Import Necessary Libraries

```
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score


#Disable warnings
warnings.filterwarnings("ignore")
```

Load Datasets

```
#load training and testing datasets into pandad dataframe
train_data = pd.read_csv("/content/drive/MyDrive/Rocket/train.csv")
test_data = pd.read_csv("/content/drive/MyDrive/Rocket/test.csv")
```

Explore Dataset

```
#display first few rows of dataset
train_data.head()
```

	Unnamed: 0	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	...	open_acc	pub_rec
0	0	6400.0	36 months	15.61	223.78	C	C3	Accountant	< 1 year	RENT	...	12.0	0.0
1	1	25000.0	60 months	19.99	662.21	E	E1	Electronic Technician	10+ years	MORTGAGE	...	18.0	1.0
2	2	15000.0	36 months	5.32	451.73	A	A1	Transportation Coordinator	10+ years	MORTGAGE	...	12.0	0.0
3	3	16000.0	36 months	15.61	559.44	C	C3	ironworker	< 1 year	RENT	...	8.0	0.0
4	4	8725.0	36 months	12.12	290.30	B	B3	Hathaway-Sycamores child & Family Serv	10+ years	MORTGAGE	...	10.0	0.0

5 rows × 28 columns

```
#check number of rows and columns
train_data.shape
```

```
(316970, 28)
```

```
#check data types of columns
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 316970 entries, 0 to 316969
Data columns (total 28 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Unnamed: 0            316970 non-null  int64
```

```
 1   loan_amnt             316970 non-null  float64
 2   term                  316970 non-null  object
 3   int_rate              316970 non-null  float64
 4   installment           316970 non-null  float64
 5   grade                 316970 non-null  object
 6   sub_grade             316970 non-null  object
 7   emp_title             298572 non-null  object
 8   emp_length            302294 non-null  object
 9   home_ownership        316970 non-null  object
 10  annual_inc            316970 non-null  float64
 11  verification_status   316970 non-null  object
 12  issue_d               316970 non-null  object
 13  loan_status           316970 non-null  object
 14  purpose               316970 non-null  object
 15  title                 315573 non-null  object
 16  dti                   316970 non-null  float64
 17  earliest_cr_line      316970 non-null  object
 18  open_acc              316970 non-null  float64
 19  pub_rec               316970 non-null  float64
 20  revol_bal             316970 non-null  float64
 21  revol_util            316757 non-null  float64
 22  total_acc             316970 non-null  float64
 23  initial_list_status   316970 non-null  object
 24  application_type      316970 non-null  object
 25  mort_acc              286753 non-null  float64
 26  pub_rec_bankruptcies  316540 non-null  float64
 27  address               316970 non-null  object
dtypes: float64(12), int64(1), object(15)
memory usage: 67.7+ MB
```

EXPLORATORY DATA ANALYSIS(EDA)

A. Data Preprocessing

1. Remove Unnecessary Columns

```
# train_data = train_data.drop(columns=['Unnamed: 0'])
columns_to_drop = ['Unnamed: 0', 'emp_title', 'title', 'address']
train_data = train_data.drop(columns=columns_to_drop, errors='ignore')
test_data = test_data.drop(columns=columns_to_drop, errors='ignore')
```

```
train_data.head()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownership	annual_inc	verification_status	...	earliest
0	6400.0	36 months	15.61	223.78	C	C3	< 1 year	RENT	60000.0	Verified	...	
1	25000.0	60 months	19.99	662.21	E	E1	10+ years	MORTGAGE	67000.0	Source Verified	...	
2	15000.0	36 months	5.32	451.73	A	A1	10+ years	MORTGAGE	59000.0	Source Verified	...	
3	16000.0	36 months	15.61	559.44	C	C3	< 1 year	RENT	72000.0	Verified	...	
4	8725.0	36 months	12.12	290.30	B	B3	10+ years	MORTGAGE	50000.0	Source Verified	...	

5 rows × 24 columns

2. Handling Missing Values

```
## Check for missing values
missing_values = train_data.isnull().sum()
print("Missing values:\n", missing_values)
```

```
Missing values:
 loan_amnt                 0
term                      0
int_rate                  0
installment               0
```

```
grade                      0
sub_grade                  0
emp_length             14676
home_ownership             0
annual_inc                 0
verification_status        0
issue_d                    0
loan_status                0
purpose                    0
dti                        0
earliest_cr_line           0
open_acc                   0
pub_rec                    0
revol_bal                  0
revol_util               213
total_acc                  0
initial_list_status        0
application_type           0
mort_acc               30217
pub_rec_bankruptcies     430
dtype: int64
```

Interpretation: There are missing values for the columns `emp_length`, `revol_util` , `mort_acc` and `pub_rec_bankruptcies`.

