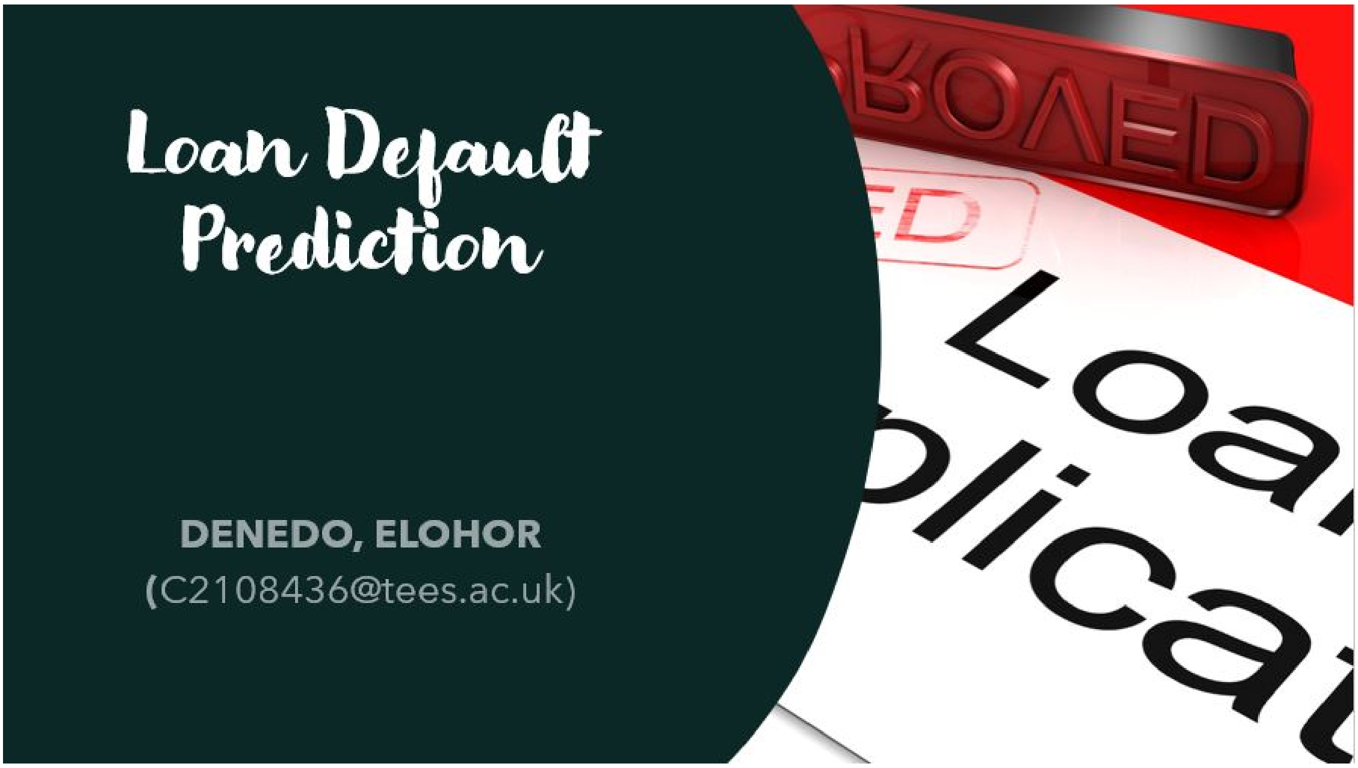


Contents

Load the Neccessary Libraries

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, accuracy_score, confusion_matrix,

from sklearn.model_selection import cross_val_score
import numpy as np


```

--------Data Exploration and Cleaning--------

In [2]:

```
# Load the dataset
df = pd.read_csv('Loan_Default.csv')
```

In [3]:

```python
# Understanding the data
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148670 entries, 0 to 148669
Data columns (total 34 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   ID                         148670 non-null  int64  
 1   year                       148670 non-null  int64  
 2   loan_limit                 145326 non-null  object 
 3   Gender                     148670 non-null  object 
 4   approv_in_adv              147762 non-null  object 
 5   loan_type                  148670 non-null  object 
 6   loan_purpose               148536 non-null  object 
 7   Credit_Worthiness          148670 non-null  object 
 8   open_credit                148670 non-null  object 
 9   business_or_commercial     148670 non-null  object 
 10  loan_amount                148670 non-null  int64  
 11  rate_of_interest           112231 non-null  float64
 12  Interest_rate_spread       112031 non-null  float64
 13  Upfront_charges            109028 non-null  float64
 14  term                       148629 non-null  float64
 15  Neg_ammortization          148549 non-null  object 
 16  interest_only              148670 non-null  object 
 17  lump_sum_payment           148670 non-null  object 
 18  property_value             133572 non-null  float64
 19  construction_type          148670 non-null  object 
 20  occupancy_type             148670 non-null  object 
 21  Secured_by                 148670 non-null  object 
 22  total_units                148670 non-null  object 
 23  income                     139520 non-null  float64
 24  credit_type                148670 non-null  object 
 25  Credit_Score               148670 non-null  int64  
 26  co-applicant_credit_type   148670 non-null  object 
 27  age                        148470 non-null  object 
 28  submission_of_application  148470 non-null  object 
 29  LTV                        133572 non-null  float64
 30  Region                     148670 non-null  object 
 31  Security_Type              148670 non-null  object 
 32  Status                     148670 non-null  int64  
 33  dtir1                      124549 non-null  float64
dtypes: float64(8), int64(5), object(21)
memory usage: 38.6+ MB
None
```

In [4]:

```
df.describe()
```

Out[4]:

	ID	year	loan_amount	rate_of_interest	Interest_rate_spread	Upfront_(
count	148670.000000	148670.0	1.486700e+05	112231.000000	112031.000000	109028
mean	99224.500000	2019.0	3.311177e+05	4.045476	0.441656	3224
std	42917.476598	0.0	1.839093e+05	0.561391	0.513043	3251
min	24890.000000	2019.0	1.650000e+04	0.000000	-3.638000	0
25%	62057.250000	2019.0	1.965000e+05	3.625000	0.076000	581
50%	99224.500000	2019.0	2.965000e+05	3.990000	0.390400	2596
75%	136391.750000	2019.0	4.365000e+05	4.375000	0.775400	4812
max	173559.000000	2019.0	3.576500e+06	8.000000	3.357000	60000

In [5]:

```
df.head()
```

Out[5]:

	ID	year	loan_limit	Gender	approv_in_adv	loan_type	loan_purpose	Credit_Worthin
0	24890	2019	cf	Sex Not Available	nopre	type1	p1	
1	24891	2019	cf	Male	nopre	type2	p1	
2	24892	2019	cf	Male	pre	type1	p1	
3	24893	2019	cf	Male	nopre	type1	p4	
4	24894	2019	cf	Joint	pre	type1	p1	

5 rows × 34 columns

Handling Missing Value

In [6]:

