```python
#depency import
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
import numpy as np
```

## Data Import

```python
#import data
data_full = pd.read_csv("./Pre-
Super_Day_candidate_dataset__28candidate_29 2.csv")

data_full.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 14 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   User ID                       100000 non-null  object
 1   applications                  100000 non-null  int64
 2   Reason                        100000 non-null  object
 3   Loan_Amount                   100000 non-null  int64
 4   FICO_score                    100000 non-null  int64
 5   Fico_Score_group              100000 non-null  object
 6   Employment_Status             100000 non-null  object
 7   Employment_Sector             93593 non-null   object
 8   Monthly_Gross_Income          100000 non-null  int64
 9   Monthly_Housing_Payment       100000 non-null  int64
 10  Ever_Bankrupt_or_Foreclose    100000 non-null  int64
 11  Lender                        100000 non-null  object
 12  Approved                      100000 non-null  int64
 13  bounty                        100000 non-null  int64
dtypes: int64(8), object(6)
memory usage: 10.7+ MB
```

```python
#data description
data_full.describe()
```

```
       applications    Loan_Amount    FICO_score  Monthly_Gross_Income
\
count      100000.0  100000.000000  100000.00000         100000.000000

mean            1.0   45234.350000     629.34961           5871.899350

std             0.0   28705.453665      88.66160           2882.939639

min             1.0    5000.000000     300.00000           2000.000000
```

```
25%                 1.0    20000.000000      572.00000              3704.000000

50%                 1.0    40000.000000      634.00000              5172.500000

75%                 1.0    70000.000000      693.00000              7631.000000

max                 1.0   100000.000000      850.00000             19997.000000


       Monthly_Housing_Payment  Ever_Bankrupt_or_Foreclose
Approved  \
count             100000.000000               100000.000000
100000.000000
mean                1649.693970                    0.022460
0.109760
std                  623.443127                    0.148175
0.312592
min                  300.000000                    0.000000
0.000000
25%                 1229.000000                    0.000000
0.000000
50%                 1665.000000                    0.000000
0.000000
75%                 2046.000000                    0.000000
0.000000
max                 3300.000000                    1.000000
1.000000

              bounty
count  100000.000000
mean       26.415000
std        78.385644
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max       350.000000
```

# Data preprocessing

- Dropping Features: I have decided to drop `User_ID` columns because it is unique for every data point. I also drop the `applications` column because it is redundant.
- Label Encoding: I have also decided to label encode `object` (i.e. `String`) columns, and keep track of which categories these encodings fall into.

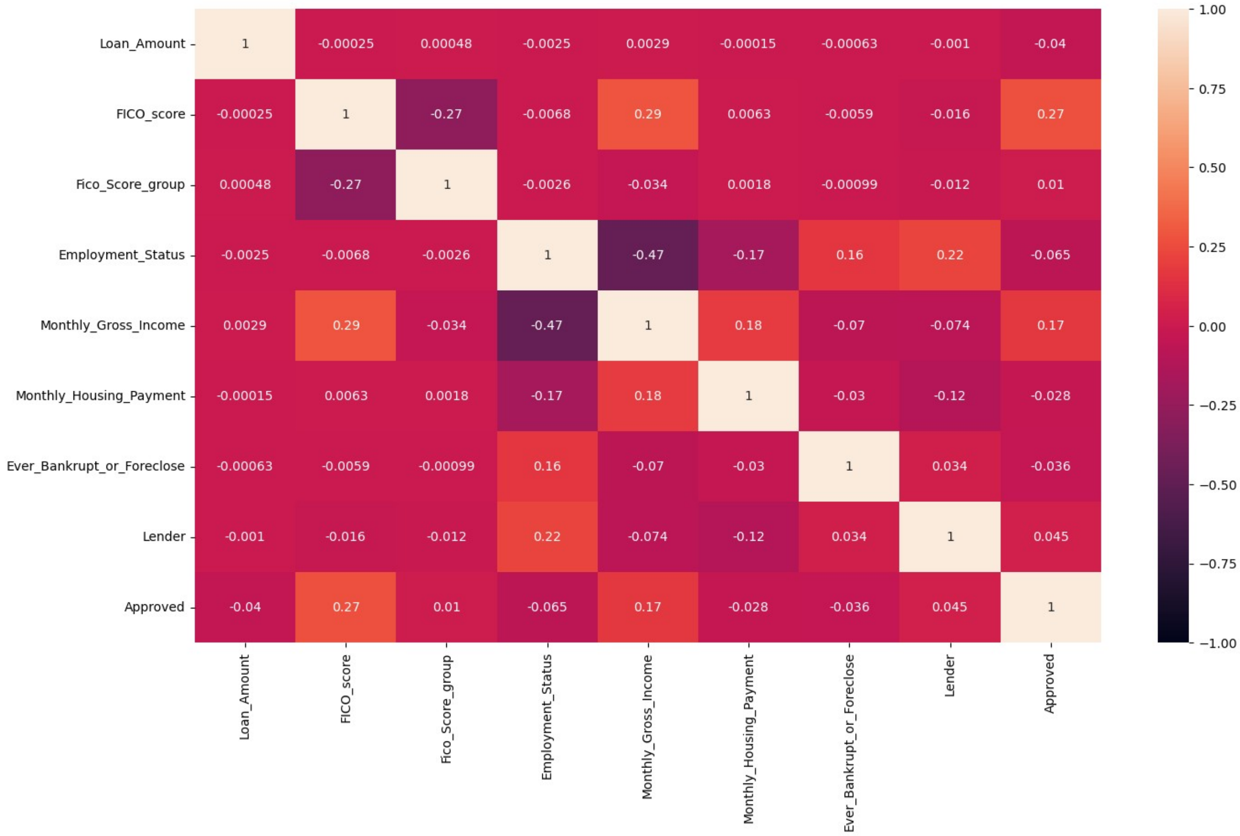
```
#drop useless features
to_drop = ["User ID","applications","Reason","Employment_Sector"]
data= data_full.drop (data_full [to_drop], axis=1)
```

```python
#encode non-numerical features

from sklearn.preprocessing import LabelEncoder
label_encoder  = LabelEncoder()


categorical_columns =
["Fico_Score_group","Employment_Status","Lender"]

#transform each column
for column in categorical_columns:
    data[column] = label_encoder.fit_transform(data[column])
```

