)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8019 entries, 0 to 8018
Data columns (total 27 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   loan_amnt             8019 non-null   object 
 1   term                  8019 non-null   object 
 2   int_rate              8019 non-null   float64
 3   installment           8019 non-null   float64
 4   grade                 8019 non-null   int64  
 5   sub_grade             8019 non-null   int64  
 6   emp_title             8019 non-null   object 
 7   emp_length            8019 non-null   object 
 8   home_ownership        8019 non-null   object 
 9   annual_inc            8019 non-null   float64
 10  verification_status   8019 non-null   object 
 11  issue_d               8019 non-null   object 
 12  loan_status           8019 non-null   object 
 13  purpose               8019 non-null   object 
 14  title                 8019 non-null   object 
 15  dti                   8019 non-null   float64
 16  earliest_cr_line      8019 non-null   object 
 17  open_acc              8019 non-null   float64
 18  pub_rec               8019 non-null   float64
 19  revol_bal             8019 non-null   float64
 20  revol_util            8019 non-null   float64
 21  total_acc             8019 non-null   float64
 22  initial_list_status   8019 non-null   object 
 23  application_type      8019 non-null   object 
 24  mort_acc              8019 non-null   float64
 25  pub_rec_bankruptcies  8019 non-null   float64
 26  address               8019 non-null   object 
dtypes: float64(11), int64(2), object(14)
memory usage: 1.7+ MB
```

```
for col in df.select_dtypes(include=['object']).columns:
    # Create a mapping for each column
    unique_values = df[col].unique()
    value_mapping = {value: idx + 1 for idx, value in enumerate(unique_values)}

    # Map the column values
    df[col] = df[col].map(value_mapping)

print(df)
```

```
      loan_amnt  term  int_rate  installment  grade  sub_grade  emp_title  \
0             1     1     11.44       329.48      2          9          1   
1             2     2     13.33       378.58      2          9          2   
2             3     1     11.99       265.68      2         10          3   
3             4     3     13.33       378.58      2          9          2   
4             5     1     10.49       506.97      2          8          4   
...         ...   ...       ...          ...    ...        ...        ...   
8014       2651     1      8.18       213.66      2          6       2892   
8015       3948    96     13.33       378.58      2          9          2   
8016          1     1     14.09       342.22      2         10       2893   
8017       3949   405     13.33       378.58      2          9          2   
8018          1     1      8.90       317.54      1          5       2894   

      emp_length  home_ownership  annual_inc  ...  open_acc  pub_rec  \
0              1               1    117000.0  ...      16.0      0.0   
1              1               2     64000.0  ...      11.0      0.0   
2              2               2     65000.0  ...      17.0      0.0   
3              1               2     64000.0  ...      11.0      0.0   
4              3               1     43057.0  ...      13.0      0.0   
...          ...             ...         ...  ...       ...      ...   
8014           6               1     56160.0  ...      15.0      1.0   
8015           1               2     64000.0  ...      11.0      0.0   
8016          10               1     35000.0  ...      19.0      0.0   
8017           1               2     64000.0  ...      11.0      0.0   
8018          11               2     45000.0  ...      11.0      0.0   

      revol_bal  revol_util  total_acc  initial_list_status  application_type  \
0       36369.0        41.8       25.0                    1                 1   
1       10726.0        54.6       24.0                    2                 1   
2       20131.0        53.3       27.0                    2                 1   
3       10726.0        54.6       24.0                    2                 1   
4       11987.0        92.2       26.0                    2                 1   
...         ...         ...        ...                  ...               ...   
8014     9034.0        46.1       44.0                    1                 1   
```

```
8015    10726.0        54.6       24.0                    2                 1
8016    11003.0        55.9       32.0                    1                 1
8017    10726.0        54.6       24.0                    2                 1
8018    10726.0        54.6       24.0                    2                 1

      mort_acc  pub_rec_bankruptcies  address
0          0.0                   0.0        1
1          1.0                   0.0        2
2          3.0                   0.0        3
3          1.0                   0.0        2
4          0.0                   0.0        4
...        ...                   ...      ...
8014       0.0                   1.0     3868
8015       1.0                   0.0        2
8016       0.0                   0.0     3869
8017       1.0                   0.0        2
8018       1.0                   0.0        2