```
# Handling Missing Values - Imputation

#Fill missing values of emp_length with the mode
train_data['emp_length'].fillna(train_data['emp_length'].mode()[0], inplace=True)

#Fill missing values of revol_util with the median
train_data['revol_util'].fillna(train_data['revol_util'].median(), inplace=True)

#Fill missing valuesof mort_acc with the median
train_data['mort_acc'].fillna(train_data['mort_acc'].median(), inplace=True)

#Fill missing valuesof pub_rec_bankruptcies with the median
train_data['pub_rec_bankruptcies'].fillna(train_data['pub_rec_bankruptcies'].median(), inplace=True)

# Handle missing values in test_data
test_data['emp_length'].fillna(train_data['emp_length'].mode()[0], inplace=True)
test_data['revol_util'].fillna(train_data['revol_util'].median(), inplace=True)
test_data['mort_acc'].fillna(train_data['mort_acc'].median(), inplace=True)
test_data['pub_rec_bankruptcies'].fillna(train_data['pub_rec_bankruptcies'].median(), inplace=True)

# Check for missing values
missing_values = train_data.isnull().sum()
print("Missing values:\n", missing_values)
```

```
Missing values:
 loan_amnt              0
term                   0
int_rate               0
installment            0
grade                  0
sub_grade              0
emp_length             0
home_ownership         0
annual_inc             0
verification_status    0
issue_d                0
loan_status            0
purpose                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
dtype: int64
```

3. Check for Duplicates

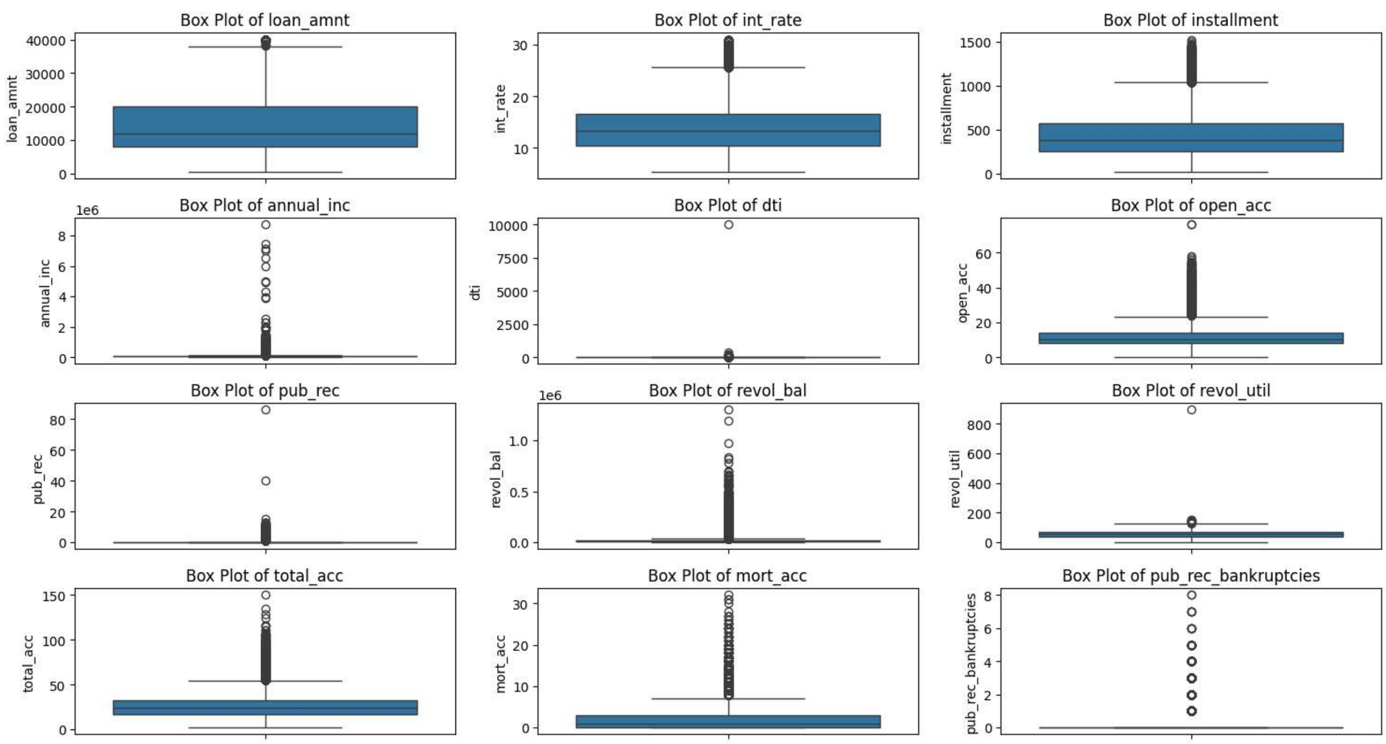
```
#3. check duplicates - duplicate rows should be removed.
#True -> there are duplicate rows
#False -> There are no duplicate rows
train_data.duplicated().any()
```