# 1. Explore the variables relationship with approvability:

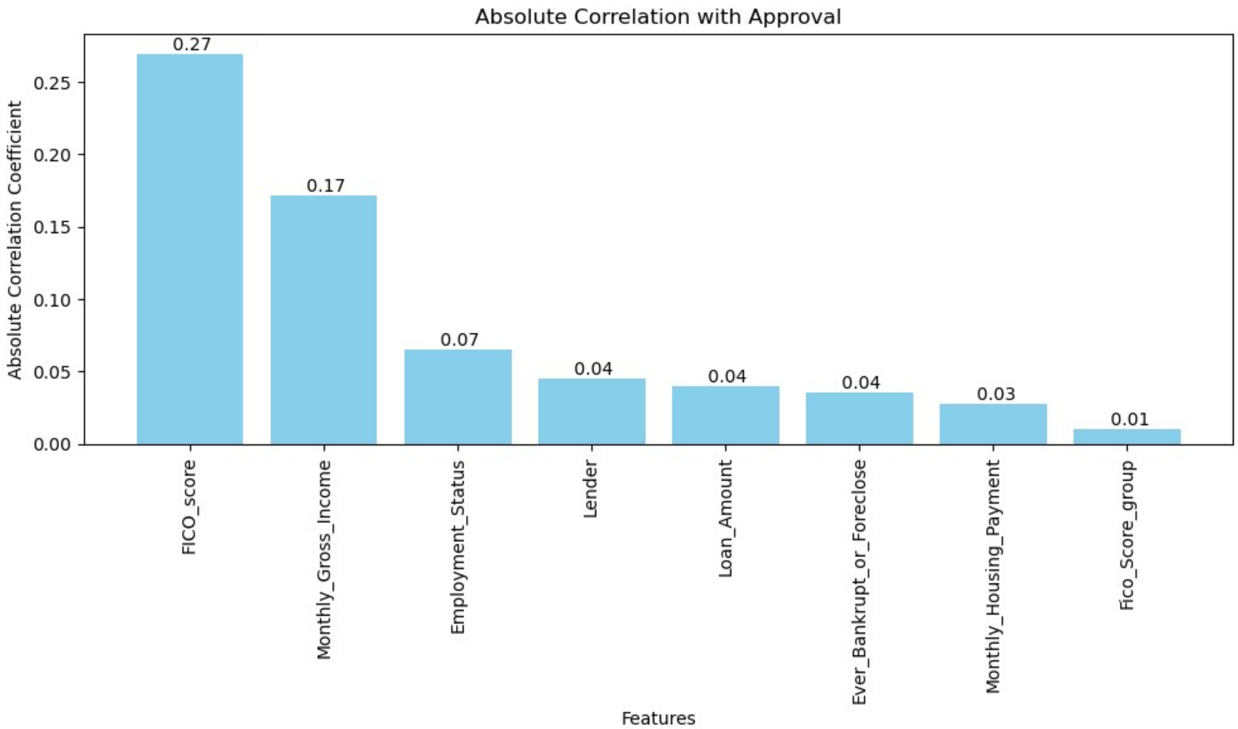
Feature relevance was assessed using correlation analysis and Random Forest feature importance.

```python
#define correlation matrix.
correlation_matrix = data.drop("bounty",axis  = 1).corr()

#visualize correlation
plt.figure(figsize = (16, 9))
sns.heatmap(correlation_matrix, annot = True, vmin = -1, vmax = 1)
plt.show()
```




```python
# rank correlation with approval based on absolute value of
correlation coefficient.
correlation_with_approved =
correlation_matrix["Approved"].abs().sort_values(ascending = False)


correlation_with_approved = correlation_with_approved.drop("Approved")
#plot results
plt.figure(figsize=(10,6))
bars =
plt.bar(correlation_with_approved.index,correlation_with_approved.valu
es,color = "skyblue" )
plt.title("Absolute Correlation with Approval")
plt.xlabel("Features")
plt.ylabel("Absolute Correlation Coefficient")
plt.xticks(rotation = 90)
plt.tight_layout()


# Annotate bars with their corresponding coeffcients for better
readability.
for bar in bars :
    height = bar. get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height,
```