[8019 rows x 27 columns]
```

```
import pandas as pd
import numpy as np

# Ensure columns are numeric
df['loan_amnt'] = pd.to_numeric(df['loan_amnt'], errors='coerce')
df['installment'] = pd.to_numeric(df['installment'], errors='coerce')

# Drop rows with missing values (if any were converted to NaN)
df = df.dropna(subset=['loan_amnt', 'installment'])

# Calculate correlation coefficient
correlation = np.corrcoef(df['loan_amnt'], df['installment'])[0, 1]
print(f"Correlation between Loan Amount and Installment: {correlation:.2f}")
```

```
Correlation between Loan Amount and Installment: 0.95
```

```
import pandas as pd
from sklearn.preprocessing import StandardScaler



# Separate features and target variable
X = df.drop('loan_status', axis=1)  # Features
y = df['loan_status']  # Target variable

# StandardScaler for feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert scaled features back to DataFrame for easier analysis
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
```

Preparing th data for test and train

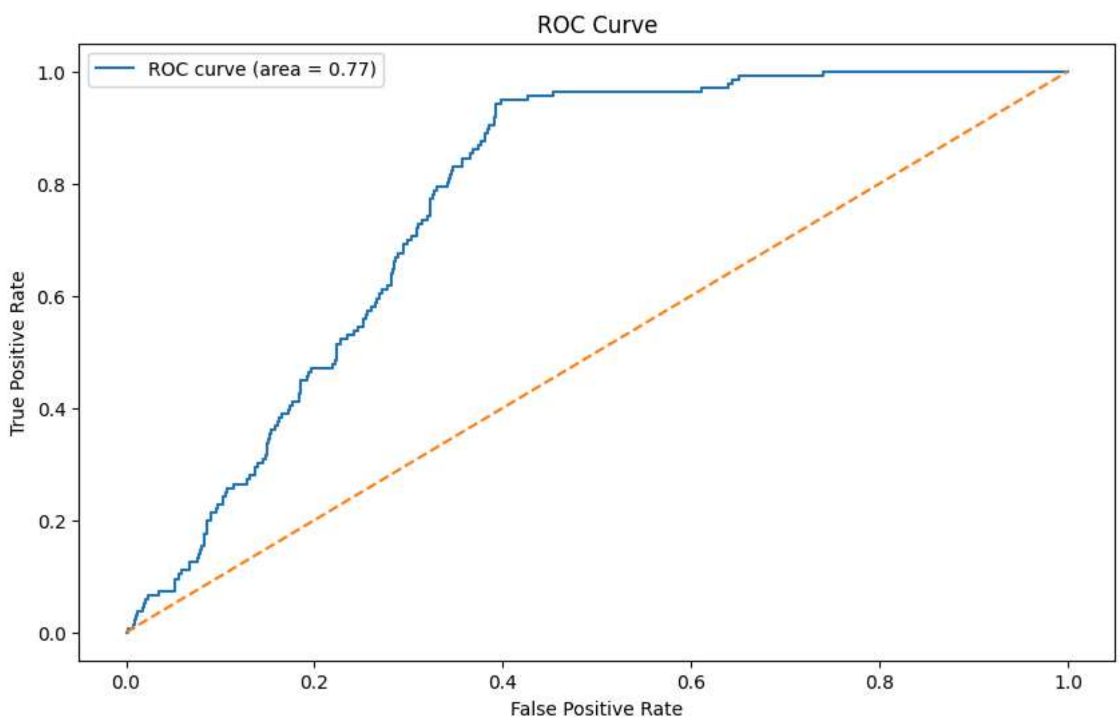
```
from sklearn.model_selection import train_test_split

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Logistic Regression using sklearn