```
# check for missing values
df.isnull().sum()
```

Out[6]:

```
ID                               0
year                             0
loan_limit                    3344
Gender                           0
approv_in_adv                  908
loan_type                        0
loan_purpose                   134
Credit_Worthiness                0
open_credit                      0
business_or_commercial           0
loan_amount                      0
rate_of_interest             36439
Interest_rate_spread         36639
Upfront_charges              39642
term                            41
Neg_ammortization              121
interest_only                    0
lump_sum_payment                 0
property_value               15098
construction_type                0
occupancy_type                   0
Secured_by                       0
total_units                      0
income                        9150
credit_type                      0
Credit_Score                     0
co-applicant_credit_type         0
age                            200
submission_of_application      200
LTV                          15098
Region                           0
Security_Type                    0
Status                           0
dtir1                        24121
dtype: int64
```

In [7]:

```
#Check the percentage of missing values in each column:
missing_percentages = (df.isnull().sum() / len(df)) * 100
print(missing_percentages)
```

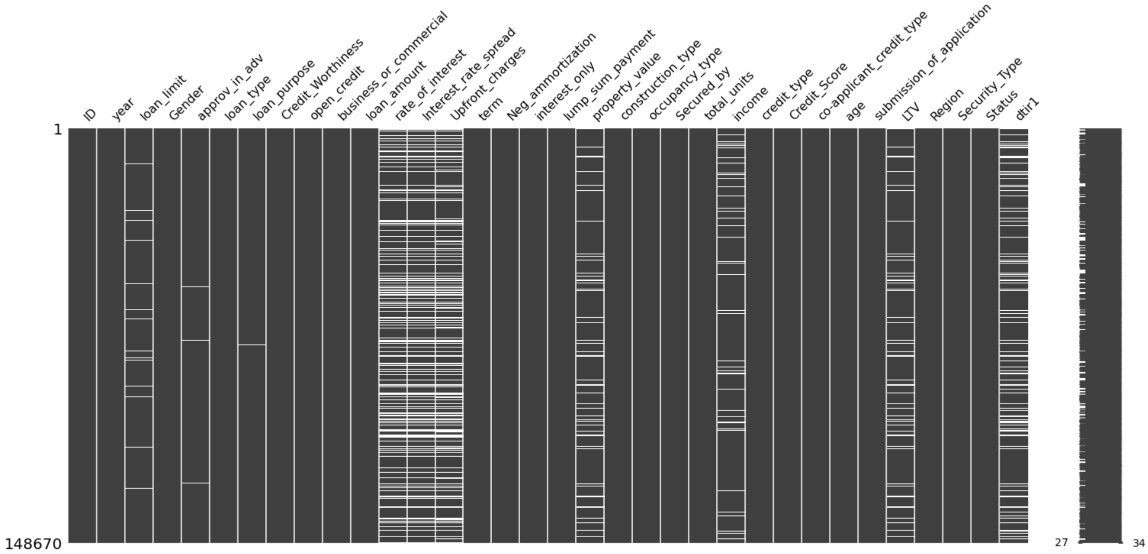
```
ID                            0.000000
year                          0.000000
loan_limit                    2.249277
Gender                        0.000000
approv_in_adv                 0.610749
loan_type                     0.000000
loan_purpose                  0.090133
Credit_Worthiness             0.000000
open_credit                   0.000000
business_or_commercial        0.000000
loan_amount                   0.000000
rate_of_interest             24.509989
Interest_rate_spread         24.644515
Upfront_charges              26.664425
term                          0.027578
Neg_ammortization             0.081388
interest_only                 0.000000
lump_sum_payment              0.000000
property_value               10.155378
construction_type             0.000000
occupancy_type                0.000000
Secured_by                    0.000000
total_units                   0.000000
income                        6.154571
credit_type                   0.000000
Credit_Score                  0.000000
co-applicant_credit_type      0.000000
age                           0.134526
submission_of_application     0.134526
LTV                          10.155378
Region                        0.000000
Security_Type                 0.000000
Status                        0.000000
dtir1                        16.224524
dtype: float64
```

In [8]:

```
#Analyze the patterns of missingness in the dataset:
import missingno as msno
msno.matrix(df)
```

Out[8]:

```
<AxesSubplot:>
```



The missing values are represented by white bars.

Preprocessing the missing Values

In [9]:

```
# Replace missing values with the mean and median in the numerical columns
df['loan_limit'] = pd.to_numeric(df['loan_limit'], errors='coerce') # convert loan_l
mean_loan_limit = df['loan_limit'].mean()
median_loan_limit = df['loan_limit'].median()
df['loan_limit'].fillna(mean_loan_limit, inplace=True) # replace missing values with
df['loan_limit'].fillna(median_loan_limit, inplace=True) # replace any remaining mis
df['rate_of_interest'].fillna(df['rate_of_interest'].mean(), inplace=True) # replace
df['Interest_rate_spread'].fillna(df['Interest_rate_spread'].median(), inplace=True)
df['Upfront_charges'].fillna(df['Upfront_charges'].median(), inplace=True) # replace
df['term'].fillna(df['term'].median(), inplace=True) # replace term with the median
df['property_value'].fillna(df['property_value'].median(), inplace=True) # replace p
df['income'].fillna(df['income'].mean(), inplace=True) # replace income with the mea
df['LTV'].fillna(df['LTV'].median(), inplace=True) # replace LTV with the median
df['dtir1'].fillna(df['dtir1'].mean(), inplace=True) # replace dtir1 with the mean

# Drop missing values in categorical columns
df.dropna(subset=['approv_in_adv', 'loan_purpose', 'Neg_ammortization', 'age', 'subm
```

In [10]:

```
# Verify the new dataframe
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 147315 entries, 0 to 148669
Data columns (total 34 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   ID                         147315 non-null  int64  
 1   year                       147315 non-null  int64  
 2   loan_limit                 0 non-null       float64
 3   Gender                     147315 non-null  object 
 4   approv_in_adv              147315 non-null  object 
 5   loan_type                  147315 non-null  object 
 6   loan_purpose               147315 non-null  object 
 7   Credit_Worthiness          147315 non-null  object 
 8   open_credit                147315 non-null  object 
 9   business_or_commercial     147315 non-null  object 
 10  loan_amount                147315 non-null  int64  
 11  rate_of_interest           147315 non-null  float64
 12  Interest_rate_spread       147315 non-null  float64
 13  Upfront_charges            147315 non-null  float64
 14  term                       147315 non-null  float64
 15  Neg_ammortization          147315 non-null  object 
 16  interest_only              147315 non-null  object 
 17  lump_sum_payment           147315 non-null  object 
 18  property_value             147315 non-null  float64
 19  construction_type          147315 non-null  object 
 20  occupancy_type             147315 non-null  object 
 21  Secured_by                 147315 non-null  object 
 22  total_units                147315 non-null  object 
 23  income                     147315 non-null  float64
 24  credit_type                147315 non-null  object 
 25  Credit_Score               147315 non-null  int64  
 26  co-applicant_credit_type   147315 non-null  object 
 27  age                        147315 non-null  object 
 28  submission_of_application  147315 non-null  object 
 29  LTV                        147315 non-null  float64
 30  Region                     147315 non-null  object 
 31  Security_Type              147315 non-null  object 
 32  Status                     147315 non-null  int64  
 33  dtir1                      147315 non-null  float64
dtypes: float64(9), int64(5), object(20)
memory usage: 39.3+ MB
None
```

In [11]:

```
print(df.isnull().sum())
```

```
ID                              0
year                            0
loan_limit                 147315
Gender                          0
approv_in_adv                   0
loan_type                       0
loan_purpose                    0
Credit_Worthiness               0
open_credit                     0
business_or_commercial          0
loan_amount                     0
rate_of_interest                0
Interest_rate_spread            0
Upfront_charges                 0
term                            0
Neg_ammortization               0
interest_only                   0
lump_sum_payment                0
property_value                  0
construction_type               0
occupancy_type                  0
Secured_by                      0
total_units                     0
income                          0
credit_type                     0
Credit_Score                    0
co-applicant_credit_type        0
age                             0
submission_of_application       0
LTV                             0
Region                          0
Security_Type                   0
Status                          0
dtir1                           0
dtype: int64
```

In [12]:

```
# Get the unique values in the 'loan_limit' column
unique_loan_limits = df['loan_limit'].unique()

print(unique_loan_limits)
```

```
[nan]
```

In [13]:

```
df = df.drop('loan_limit', axis=1)
```

In [14]:

```
# Check if there are any duplicate rows in the DataFrame
print(df.duplicated().sum())
```

```
0
```

Handling Outliers

In [15]:

```
# Define numerical and categorical variables
num_vars = df.select_dtypes(include=['float64', 'int64']).columns.tolist()
cat_vars = df.select_dtypes(include=['object']).columns.tolist()
```

In [16]:

```
print(cat_vars)

```

```
['Gender', 'approv_in_adv', 'loan_type', 'loan_purpose', 'Credit_Worthines
s', 'open_credit', 'business_or_commercial', 'Neg_ammortization', 'interes
t_only', 'lump_sum_payment', 'construction_type', 'occupancy_type', 'Secur
ed_by', 'total_units', 'credit_type', 'co-applicant_credit_type', 'age',
'submission_of_application', 'Region', 'Security_Type']
```

In [17]:

```
num_vars = df.select_dtypes(include=['float64', 'int64']).drop('Status', axis=1).col
```

In [18]:

```
# Handle outliers
def find_outliers_IQR(col):
    Q1 = col.quantile(0.25)
    Q3 = col.quantile(0.75)
    IQR = Q3 - Q1
    outliers = col[((col < (Q1 - 3*IQR)) | (col > (Q3 + 3*IQR)))]
    return outliers

#replacing outliers with median value
for col in num_vars:
    outliers = find_outliers_IQR(df[col])
    df.loc[outliers.index, col] = df[col].median()
```

Data Visualisation

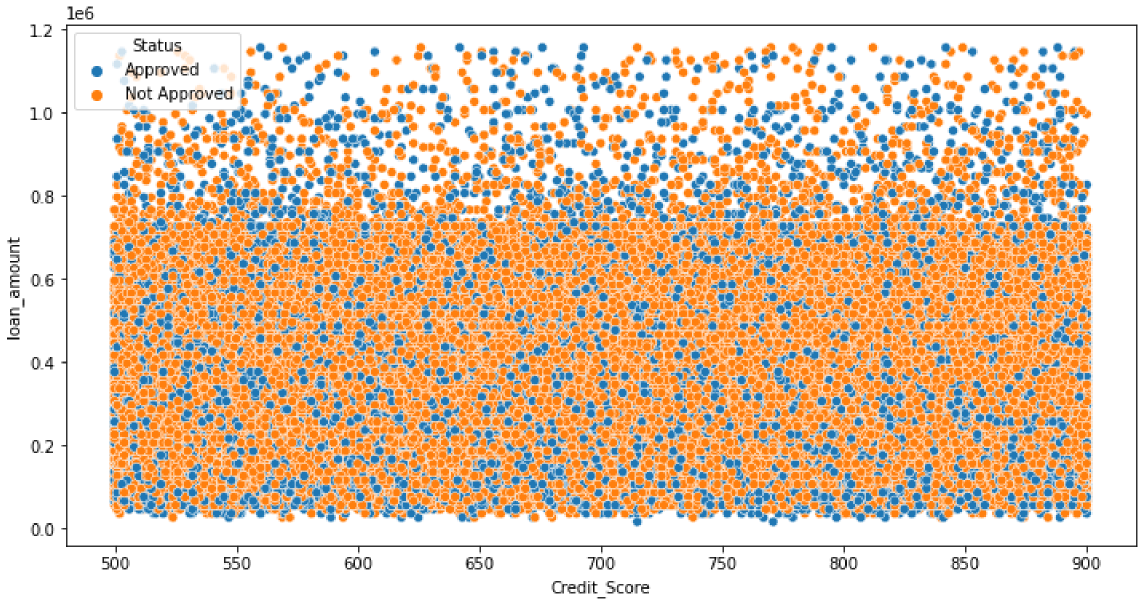
In [19]:

```
#convert 'Status' column to categorical type and replace 0 and 1 with 'Not Approved'
df['Status']=df['Status'].astype('category')
change={0:'Not Approved',1:'Approved'}
df['Status']=df['Status'].replace(change)
```

In [20]:

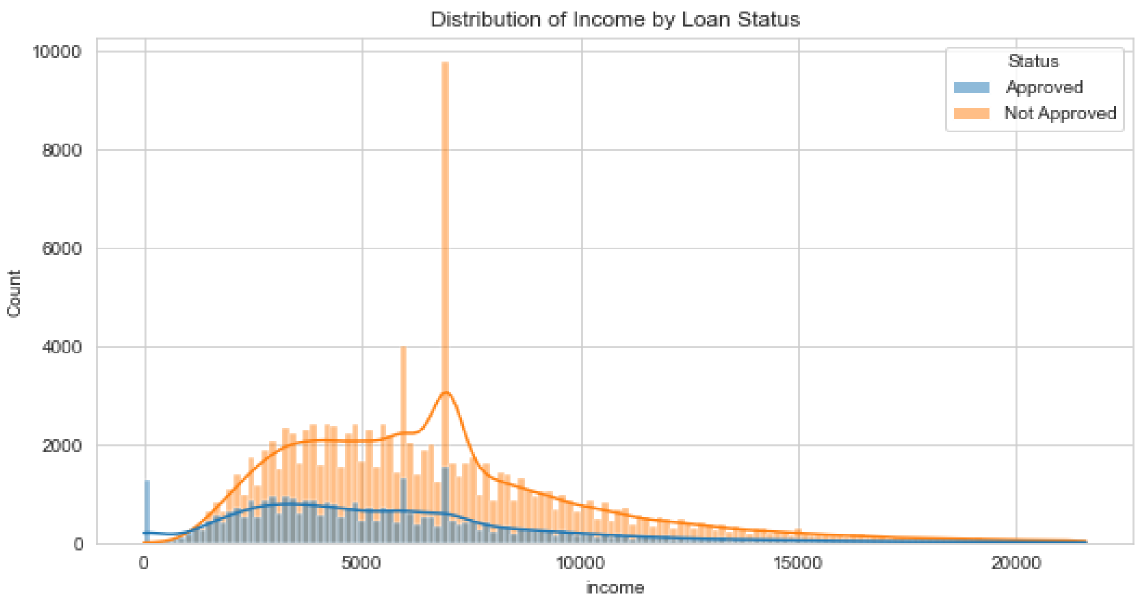
```
fig,ax=plt.subplots()
sns.scatterplot(x='Credit_Score',y='loan_amount',data=df,hue='Status')
fig.set_size_inches([12,6])
plt.show()
```

```
C:\Anaconda\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning:
Creating legend with loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



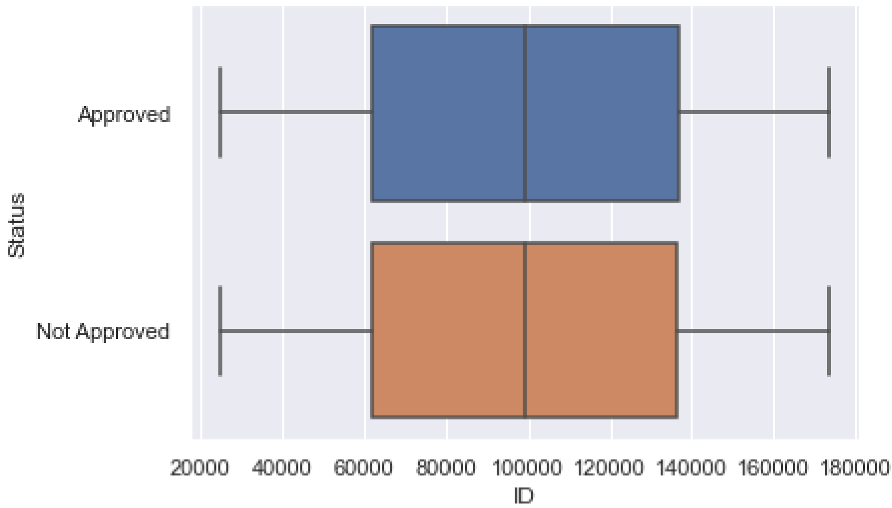
In [21]:

```
sns.set_style('whitegrid')

plt.figure(figsize=(10, 5))

sns.histplot(data=df, x='income', hue='Status', kde=True)

plt.title('Distribution of Income by Loan Status')
plt.show()

```



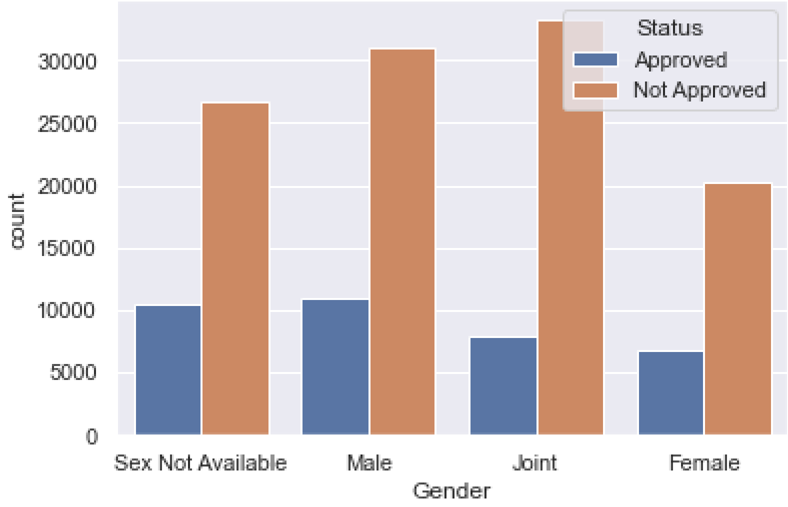
In [22]:

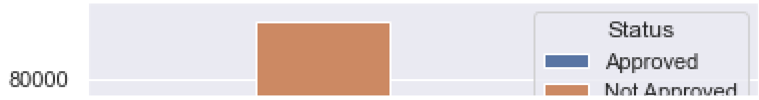
```
# Boxplots
for i in num_vars:
    plt.figsize=(16,6)
    sns.set_theme(style='darkgrid')
    sns.boxplot(x=i, y='Status', data=df)
    plt.show()

# Histograms
for i in num_vars:
    plt.figsize=(16,6)
    sns.set_theme(style='darkgrid')
    sns.histplot(data=df, x=i, hue="Status", multiple="dodge", shrink=.8, bins=4)
    plt.show()

```



In [23]:

```
#Categorical Variable Countplots by Status
for var in cat_vars:
    sns.countplot(data=df, x=var, hue='Status')
    plt.show()
```