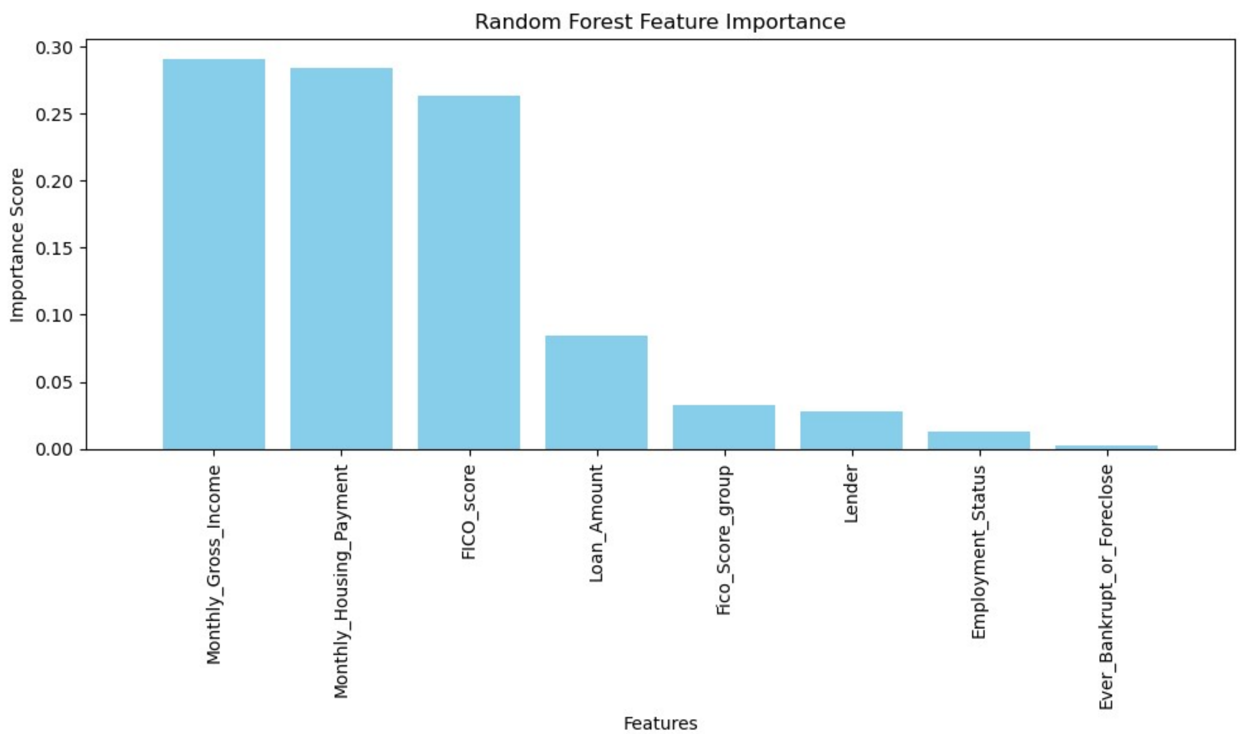
```
f'{height: .2f}', ha = 'center', va = 'bottom')

plt.show()
```




Random Forest Feature Importance

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

#dataset split
X = data.drop(["Approved","bounty"],axis =1)
y = data["Approved"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3, random_state = 1)

# Initialize and train the model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Get feature importances
importances = rf.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]