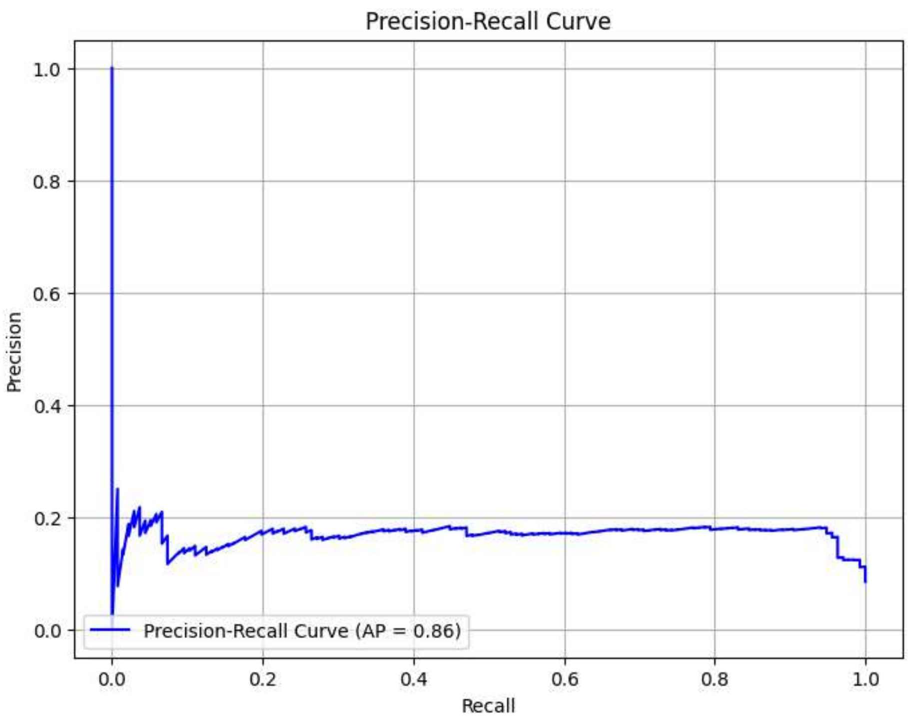
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, precision_score, f1_score, roc_auc_score, roc_curve
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

```
# Logistic Regression
lr_model = LogisticRegression(max_iter=2000, solver='saga', random_state=42, class_weight='balanced')
lr_model.fit(X_train, y_train)

# Predictions
y_pred = lr_model.predict(X_test)
y_pred_proba = lr_model.predict_proba(X_test)[:, 1]
y_test_adjusted = y_test.map({1: 0, 2: 1})

# Evaluation
conf_matrix = confusion_matrix(y_test, y_pred)
print("Precision Score:", precision_score(y_test_adjusted, y_pred_statsmodels))
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_report(y_test, y_pred, zero_division=0))
print("F1 Score:", f1_score(y_test, y_pred, average='binary', zero_division=0))

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test_adjusted, y_pred_proba)
roc_auc = roc_auc_score(y_test_adjusted, y_pred_proba)
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:349: ConvergenceWarning: The max_iter was reached which means the c
  warnings.warn(
Precision Score: 0.5434782608695652
Accuracy: 0.6970074812967582
Confusion Matrix:
 [[1022  446]
 [  40   96]]
Classification Report:
               precision    recall  f1-score   support

           1       0.96      0.70      0.81      1468
           2       0.18      0.71      0.28       136

    accuracy                           0.70      1604
   macro avg       0.57      0.70      0.55      1604
weighted avg       0.90      0.70      0.76      1604

F1 Score: 0.807905138339921
```



```
precision, recall, thresholds = precision_recall_curve(y_test_adjusted, y_pred_proba)

# Compute average precision score
average_precision = average_precision_score(y_test, y_pred_proba)

# Plot the Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label=f'Precision-Recall Curve (AP = {average_precision:.2f})', color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.grid()
plt.show()
```




```
# Calculate the metrics
```