```
np.False_
```

Interpretation: There are no duplicate records.

4. Handling Outliers

```
#check for outliers
# Select all numerical columns
num_cols = train_data.select_dtypes(include=['number']).columns.tolist()

# Visualization - Plot box plots for numerical columns
plt.figure(figsize=(15, 10))
for i, col in enumerate(num_cols, 1):
    plt.subplot((len(num_cols) // 3) + 1, 3, i)
    sns.boxplot(y=train_data[col])
    plt.title(f'Box Plot of {col}')
plt.tight_layout()
plt.show()
```




```
#check for outliers with z-score
from scipy.stats import zscore

# Calculate Z-scores for numerical columns
z_scores = train_data[num_cols].apply(zscore)
```

```
# Identify outliers per feature (absolute Z-score > 3)
outliers_per_feature = (z_scores.abs() > 3).sum()

print("Number of outliers per feature (Z-Score Method):")
print(outliers_per_feature)
```

```
Number of outliers per feature (Z-Score Method):
loan_amnt              144
int_rate               604
installment           4026
annual_inc            2581
dti                     10
open_acc              3898
pub_rec               6447
revol_bal             3933
revol_util              14
total_acc             2699
mort_acc              5453
pub_rec_bankruptcies  1873
dtype: int64
```

Interpretation: There are outliers present in the dataset.

```
# Capping of outliers

# Calculate IQR
Q1 = train_data[num_cols].quantile(0.25)
Q3 = train_data[num_cols].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds for capping
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Cap outliers
train_data[num_cols] = train_data[num_cols].clip(lower=lower_bound, upper=upper_bound, axis=1)
```

Interpretation: handled outliers by capping (Winsorizing) them by limiting extreme values.

5. Feature Engineering

```
# Feature Engineering

# Create new feature: Loan-to-Income Ratio
train_data['loan_to_income'] = train_data['loan_amnt'] / (train_data['annual_inc'] + 1)

# Convert 'term' column into numerical (36 months -> 36, 60 months -> 60)
train_data['term'] = train_data['term'].str.extract('(\d+)').astype(float)

# Convert 'emp_length' column into numerical
train_data['emp_length'] = train_data['emp_length'].replace({'< 1 year': 0,'1 year': 1,'2 years': 2,'3 years': 3,'4 years': 4,'5 years': 5,'6
                                                              '7 years': 7,'8 years': 8,'9 years': 9, '10+ years': 10, 'n/a': np.nan}).astype(

# Create a binary feature indicating whether the applicant owns a home
train_data['is_homeowner'] = train_data['home_ownership'].apply(lambda x: 1 if x in ['MORTGAGE', 'OWN'] else 0)