# Visualize feature importances
```

```
plt.figure(figsize=(10, 6))
plt.title("Random Forest Feature Importance")
plt.bar(range(X.shape[1]), importances[indices], align="center", color
= "skyblue")
plt.xticks(range(X.shape[1]), X.columns[indices], rotation=90)
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.xlim([-1, X.shape[1]])
plt.tight_layout()
plt.show()
```



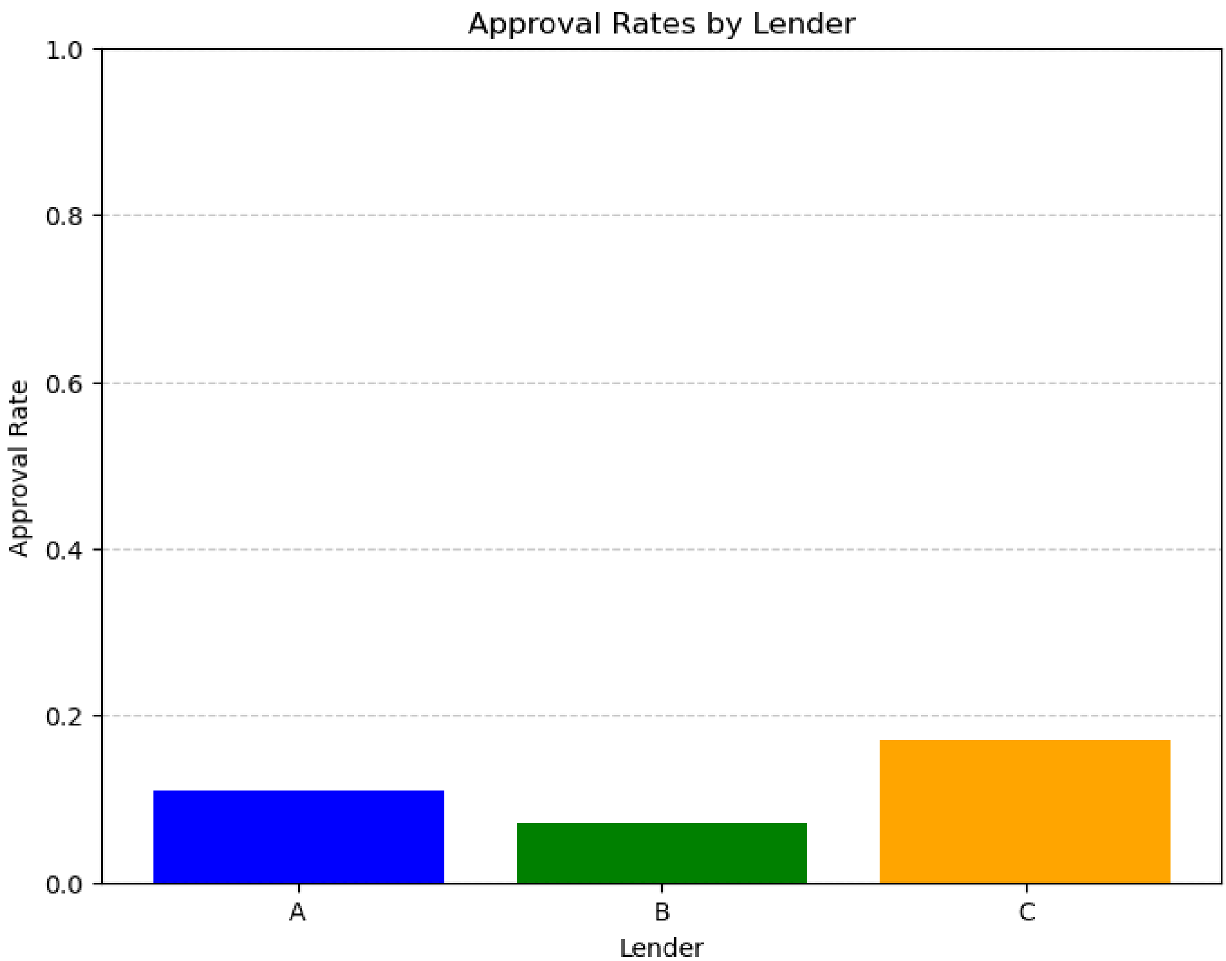

# Exploring Lender Approval Rate

## Average approval rate

```
# Group the data by lender and calculate the average approval rate
lender_approval_rates = data_full.groupby('Lender')['Approved'].mean()

# Display the average approval rates for each lender
print("Average Approval Rates by Lender:")
print(lender_approval_rates)


# Visualize approval rates using a bar plot
plt.figure(figsize=(8, 6))
plt.bar(lender_approval_rates.index, lender_approval_rates.values,
color=['blue', 'green', 'orange'])
```