Logistic Regression Using StatsModel

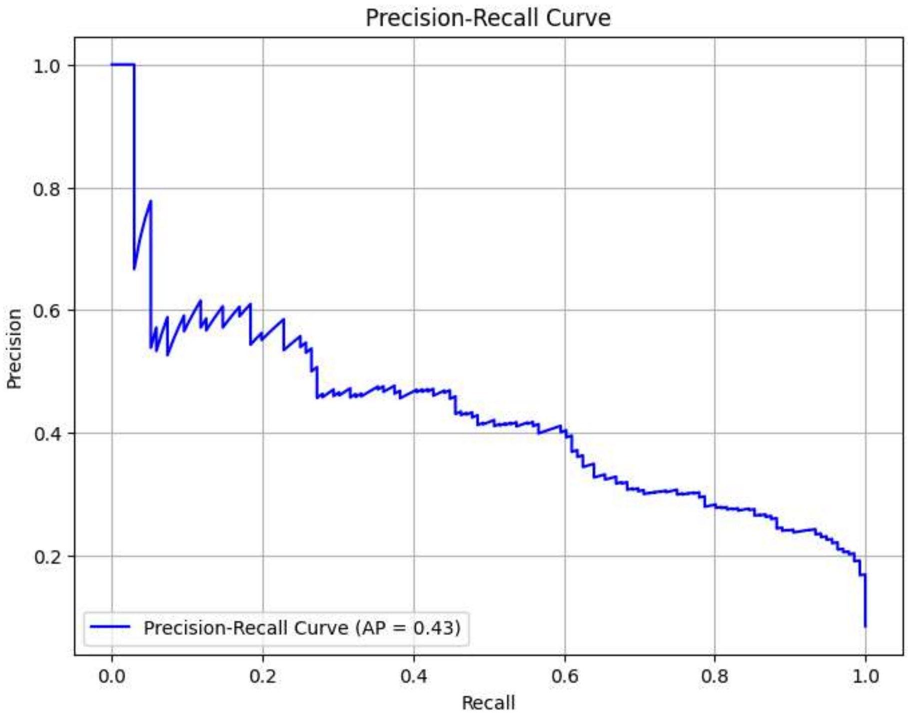
```
import statsmodels.api as sm
import numpy as np
from sklearn.metrics import (
    confusion_matrix, accuracy_score, classification_report, precision_score, f1_score,
    roc_auc_score, roc_curve, precision_recall_curve, average_precision_score
)
import matplotlib.pyplot as plt

# Ensure y_test is binary: Adjust labels to {0, 1} if needed
y_test_adjusted = y_test.map({1: 0, 2: 1})

# Add constant to X for intercept in statsmodels
X_test_const = sm.add_constant(X_test)

# Fit Logistic Regression model using statsmodels
logit_model = sm.Logit(y_test_adjusted, X_test_const).fit()

# Display summary statistics
print(logit_model.summary())

# Predict probabilities
y_pred_proba_statsmodels = logit_model.predict(X_test_const)
```

```
# Convert probabilities to binary predictions (threshold = 0.5)
y_pred_statsmodels = (y_pred_proba_statsmodels >= 0.5).astype(int)

# Evaluate Model
conf_matrix = confusion_matrix(y_test_adjusted, y_pred_statsmodels)
print("Confusion Matrix:\n", conf_matrix)
print("Accuracy:", accuracy_score(y_test_adjusted, y_pred_statsmodels))
print("Precision Score:", precision_score(y_test_adjusted, y_pred_statsmodels))
print("F1 Score:", f1_score(y_test_adjusted, y_pred_statsmodels))
roc_auc = roc_auc_score(y_test_adjusted, y_pred_proba_statsmodels)
print("ROC AUC Score:", roc_auc)

# Plot ROC Curve
fpr, tpr, thresholds = roc_curve(y_test_adjusted, y_pred_proba_statsmodels)
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.193740
         Iterations: 35
                           Logit Regression Results
==============================================================================
Dep. Variable:            loan_status   No. Observations:                 1604
Model:                          Logit   Df Residuals:                     1577
Method:                           MLE   Df Model:                           26
Date:                Mon, 02 Dec 2024   Pseudo R-squ.:                  0.3326
Time:                        05:18:55   Log-Likelihood:                -310.76
converged:                      False   LL-Null:                       -465.66
Covariance Type:            nonrobust   LLR p-value:                 2.310e-50
=======================================================================================
                          coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                  23.6852   2.37e+06      1e-05      1.000   -4.64e+06    4.64e+06
loan_amnt           -3.232e-05      0.000     -0.252      0.801      -0.000       0.000
term                   -0.0345      0.019     -1.856      0.063      -0.071       0.002
int_rate               -0.3113      0.115     -2.697      0.007      -0.537      -0.085
installment             0.0009      0.001      1.729      0.084      -0.000       0.002
grade                   0.3919      0.381      1.029      0.303      -0.355       1.138
sub_grade               0.2393      0.107      2.239      0.025       0.030       0.449
emp_title               0.0004      0.000      2.220      0.026    4.85e-05       0.001
emp_length              0.0199      0.031      0.635      0.525      -0.041       0.081
home_ownership         -0.2161      0.181     -1.195      0.232      -0.571       0.138
annual_inc          -1.837e-06   4.06e-06     -0.452      0.651    -9.8e-06    6.12e-06
verification_status     0.3349      0.142      2.361      0.018       0.057       0.613
issue_d                -0.0037      0.006     -0.677      0.498      -0.015       0.007
purpose                 0.0559      0.053      1.061      0.289      -0.047       0.159
title                   0.0004      0.001      0.761      0.447      -0.001       0.002
dti                     0.0355      0.016      2.251      0.024       0.005       0.066
earliest_cr_line        0.0029      0.001      2.722      0.006       0.001       0.005
open_acc               -0.0233      0.034     -0.679      0.497      -0.091       0.044
pub_rec                 0.4659      0.329      1.414      0.157      -0.180       1.111
revol_bal            3.912e-06   7.86e-06      0.498      0.619   -1.15e-05    1.93e-05
revol_util              0.0058      0.005      1.100      0.271      -0.004       0.016
total_acc              -0.0234      0.015     -1.591      0.112      -0.052       0.005
initial_list_status    -0.0367      0.229     -0.160      0.873      -0.485       0.412
application_type      -26.5055   2.37e+06  -1.12e-05      1.000   -4.64e+06    4.64e+06
mort_acc               -0.0211      0.071     -0.296      0.767      -0.161       0.119
pub_rec_bankruptcies   -0.4347      0.430     -1.011      0.312      -1.277       0.408
address                -0.0003      0.000     -1.824      0.068      -0.001    2.07e-05
=======================================================================================

Possibly complete quasi-separation: A fraction 0.26 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
Confusion Matrix:
 [[1447   21]
 [ 111   25]]
Accuracy: 0.9177057356608479
Precision Score: 0.5434782608695652
F1 Score: 0.27472527472527475
ROC AUC Score: 0.89866164449431
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to c
  warnings.warn("Maximum Likelihood optimization failed to "
```