# Feature Engineering for test_data
test_data['loan_to_income'] = test_data['loan_amnt'] / (test_data['annual_inc'] + 1)
test_data['term'] = test_data['term'].str.extract('(\d+)').astype(float)
test_data['emp_length'] = test_data['emp_length'].replace({'< 1 year': 0, '1 year': 1, '2 years': 2,
                                                            '3 years': 3, '4 years': 4, '5 years': 5,
                                                            '6 years': 6, '7 years': 7, '8 years': 8,
                                                            '9 years': 9, '10+ years': 10, 'n/a': np.nan}).astype(float)
test_data['is_homeowner'] = test_data['home_ownership'].apply(lambda x: 1 if x in ['MORTGAGE', 'OWN'] else 0)
```

Interpretation: Added new features and removed reducndant features

6. Encoding

```
# 4. Label encoding for categorical columns
label_encoders = {}
```

```
label_encoders = {}
for column in train_data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    train_data[column] = le.fit_transform(train_data[column].astype(str))
    label_encoders[column] = le


# Label encoding for test data
for column in test_data.select_dtypes(include=['object']).columns:
    if column in label_encoders:
        test_data[column] = test_data[column].map(lambda x: label_encoders[column].transform([x])[0] if x in label_encoders[column].classes_ 
    else:
      #Assign unknown labels a default value
        test_data[column] = -1
```

Interpretation: Categorical fields are encoded into numerical vectors

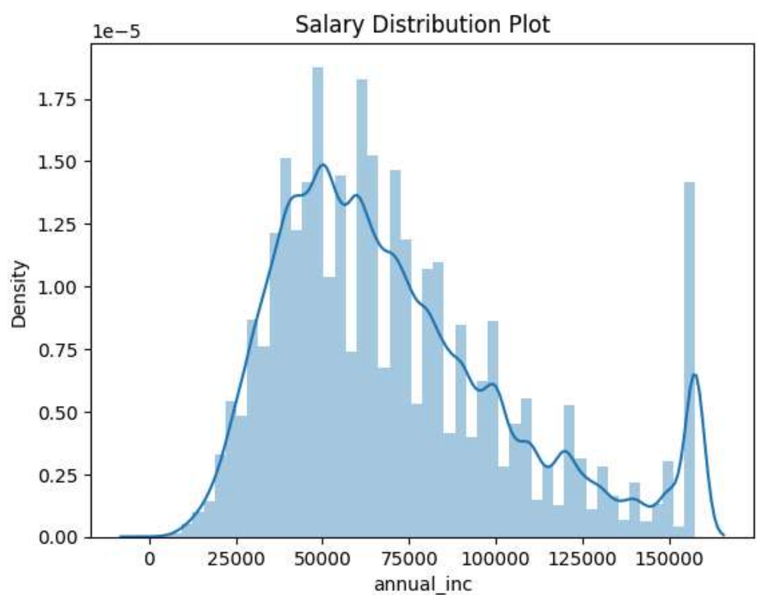
B. Other EDAs

```
#Descriptive Statistics

train_data.describe()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownership	annual_i
count	316970.000000	316970.000000	316970.000000	316970.000000	316970.000000	316970.000000	316970.000000	316970.000000	316970.0000
mean	14122.829369	41.718977	13.634503	428.424574	1.823154	11.088254	6.124034	2.900439	70988.7464
std	8354.792864	10.224924	4.454000	240.343783	1.334792	6.606825	3.662754	1.924452	34316.1103
min	500.000000	36.000000	5.320000	16.080000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	8000.000000	36.000000	10.490000	250.330000	1.000000	6.000000	3.000000	1.000000	45000.0000
50%	12000.000000	36.000000	13.330000	375.490000	2.000000	10.000000	7.000000	1.000000	64000.0000
75%	20000.000000	36.000000	16.550000	568.107500	3.000000	15.000000	10.000000	5.000000	90000.0000
max	38000.000000	60.000000	25.640000	1044.773750	6.000000	34.000000	10.000000	5.000000	157500.0000