```
plt.xlabel('Lender')
plt.ylabel('Approval Rate')
plt.title('Approval Rates by Lender')
plt.ylim(0, 1)  # Set y-axis limits to ensure proper visualization of approval rates
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add gridlines for better readability
plt.show()
```

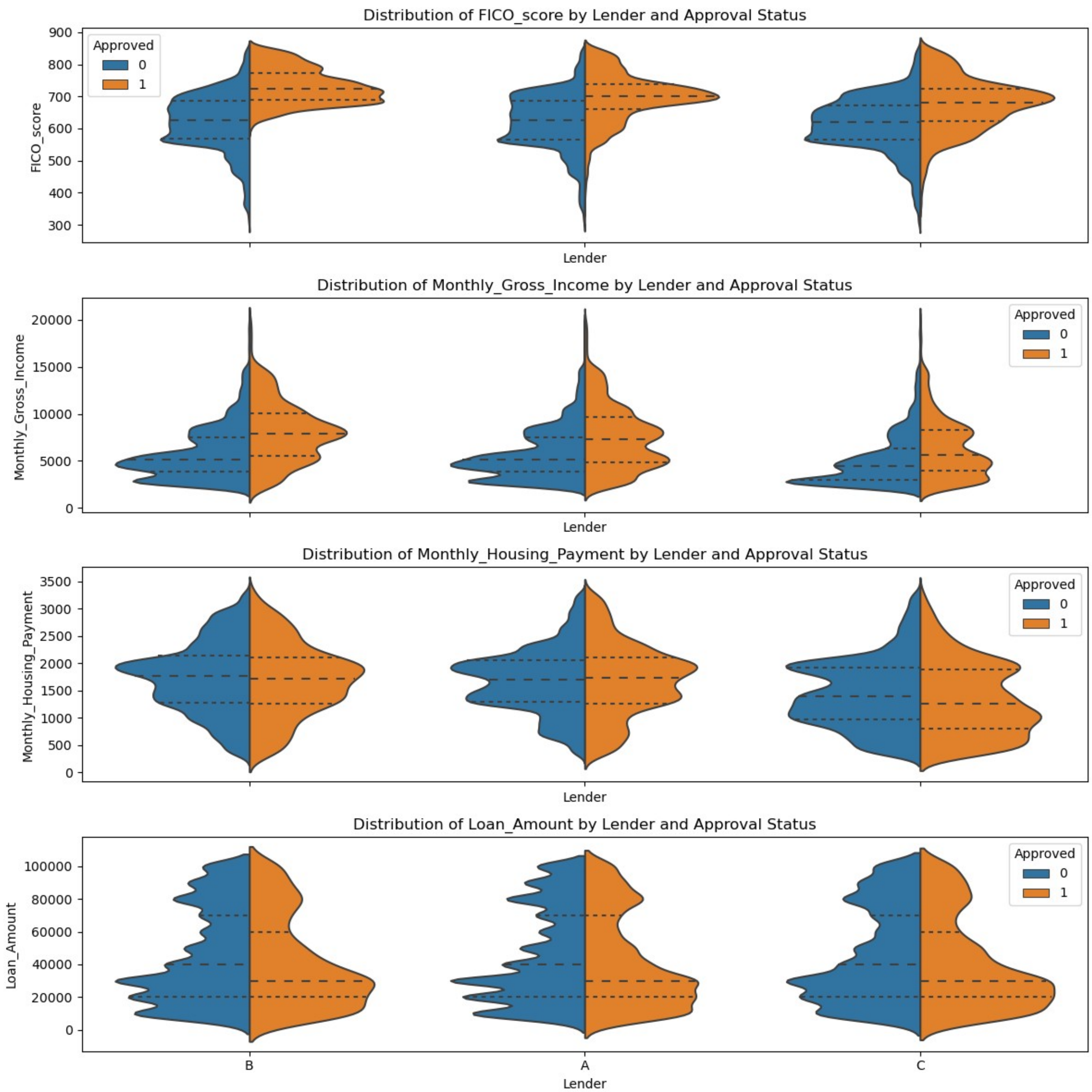
```
Average Approval Rates by Lender:
Lender
A    0.109655
B    0.071273
C    0.170571
Name: Approved, dtype: float64
```




## Explore differences in costumers based on lender

```
#Define relevant variables for comparison
variables_of_interest = ['FICO_score', 'Monthly_Gross_Income',
```

```python
'Monthly_Housing_Payment',"Loan_Amount"]

# Set up the figure with subplots
fig, axes = plt.subplots(len(variables_of_interest), 1, figsize=(12,
12), sharex=True)

# Plot violinplots for each variable of interest
for i, variable in enumerate(variables_of_interest):
    sns.violinplot(data=data_full, x='Lender', y=variable,
hue='Approved', ax=axes[i], split=True, inner='quartile')
    axes[i].set_title(f'Distribution of {variable} by Lender and
Approval Status')
    axes[i].set_xlabel('Lender')
    axes[i].set_ylabel(variable)
    axes[i].legend(title='Approved')