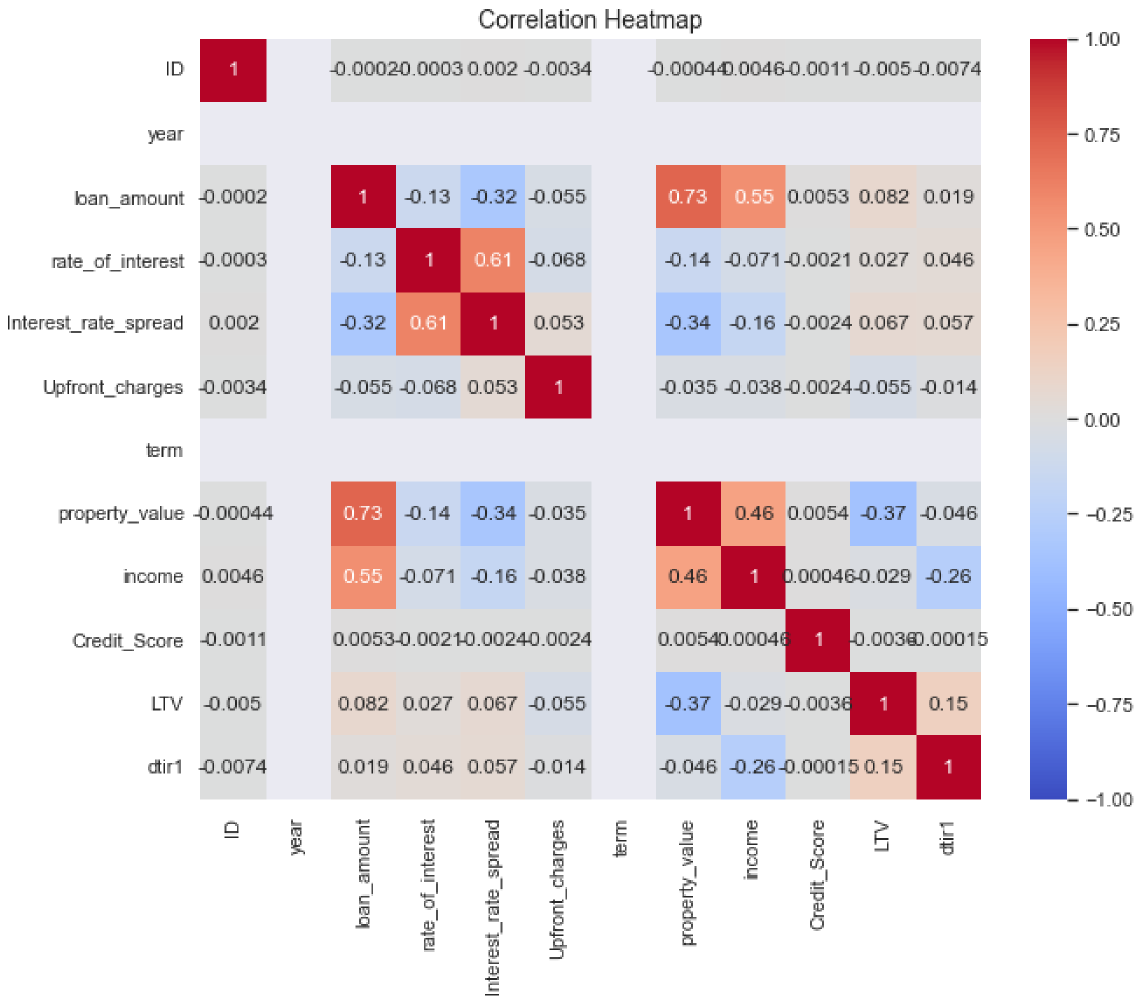




Handling Multicolinearity

In [24]:

```
# Define numerical variables
num_vars = df.select_dtypes(include=['float64', 'int64']).columns.tolist()

# Create a correlation matrix
corr = df[num_vars + ['Status']].corr()

# Plot a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap', fontsize=14)
plt.show()

```



In [25]:

```
# Select the numerical columns from df
num_df = df[num_vars]

# Calculate the correlation matrix
corr_matrix = num_df.corr()

# Print the correlation matrix
print(corr_matrix)

# Find pairs of variables with high correlation coefficients
high_corr = []
for i in range(len(corr_matrix.columns)):
    for j in range(i+1, len(corr_matrix.columns)):
        if abs(corr_matrix.iloc[i, j]) > 0.7:
            high_corr.append((corr_matrix.columns[i], corr_matrix.columns[j], corr_m

# Print the pairs of variables with high correlation coefficients
print(high_corr)
```

```
                           ID  year  loan_amount  rate_of_interest  \
ID                   1.000000   NaN    -0.000196         -0.000295   
year                      NaN   NaN          NaN               NaN   
loan_amount         -0.000196   NaN     1.000000         -0.128386   
rate_of_interest    -0.000295   NaN    -0.128386          1.000000   
Interest_rate_spread  0.002021   NaN    -0.322804          0.606784   
Upfront_charges     -0.003413   NaN    -0.054611         -0.068133   
term                      NaN   NaN          NaN               NaN   
property_value      -0.000437   NaN     0.729564         -0.141982   
income               0.004650   NaN     0.547796         -0.070687   
Credit_Score        -0.001112   NaN     0.005340         -0.002146   
LTV                 -0.005020   NaN     0.081831          0.027318   
dtir1               -0.007411   NaN     0.018722          0.045542   

                      Interest_rate_spread  Upfront_charges  term  \
ID                                0.002021        -0.003413   NaN   
year                                   NaN              NaN   NaN   
loan_amount                      -0.322804        -0.054611   NaN   
rate_of_interest                  0.606784        -0.068133   NaN   
Interest_rate_spread              1.000000         0.053245   NaN   
Upfront_charges                   0.053245         1.000000   NaN   
term                                   NaN              NaN   NaN   
property_value                   -0.341126        -0.034610   NaN   
income                           -0.164849        -0.037989   NaN   
Credit_Score                     -0.002413        -0.002385   NaN   
LTV                               0.066583        -0.055004   NaN   
dtir1                             0.057295        -0.013920   NaN   

                      property_value    income  Credit_Score       LTV  \
ID                         -0.000437  0.004650     -0.001112 -0.005020   
year                             NaN       NaN           NaN       NaN   
loan_amount                 0.729564  0.547796      0.005340  0.081831   
rate_of_interest           -0.141982 -0.070687     -0.002146  0.027318   
Interest_rate_spread       -0.341126 -0.164849     -0.002413  0.066583   
Upfront_charges            -0.034610 -0.037989     -0.002385 -0.055004   
term                             NaN       NaN           NaN       NaN   
property_value              1.000000  0.460629      0.005377 -0.369274   
income                      0.460629  1.000000      0.000456 -0.029024   
Credit_Score                0.005377  0.000456      1.000000 -0.003629   
LTV                        -0.369274 -0.029024     -0.003629  1.000000   
dtir1                      -0.046249 -0.260424     -0.000150  0.151461   

                         dtir1  
ID                   -0.007411  
year                       NaN  
loan_amount           0.018722  
rate_of_interest      0.045542  
Interest_rate_spread  0.057295  
Upfront_charges      -0.013920  
term                       NaN  
property_value       -0.046249  
income               -0.260424  
Credit_Score         -0.000150  
LTV                   0.151461  
dtir1                 1.000000  
[('loan_amount', 'property_value', 0.7295637877973961)]
```