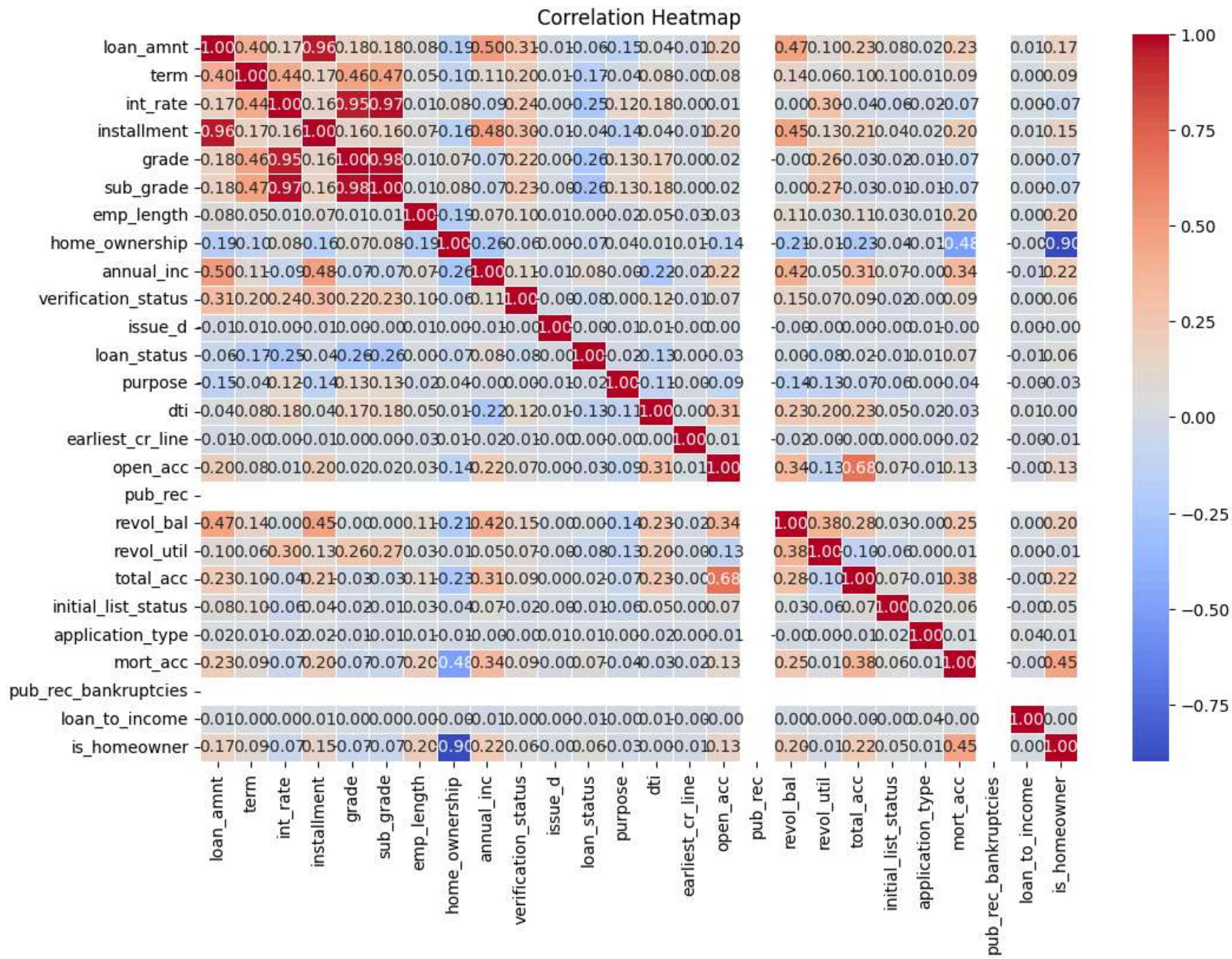
8 rows × 26 columns

```
# Annual Income distribution
plt.title('Salary Distribution Plot')
sns.distplot(train_data['annual_inc'])
plt.show()
```