# Adjust layout
plt.tight_layout()
plt.show()
```




# explore feature importance based on lender

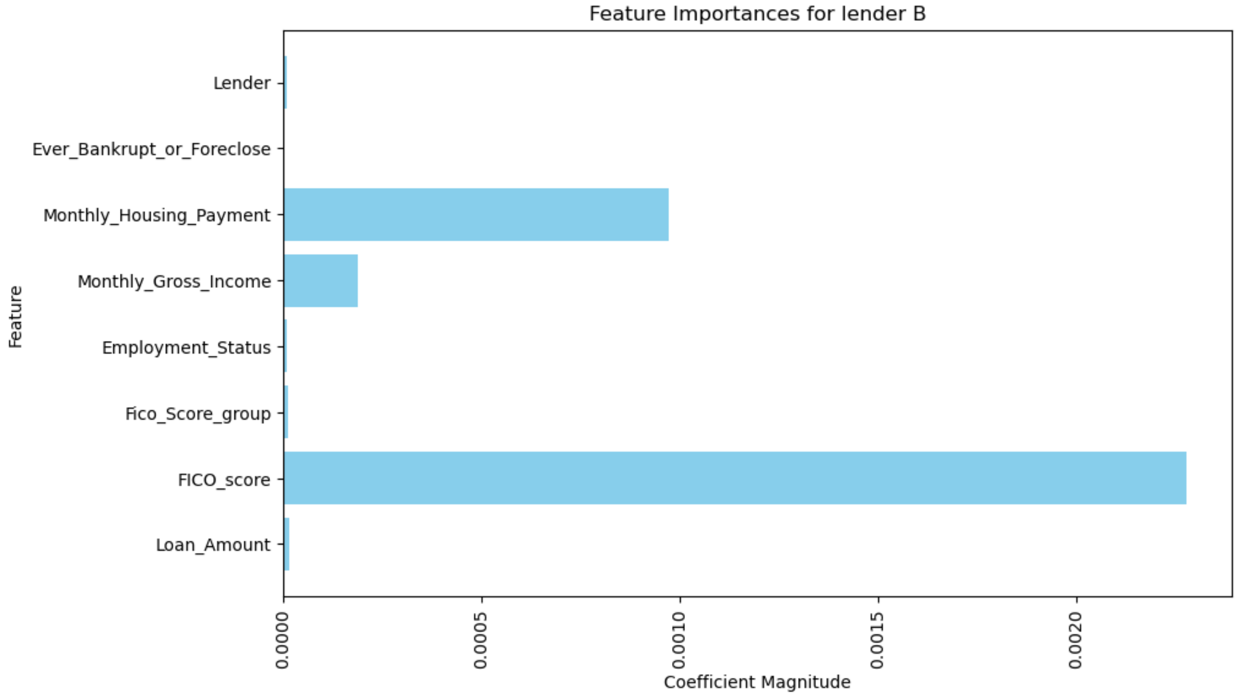
```
# For demonstration, let's use logistic regression to identify
important variables
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Define features and target variable
features = ['FICO Score', 'Monthly Gross Income', 'Employment Status']
target = 'Approved'
```

```python
# Fit a logistic regression model for each lender
lenders = data_full['Lender'].unique()

for lender in lenders:
    print(f"\nLogistic Regression Model for Lender: {lender}")

    # Filter data for the current lender
    X_train_lender = X_train[data_full['Lender'] == lender]
    y_train_lender = y_train[data_full['Lender'] == lender]
    X_test_lender = X_test[data_full['Lender'] == lender]
    y_test_lender = y_test[data_full['Lender'] == lender]

    # Create and fit logistic regression model
    model = LogisticRegression()
    model.fit(X_train_lender, y_train_lender)

    # Evaluate model performance
    train_accuracy = model.score(X_train_lender, y_train_lender)
    test_accuracy = model.score(X_test_lender, y_test_lender)
    print(f"Train Accuracy: {train_accuracy:.2f}")
    print(f"Test Accuracy: {test_accuracy:.2f}")


    coefficients = np.abs(model.coef_[0])
    feature_names = X.columns

    # Plotting
    plt.figure(figsize=(10, 6))
    plt.barh(feature_names, coefficients, color='skyblue')
    plt.xlabel('Coefficient Magnitude')
    plt.ylabel('Feature')
    plt.xticks(rotation = 90)
    plt.title('Feature Importances for lender '+lender)
    plt.show()