Receiver Operating Characteristic (ROC)

```python
# Precision-Recall Curve
precision, recall, thresholds = precision_recall_curve(y_test_adjusted, y_pred_proba_statsmodels)
average_precision = average_precision_score(y_test_adjusted, y_pred_proba_statsmodels)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label=f'Precision-Recall Curve (AP = {average_precision:.2f})', color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.grid()
plt.show()
```



Insights

Model Performance:

Confusion Matrix: Indicates that the model correctly predicted 1447 instances as negatives and 25 as positives, with 132 misclassifications (111 false negatives and 21 false positives). **Accuracy:** The model achieved 91.77%, indicating high overall correctness. However, accuracy alone may not be sufficient to evaluate the model given class imbalance. **Precision (0.543)**: This reflects that 54.3% of the predicted positive cases are correct. The relatively low precision suggests that the model struggles to reliably identify true positives. **F1 Score (0.275)**: The low F1 score indicates poor balance between precision and recall, which implies the model is underperforming for the minority class. **ROC AUC Score (0.899)**: A high ROC AUC score suggests the model can distinguish between positive and negative classes fairly well.

Significant Features: int_rate (interest rate): Negative coefficient indicates higher interest rates are associated with a lower likelihood of a positive loan_status. **sub_grade:** Positively associated with the target variable, suggesting sub-grades influence loan approval likelihood. emp_title and verification_status: Positive coefficients suggest they increase the likelihood of a positive loan status. **dti (debt-to-income ratio):** A small but positive coefficient indicates higher DTI is slightly associated with increased loan default risk. **earliest_cr_line:** Older credit history (higher values) seems positively associated with better loan outcomes. **Non-significant Features:** Many features, such as annual_inc, home_ownership, and pub_rec_bankruptcies, have p-values > 0.05, suggesting their contribution to predicting loan_status is negligible in this model.