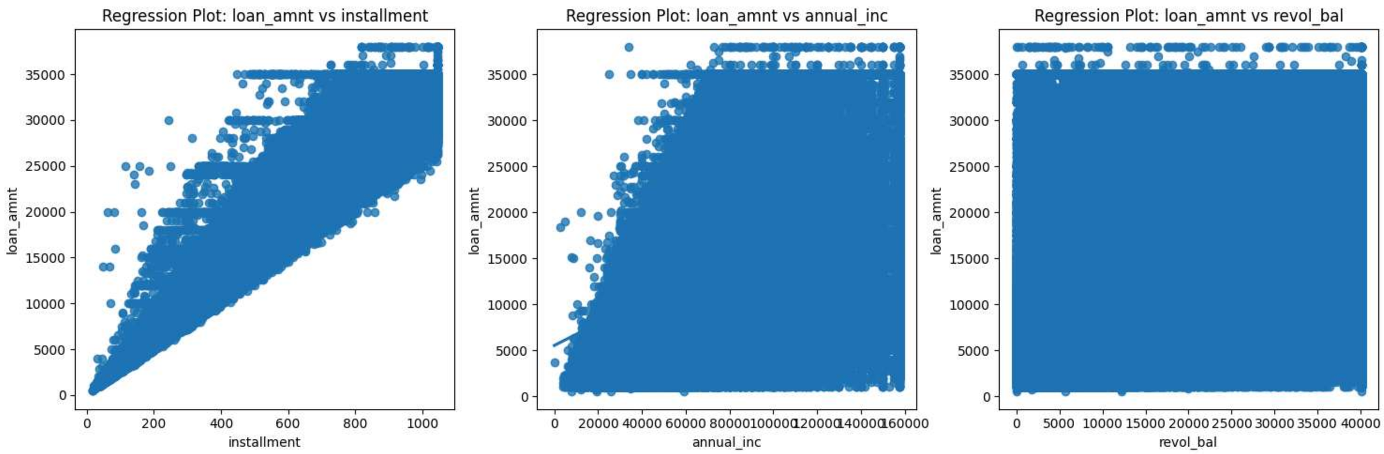
Heat map

```
# 4. Correlation Heatmap for Numerical Features
plt.figure(figsize=(12, 8))
correlation_matrix = train_data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



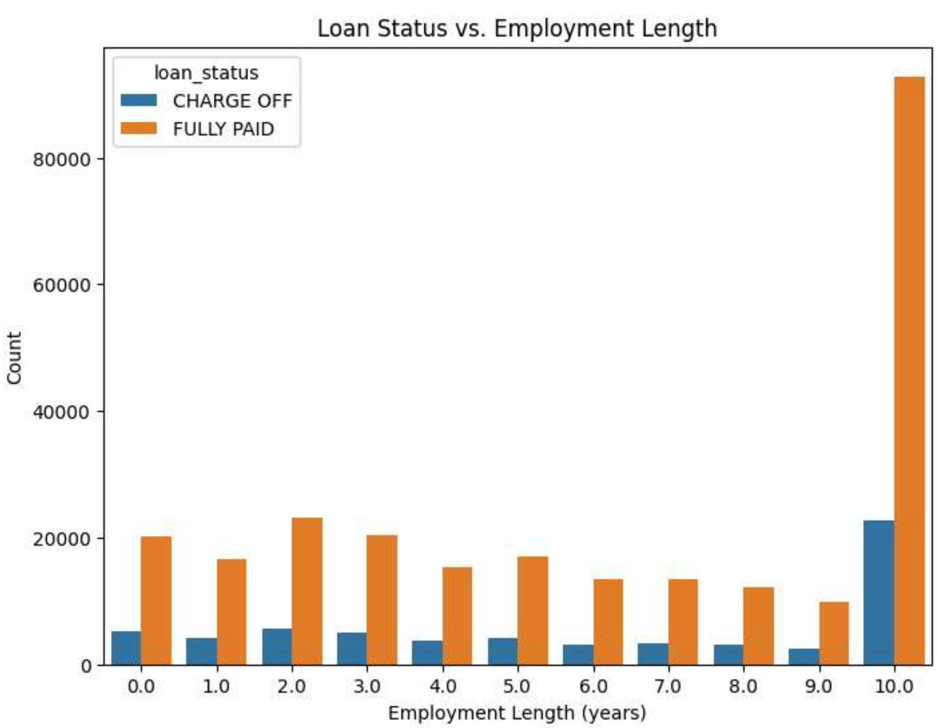
Regression Plots

```
#Regression Plot

x_vars = ['installment', 'annual_inc', 'revol_bal']
y_var = 'loan_amnt'