    # Generate classification report
    y_pred = model.predict(X_test_lender)
    print(classification_report(y_test_lender, y_pred))
```

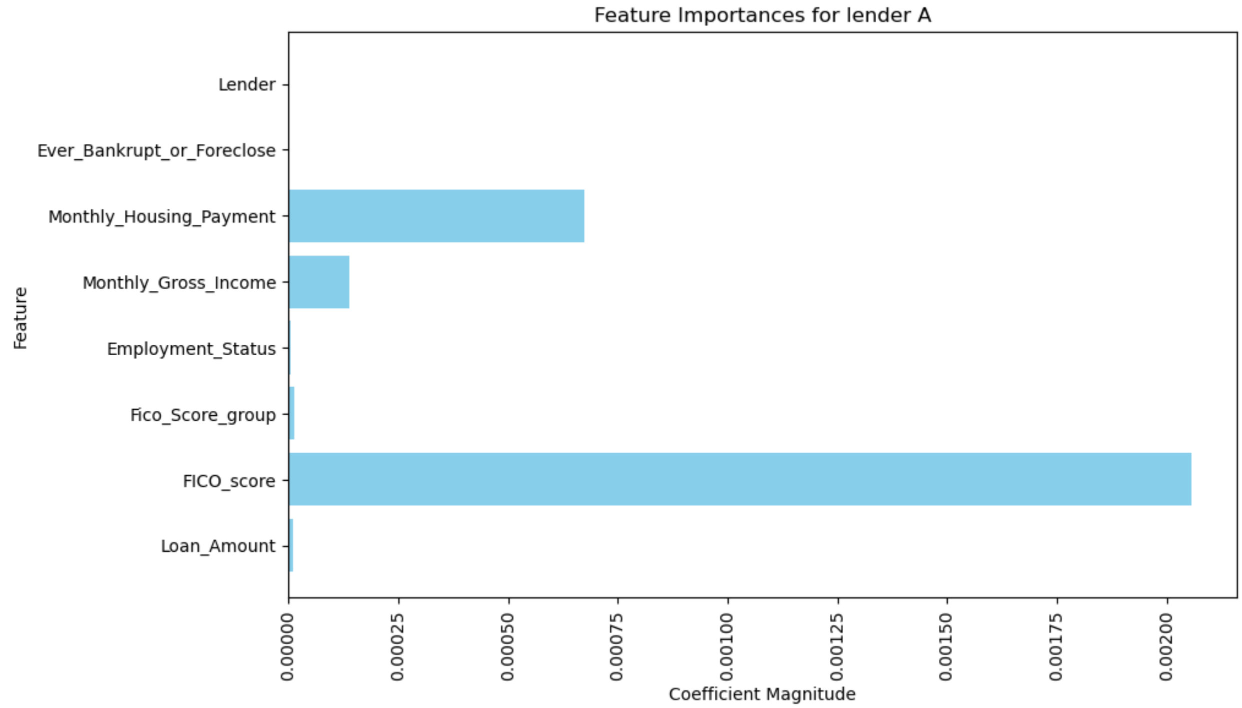
```
Logistic Regression Model for Lender: B
Train Accuracy: 0.93
Test Accuracy: 0.93
```

```
/var/folders/yn/ftl55p110bl7f4vqh175t6mh0000gn/T/
ipykernel_94964/3427714349.py:18: UserWarning: Boolean Series key will
be reindexed to match DataFrame index.
  X_train_lender = X_train[data_full['Lender'] == lender]
/var/folders/yn/ftl55p110bl7f4vqh175t6mh0000gn/T/ipykernel_94964/34277
14349.py:20: UserWarning: Boolean Series key will be reindexed to
```

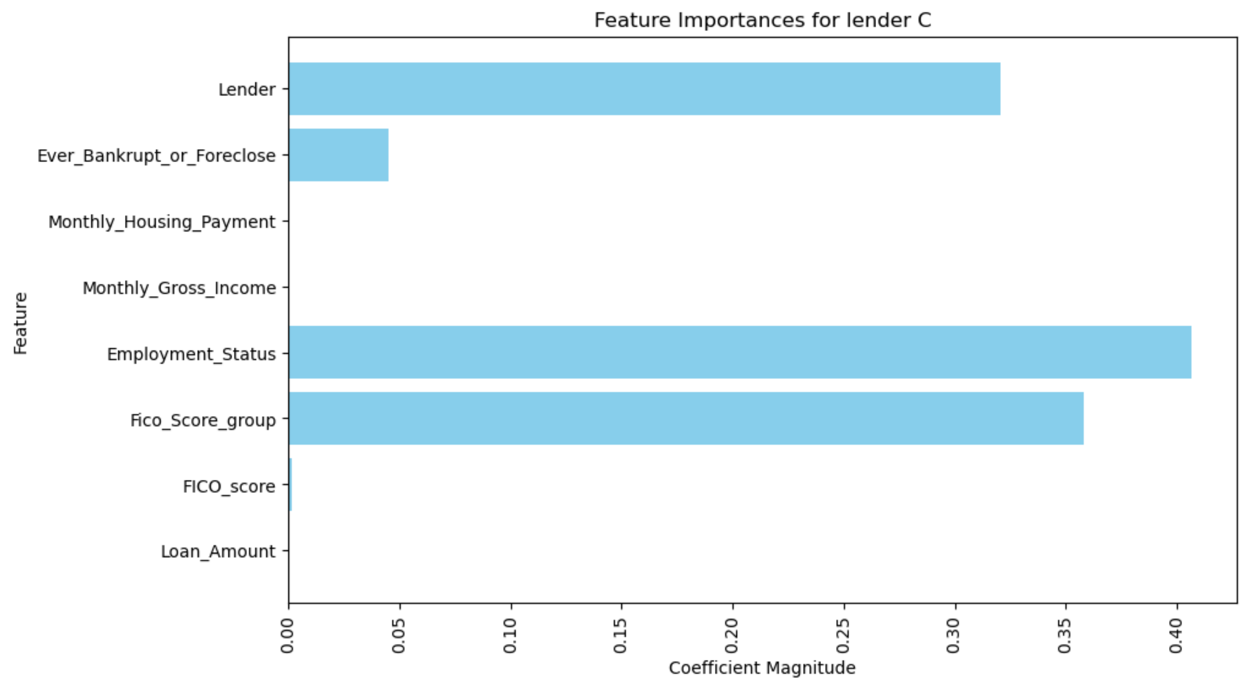
```
match DataFrame index.
  X_test_lender = X_test[data_full['Lender'] == lender]
```