In [26]:

```
# Display encoded data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 147315 entries, 0 to 148669
Data columns (total 33 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   ID                         147315 non-null  int64  
 1   year                       147315 non-null  int64  
 2   Gender                     147315 non-null  object 
 3   approv_in_adv              147315 non-null  object 
 4   loan_type                  147315 non-null  object 
 5   loan_purpose               147315 non-null  object 
 6   Credit_Worthiness          147315 non-null  object 
 7   open_credit                147315 non-null  object 
 8   business_or_commercial     147315 non-null  object 
 9   loan_amount                147315 non-null  int64  
 10  rate_of_interest           147315 non-null  float64
 11  Interest_rate_spread       147315 non-null  float64
 12  Upfront_charges            147315 non-null  float64
 13  term                       147315 non-null  float64
 14  Neg_ammortization          147315 non-null  object 
 15  interest_only              147315 non-null  object 
 16  lump_sum_payment           147315 non-null  object 
 17  property_value             147315 non-null  float64
 18  construction_type          147315 non-null  object 
 19  occupancy_type             147315 non-null  object 
 20  Secured_by                 147315 non-null  object 
 21  total_units                147315 non-null  object 
 22  income                     147315 non-null  float64
 23  credit_type                147315 non-null  object 
 24  Credit_Score               147315 non-null  int64  
 25  co-applicant_credit_type   147315 non-null  object 
 26  age                        147315 non-null  object 
 27  submission_of_application  147315 non-null  object 
 28  LTV                        147315 non-null  float64
 29  Region                     147315 non-null  object 
 30  Security_Type              147315 non-null  object 
 31  Status                     147315 non-null  object 
 32  dtir1                      147315 non-null  float64
dtypes: float64(8), int64(4), object(21)
memory usage: 42.2+ MB
```

In [27]:

```
df.columns
```

Out[27]:

```
Index(['ID', 'year', 'Gender', 'approv_in_adv', 'loan_type', 'loan_purpos
e',
       'Credit_Worthiness', 'open_credit', 'business_or_commercial',
       'loan_amount', 'rate_of_interest', 'Interest_rate_spread',
       'Upfront_charges', 'term', 'Neg_ammortization', 'interest_only',
       'lump_sum_payment', 'property_value', 'construction_type',
       'occupancy_type', 'Secured_by', 'total_units', 'income', 'credit_ty
pe',
       'Credit_Score', 'co-applicant_credit_type', 'age',
       'submission_of_application', 'LTV', 'Region', 'Security_Type', 'Sta
tus',
       'dtir1'],
      dtype='object')
```

In [28]:

```
# Encode categorical variables
encoder = LabelEncoder()
cat_vars_encoded = pd.DataFrame()
for col in cat_vars:
    cat_vars_encoded[col] = encoder.fit_transform(df[col])

# Concatenate numerical, categorical, and 'Status' columns
num_cat_vars = num_vars + cat_vars_encoded.columns.tolist() + ['Status']
df_encoded = pd.concat([df[num_vars], pd.get_dummies(df[cat_vars], drop_first=True),
```

In [29]:

```
df_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 147315 entries, 0 to 148669
Data columns (total 50 columns):
 #   Column                             Non-Null Count   Dtype  
---  ------                             --------------   -----  
 0   ID                                 147315 non-null  int64  
 1   year                               147315 non-null  int64  
 2   loan_amount                        147315 non-null  int64  
 3   rate_of_interest                   147315 non-null  float64
 4   Interest_rate_spread               147315 non-null  float64
 5   Upfront_charges                    147315 non-null  float64
 6   term                               147315 non-null  float64
 7   property_value                     147315 non-null  float64
 8   income                             147315 non-null  float64
 9   Credit_Score                       147315 non-null  int64  
 10  LTV                                147315 non-null  float64
 11  dtir1                              147315 non-null  float64
 12  Gender_Joint                       147315 non-null  uint8  
 13  Gender_Male                        147315 non-null  uint8  
 14  Gender_Sex Not Available           147315 non-null  uint8  
 15  approv_in_adv_pre                  147315 non-null  uint8  
 16  loan_type_type2                    147315 non-null  uint8  
 17  loan_type_type3                    147315 non-null  uint8  
 18  loan_purpose_p2                    147315 non-null  uint8  
 19  loan_purpose_p3                    147315 non-null  uint8  
 20  loan_purpose_p4                    147315 non-null  uint8  
 21  Credit_Worthiness_l2               147315 non-null  uint8  
 22  open_credit_opc                    147315 non-null  uint8  
 23  business_or_commercial_nob/c       147315 non-null  uint8  
 24  Neg_ammortization_not_neg          147315 non-null  uint8  
 25  interest_only_not_int              147315 non-null  uint8  
 26  lump_sum_payment_not_lpsm          147315 non-null  uint8  
 27  construction_type_sb               147315 non-null  uint8  
 28  occupancy_type_pr                  147315 non-null  uint8  
 29  occupancy_type_sr                  147315 non-null  uint8  
 30  Secured_by_land                    147315 non-null  uint8  
 31  total_units_2U                     147315 non-null  uint8  
 32  total_units_3U                     147315 non-null  uint8  
 33  total_units_4U                     147315 non-null  uint8  
 34  credit_type_CRIF                   147315 non-null  uint8  
 35  credit_type_EQUI                   147315 non-null  uint8  
 36  credit_type_EXP                    147315 non-null  uint8  
 37  co-applicant_credit_type_EXP       147315 non-null  uint8  
 38  age_35-44                          147315 non-null  uint8  
 39  age_45-54                          147315 non-null  uint8  
 40  age_55-64                          147315 non-null  uint8  
 41  age_65-74                          147315 non-null  uint8  
 42  age_<25                            147315 non-null  uint8  
 43  age_>74                            147315 non-null  uint8  
 44  submission_of_application_to_inst  147315 non-null  uint8  
 45  Region_North-East                  147315 non-null  uint8  
 46  Region_central                     147315 non-null  uint8  
 47  Region_south                       147315 non-null  uint8  
 48  Security_Type_direct               147315 non-null  uint8  
 49  Status                             147315 non-null  object 
dtypes: float64(8), int64(4), object(1), uint8(37)
memory usage: 25.0+ MB
```

In [30]:

```
# drop the ID and property_value columns
df_encoded = df_encoded.drop(['ID', 'property_value'], axis=1)

```

Data Training

In [31]:

```
# split the data into training and testing sets

X = df_encoded.drop("Status", axis=1)
y = df_encoded["Status"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