# egression plots
plt.figure(figsize=(15, 5))
for i, col in enumerate(x_vars, 1):
    plt.subplot(1, 3, i)
    sns.regplot(x=train_data[col], y=train_data[y_var])
    plt.title(f'Regression Plot: {y_var} vs {col}')
    plt.xlabel(col)
    plt.ylabel(y_var)

plt.tight_layout()
plt.show()
```



Bar Plot - Loan Status vs Employment Length

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

# Assuming 'loan_status' is coded as 1 for "FULLY PAID" and 0 for "CHARGE OFF"
train_data['loan_status'] = train_data['loan_status'].map({1: 'FULLY PAID', 0: 'CHARGE OFF'})

# Plotting the bar plot
plt.figure(figsize=(8, 6))
sns.countplot(x='emp_length', hue='loan_status', data=train_data, hue_order=['CHARGE OFF','FULLY PAID'])
plt.title('Loan Status vs. Employment Length')
plt.xlabel('Employment Length (years)')
plt.ylabel('Count')
plt.show()
```




```
# Drop redundant features
```

```
columns_to_drop = ['loan_amnt', 'annual_inc', 'home_ownership']
train_data = train_data.drop(columns=columns_to_drop, errors='ignore')
test_data = test_data.drop(columns=columns_to_drop, errors='ignore')
```

Split Dataset

```
# Split the data into training and test sets

X = train_data.drop('loan_status', axis=1)  #Features
y = train_data['loan_status']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

6. Scaling

```
# Initialize the Standard Scaler
scaler = StandardScaler()

# Fit the scaler on the resampled training data and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the validation data
X_test_scaled = scaler.transform(X_test)

# scaling on test data
test_data_scaled = scaler.transform(test_data)
```

Build and Evaluate Models - Logistic Regression, Decision Tree,Random Forest

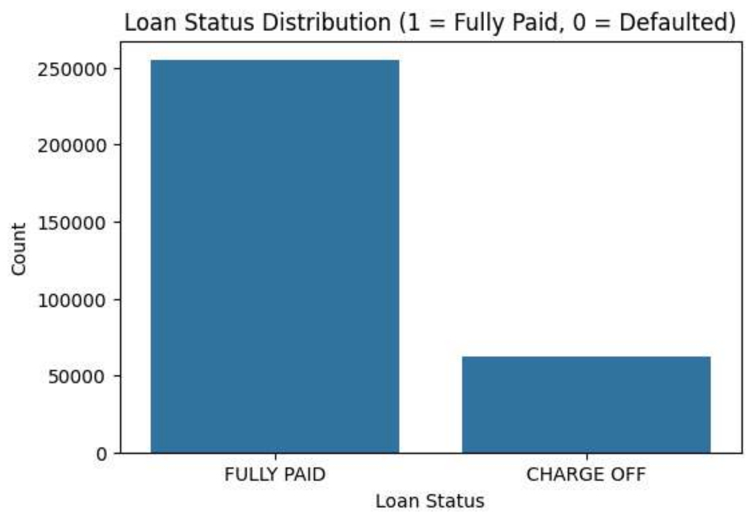
```
# Initialize models
models = {
    'Logistic Regression': LogisticRegression(C=0.1, solver='liblinear'),
    'Decision Tree': DecisionTreeClassifier(max_depth=10, min_samples_split=10, min_samples_leaf=5),
    'Random Forest': RandomForestClassifier(n_estimators=200, max_depth=20, min_samples_split=10, min_samples_leaf=5, random_state=42)
}

# Train and evaluate models
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    print(f'{name} Accuracy: {accuracy_score(y_test, y_pred)}')
    print(f'{name} Classification Report:\n{classification_report(y_test, y_pred)}')
    print(f'{name} Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}')
    print('-' * 60)
```

```
Logistic Regression Accuracy: 0.8044452156355492
Logistic Regression Classification Report:
              precision    recall  f1-score   support

  CHARGE OFF       0.54      0.09      0.15     12558
  FULLY PAID       0.81      0.98      0.89     50836

    accuracy                           0.80     63394
   macro avg       0.68      0.53      0.52     63394
weighted avg       0.76      0.80      0.74     63394

Logistic Regression Confusion Matrix:
[[ 1068 11490]
 [  907 49929]]
------------------------------------------------------------
Decision Tree Accuracy: 0.7990346089535287
Decision Tree Classification Report:
              precision    recall  f1-score   support

  CHARGE OFF       0.46      0.08      0.14     12558
  FULLY PAID       0.81      0.98      0.89     50836

    accuracy                           0.80     63394
   macro avg       0.64      0.53      0.51     63394
weighted avg       0.74      0.80      0.74     63394