```
              precision    recall  f1-score   support

           0       0.93      1.00      0.96      7671
           1       0.57      0.01      0.01       560

    accuracy                           0.93      8231
   macro avg       0.75      0.50      0.49      8231
weighted avg       0.91      0.93      0.90      8231


Logistic Regression Model for Lender: A
Train Accuracy: 0.89
Test Accuracy: 0.89

/var/folders/yn/ftl55p110bl7f4vqh175t6mh0000gn/T/
ipykernel_94964/3427714349.py:18: UserWarning: Boolean Series key will
be reindexed to match DataFrame index.
  X_train_lender = X_train[data_full['Lender'] == lender]
/var/folders/yn/ftl55p110bl7f4vqh175t6mh0000gn/T/ipykernel_94964/34277
14349.py:20: UserWarning: Boolean Series key will be reindexed to
match DataFrame index.
  X_test_lender = X_test[data_full['Lender'] == lender]
```




```
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     14681
           1       0.40      0.00      0.00      1780

    accuracy                           0.89     16461
   macro avg       0.65      0.50      0.47     16461
weighted avg       0.84      0.89      0.84     16461


Logistic Regression Model for Lender: C
Train Accuracy: 0.83
Test Accuracy: 0.83

/var/folders/yn/ftl55p110bl7f4vqh175t6mh0000gn/T/
ipykernel_94964/3427714349.py:18: UserWarning: Boolean Series key will
be reindexed to match DataFrame index.
  X_train_lender = X_train[data_full['Lender'] == lender]
/var/folders/yn/ftl55p110bl7f4vqh175t6mh0000gn/T/ipykernel_94964/34277
14349.py:20: UserWarning: Boolean Series key will be reindexed to
match DataFrame index.
  X_test_lender = X_test[data_full['Lender'] == lender]
```




```
              precision    recall  f1-score   support

           0       0.83      1.00      0.90      4367
           1       0.67      0.03      0.05       941

    accuracy                           0.83      5308
   macro avg       0.75      0.51      0.48      5308
weighted avg       0.80      0.83      0.75      5308
```

## PART A

```python
lenders = data_full['Lender'].unique()

imp_features = ['FICO_score', 'Loan_Amount', 'Monthly_Gross_Income',
'Monthly_Housing_Payment', 'bounty']

for feature in imp_features:
    group = data_full.groupby('Lender')[feature]
    mean = group.mean()
    print(mean)
```

```
Lender
A    630.125727
B    630.246509
C    625.500971
Name: FICO_score, dtype: float64
Lender
```

```
A    45257.909091
B    45223.090909
C    45178.000000
Name: Loan_Amount, dtype: float64
Lender
A    5989.508036
B    5994.417745
C    5309.743143
Name: Monthly_Gross_Income, dtype: float64
Lender
A    1679.922945
B    1730.975636
C    1426.960286
Name: Monthly_Housing_Payment, dtype: float64
Lender
A    27.413636
B    24.945455
C    25.585714
Name: bounty, dtype: float64
```

PART B

```
# Create a function to determine the best lender for each customer
based on characteristics
def determine_best_lender(row):
    # Define criteria for selecting the best lender (e.g., based on
FICO score, income, etc.)
    # For demonstration purposes, let's assume the lender with the
highest average bounty is chosen
    best_lender = data.groupby('Lender')['bounty'].mean().idxmax()
    return best_lender

# Apply the function to determine the best lender for each customer
data['Best_Lender'] = data.apply(determine_best_lender, axis=1)

# Aggregate the total revenue per lender before matching
total_revenue_per_lender_before_matching = data.groupby('Lender')
['bounty'].sum()

# Aggregate the total revenue per lender after matching
total_revenue_per_lender_after_matching = data.groupby('Best_Lender')
['bounty'].sum()

# Calculate incremental revenue by subtracting total revenue per
lender before matching from after matching
incremental_revenue = total_revenue_per_lender_after_matching -
total_revenue_per_lender_before_matching

# Display the incremental revenue
#print("Incremental Revenue by Matching Customers to the Best
```

```
Lender:")
print(incremental_revenue)
```

```
Best_Lender
2    0
Name: bounty, dtype: int64
```