```

------------Model Training and Evaluation-------------

Logistic Regression

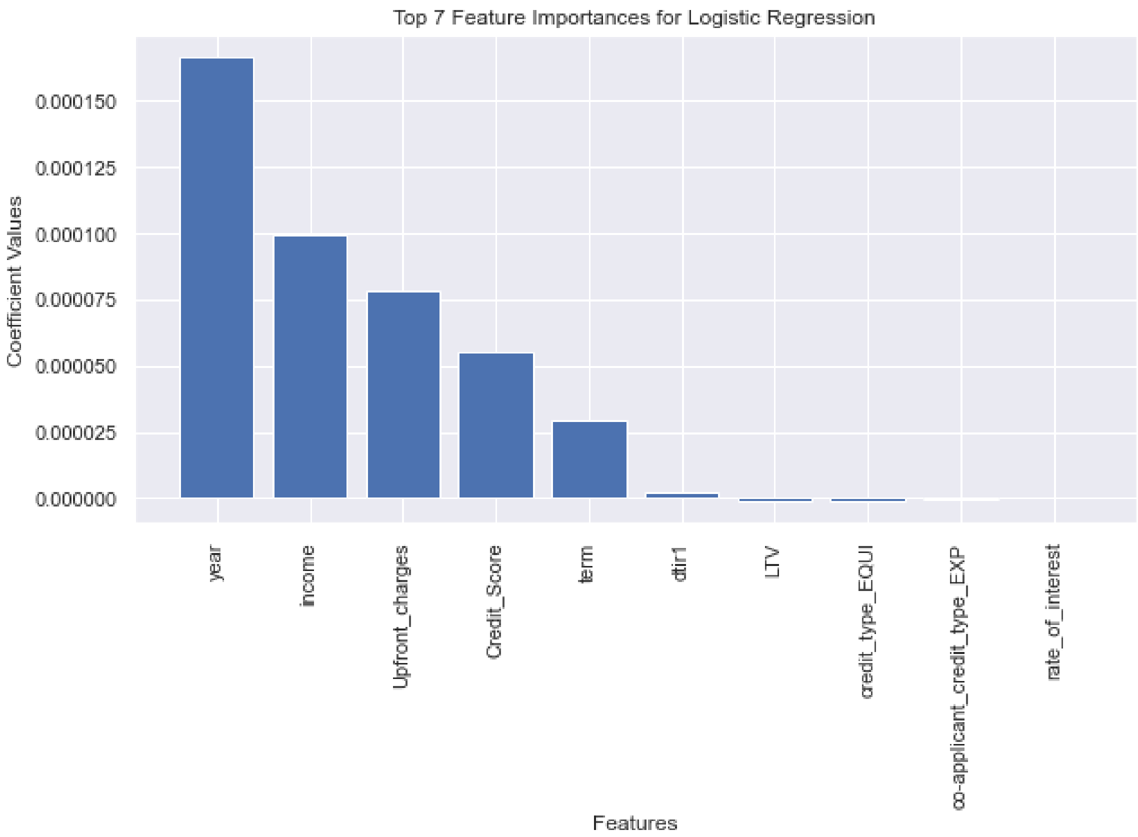
In [32]:

```
# Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
print('Logistic Regression:')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
print('ROC AUC Score:')
y_prob = logreg.predict_proba(X_test)[:, 1]
print(roc_auc_score(y_test, y_prob))


```

```
Logistic Regression:
Accuracy: 0.7542341241557208
Classification Report:

C:\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1334: Und
efinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1334: Und
efinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1334: Und
efinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

    Approved       0.00      0.00      0.00      7241
Not Approved       0.75      1.00      0.86     22222

    accuracy                           0.75     29463
   macro avg       0.38      0.50      0.43     29463
weighted avg       0.57      0.75      0.65     29463

Confusion Matrix:
[[    0  7241]
 [    0 22222]]
ROC AUC Score:
0.6087757266193019
```

In [41]:

```
# Get feature importances
coef_abs = np.abs(logreg.coef_)
indices = np.argsort(coef_abs)[0][::-1][:10]
features = X_train.columns
importances = [logreg.coef_[0][i] for i in indices]

# Create column chart
plt.figure(figsize=(10, 5))
plt.title('Top 7 Feature Importances for Logistic Regression')
plt.bar(features[indices], importances)
plt.xticks(rotation=90)
plt.xlabel('Features')
plt.ylabel('Coefficient Values')
plt.show()

```



Decision Tree

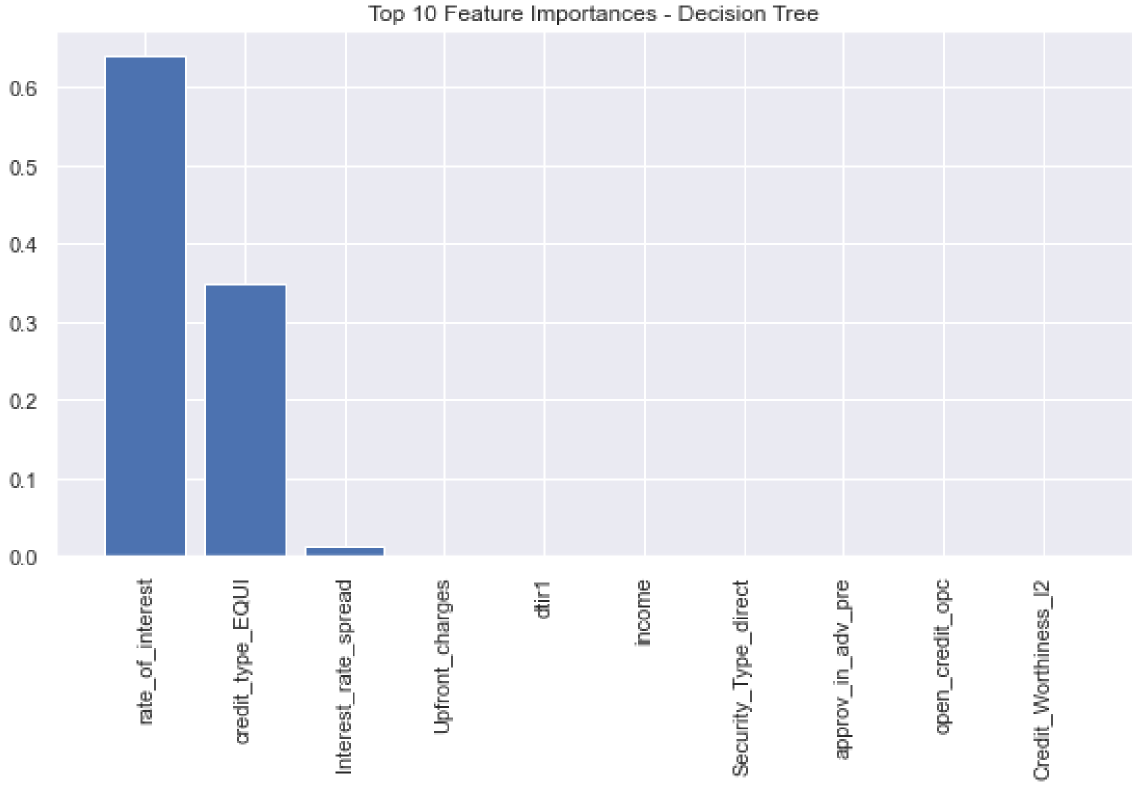
In [50]:

```
from sklearn.tree import DecisionTreeClassifier

# Create decision tree classifier
dt = DecisionTreeClassifier()

# Fit the model on training data
dt.fit(X_train, y_train)

# Make predictions on test data
y_pred_dt = dt.predict(X_test)

print('Decision Tree:')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
print('ROC AUC Score:')
y_prob = dt.predict_proba(X_test)[:, 1]
print(roc_auc_score(y_test, y_prob))