Decision Tree Confusion Matrix:
```

```
[[ 1026 11532]
 [ 1208 49628]]
------------------------------------------------------------
Random Forest Accuracy: 0.8046029592705934
Random Forest Classification Report:
              precision    recall  f1-score   support

  CHARGE OFF       0.56      0.07      0.12     12558
  FULLY PAID       0.81      0.99      0.89     50836

    accuracy                           0.80     63394
   macro avg       0.68      0.53      0.50     63394
weighted avg       0.76      0.80      0.74     63394

Random Forest Confusion Matrix:
[[  831 11727]
 [  660 50176]]
------------------------------------------------------------
```

Check Dataset balance

```
# Data Visualization to understand the dataset balance
plt.figure(figsize=(6, 4))
sns.countplot(x='loan_status', data=train_data)
plt.title('Loan Status Distribution (1 = Fully Paid, 0 = Defaulted)')
plt.xlabel('Loan Status')
plt.ylabel('Count')
plt.show()
```



Build and Evaluate Model(Logistic Regression,Decision Tree,Random Forest) with SMOTE(to balance dataset)

```
# Use SMOTE to balance the training data
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)

# Initialize the Standard Scaler
scaler1 = StandardScaler()

# Fit the scaler on the resampled training data and transform
X_train_scaled1 = scaler1.fit_transform(X_train)

# Transform the validation data
X_test_scaled1 = scaler1.transform(X_test)


# Initialize models
models = {
    'Logistic Regression': LogisticRegression(C=0.1, solver='liblinear'),
    'Decision Tree': DecisionTreeClassifier(max_depth=10, min_samples_split=10, min_samples_leaf=5),
    'Random Forest': RandomForestClassifier(n_estimators=200, max_depth=20, min_samples_split=10, min_samples_leaf=5, random_state=42)
```

```
}

# Train and evaluate models
for name, model in models.items():
    model.fit(X_train_scaled1, y_train)
    y_pred = model.predict(X_test_scaled1)
    print(f'{name} Accuracy: {accuracy_score(y_test, y_pred)}')
    print(f'{name} Classification Report:\n{classification_report(y_test, y_pred)}')
    print(f'{name} Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}')
    print('-' * 60)
```

```
Logistic Regression Accuracy: 0.6795280310439473
Logistic Regression Classification Report:
              precision    recall  f1-score   support

  CHARGE OFF       0.30      0.47      0.37     12558
  FULLY PAID       0.85      0.73      0.79     50836

    accuracy                           0.68     63394
   macro avg       0.57      0.60      0.58     63394
weighted avg       0.74      0.68      0.70     63394

Logistic Regression Confusion Matrix:
[[ 5870  6688]
 [13628 37208]]
------------------------------------------------------------
Decision Tree Accuracy: 0.7397072278133577
Decision Tree Classification Report:
              precision    recall  f1-score   support

  CHARGE OFF       0.32      0.29      0.31     12558
  FULLY PAID       0.83      0.85      0.84     50836

    accuracy                           0.74     63394
   macro avg       0.58      0.57      0.57     63394
weighted avg       0.73      0.74      0.73     63394

Decision Tree Confusion Matrix:
[[ 3626  8932]
 [ 7569 43267]]
------------------------------------------------------------
Random Forest Accuracy: 0.7795848187525634
Random Forest Classification Report:
              precision    recall  f1-score   support

  CHARGE OFF       0.40      0.23      0.29     12558
  FULLY PAID       0.83      0.92      0.87     50836

    accuracy                           0.78     63394
   macro avg       0.61      0.57      0.58     63394
weighted avg       0.74      0.78      0.75     63394

Random Forest Confusion Matrix:
[[ 2836  9722]
 [ 4251 46585]]
------------------------------------------------------------
```

Prediction on Test CSV

```
# Choose the best model- Random Forest
best_model = models['Random Forest']

# Predict on test_data
test_predictions = best_model.predict(test_data_scaled)

# Convert predictions into a DataFrame
test_data_predictions = pd.DataFrame(test_predictions, columns=['Predicted Loan Status'])

# Save predictions to CSV
test_data_predictions.to_csv('test_predictions.csv', index=False)

# Display few predicions
print(test_data_predictions.head())
```

```
  Predicted Loan Status
0            CHARGE OFF
1            CHARGE OFF
2            CHARGE OFF
```

```
3            CHARGE OFF
4            CHARGE OFF
```