Trade-Off Questionaire

1.How can we make sure that our model can detect real defaulters and there are less false positives? This is important as we can lose out on an opportunity to finance more individuals and earn interest on it.

- Balancing Sensitivity and Specificity
- Penalizing Misscalisfication .
- Fewer false positives, leading to fewer lost opportunities for financing good customers.
- Retention of real defaulters, minimizing financial risks. Improved alignment of the model with business objectives.

**2. **Since NPA (non-performing asset) is a real problem in this industry, it's important we play safe and shouldn't disburse loans to anyone.

avoiding Non-Performing Assets (NPAs) is a critical priority, you need a conservative approach to loan disbursement. This would mean prioritizing reducing false negatives (defaulters not detected) even at the risk of rejecting some potential customers

Questionarie

1. What percentage of customers have fully paid their Loan Amount?

```
Percent_of_cust_fully_paid
```

```
90.80932784636488
```

2. Comment about the correlation between Loan Amount and Installment features.

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

```
import pandas as pd
import numpy as np

# Ensure columns are numeric
df['loan_amnt'] = pd.to_numeric(df['loan_amnt'], errors='coerce')
df['installment'] = pd.to_numeric(df['installment'], errors='coerce')

# Drop rows with missing values (if any were converted to NaN)
df = df.dropna(subset=['loan_amnt', 'installment'])

# Calculate correlation coefficient
correlation = np.corrcoef(df['loan_amnt'], df['installment'])[0, 1]
print(f"Correlation between Loan Amount and Installment: {correlation:.2f}")
```

```
Correlation between Loan Amount and Installment: 0.95
```

They are highly corelated(o.95 ~ 1)

3. The majority of people have home ownership as

```
# Check the distribution of home ownership
home_ownership_distribution = df['home_ownership'].value_counts()

# Display the most common home ownership type
most_common_home_ownership = home_ownership_distribution.idxmax()
most_common_count = home_ownership_distribution.max()

print(f"The majority of people have home ownership as: {most_common_home_ownership} ({most_common_count} people)")
```

```
The majority of people have home ownership as: MORTGAGE (1884 people)
```

4. People with grades ‘A’ are more likely to fully pay their loan. (T/F)

```
# Group data by 'grade' and calculate the percentage of loans fully paid
grade_analysis = df.groupby('grade')['loan_status'].value_counts(normalize=True).unstack()

# Filter the percentage of fully paid loans for each grade
fully_paid_percentage = grade_analysis['Fully Paid'] * 100

# Check the percentage for grade 'A'
grade_a_fully_paid = fully_paid_percentage.get('A', 0)

# Determine if people with grade 'A' are more likely to fully pay their loans
most_likely_grade = fully_paid_percentage.idxmax()
highest_percentage = fully_paid_percentage.max()

print(f"Percentage of fully paid loans for grade A: {grade_a_fully_paid:.2f}%")
print(f"Grade most likely to fully pay their loan: {most_likely_grade} ({highest_percentage:.2f}%)")
```

```
Percentage of fully paid loans for grade A: 95.09%
Grade most likely to fully pay their loan: A (95.09%)
```

SO, its true that, People with grades ‘A’ are more likely to fully pay their loan

5. Name the top 2 afforded job titles.

```
# Calculate the frequency of each job title
top_job_titles = df['emp_title'].value_counts()

# Extract the top 2 job titles
top_2_job_titles = top_job_titles.head(2)

print("Top 2 afforded job titles:")
print(top_2_job_titles)
```

```
Top 2 afforded job titles:
emp_title
Teacher    279
Manager     42
Name: count, dtype: int64
```

6. Thinking from a bank's perspective, which metric should our primary focus be on.. ROC AUC Precision Recall F1 Score **bold text**

Primary Metric: Recall – Focus on identifying as many actual defaulters as possible to reduce the risk of NPAs. Secondary Metric: F1 Score – Ensure a balance if both risks (false positives and false negatives) are critical. Tertiary Metric: Precision and ROC AUC can be used for model