```

```
Decision Tree:
Accuracy: 0.9999321182500085
Classification Report:
              precision    recall  f1-score   support

    Approved       1.00      1.00      1.00      7241
Not Approved       1.00      1.00      1.00     22222

    accuracy                           1.00     29463
   macro avg       1.00      1.00      1.00     29463
weighted avg       1.00      1.00      1.00     29463

Confusion Matrix:
[[ 7241     0]
 [    2 22220]]
ROC AUC Score:
0.9999324993249932
```

In [43]:

```
# Plot feature importances
importances = dt.feature_importances_
indices = np.argsort(importances)[::-1]
features = X_train.columns

plt.figure(figsize=(10, 5))
plt.title("Top 10 Feature Importances - Decision Tree")
plt.bar(range(10), importances[indices][:10])
plt.xticks(range(10), features[indices][:10], rotation=90)
plt.show()

```



In [ ]:

```

```

Random Forest

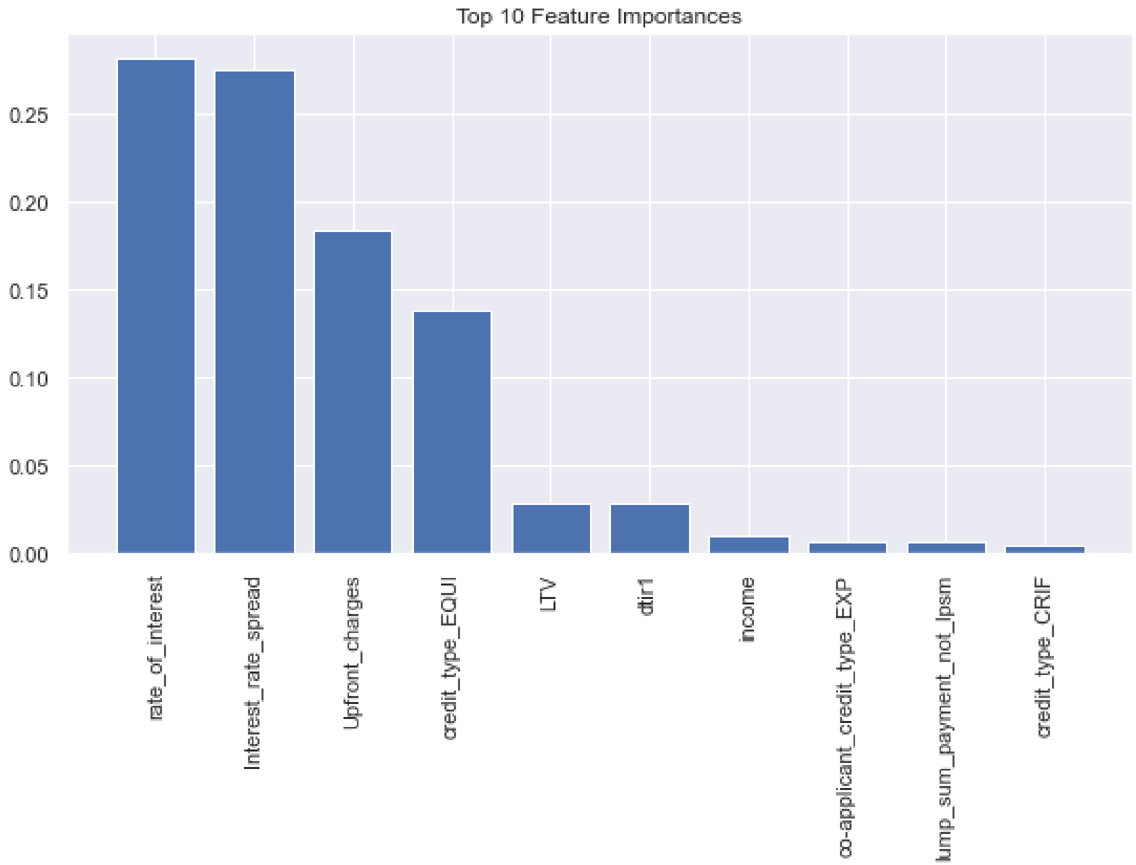
In [52]:

```
# Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print('Random Forest:')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
print('ROC AUC Score:')
y_prob = rf.predict_proba(X_test)[:, 1]
print(roc_auc_score(y_test, y_prob))


```

```
Random Forest:
Accuracy: 0.9999321182500085
Classification Report:
              precision    recall  f1-score   support

    Approved       1.00      1.00      1.00      7241
Not Approved       1.00      1.00      1.00     22222

    accuracy                           1.00     29463
   macro avg       1.00      1.00      1.00     29463
weighted avg       1.00      1.00      1.00     29463

Confusion Matrix:
[[ 7241     0]
 [    2 22220]]
ROC AUC Score:
0.9999700794549722
```

In [45]:

```
# Plot top 10 feature importances
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
features = X_train.columns

top_n = 10  # Change this value to show more or fewer features
plt.figure(figsize=(10, 5))
plt.title(f"Top {top_n} Feature Importances")
plt.bar(range(top_n), importances[indices][:top_n])
plt.xticks(range(top_n), features[indices][:top_n], rotation=90)
plt.show()

```



----------- CROSS VALIDATE THE MODELS ---------

In [46]:

```

# create instances of the models
lr = LogisticRegression()
dt = DecisionTreeClassifier()
rf = RandomForestClassifier()

# define the evaluation metric and number of folds
metric = 'accuracy'
n_folds = 5
```

In [47]:

```

# perform cross validation and get the scores
lr_scores = cross_val_score(lr, X, y, cv=n_folds, scoring=metric)
dt_scores = cross_val_score(dt, X, y, cv=n_folds, scoring=metric)
rf_scores = cross_val_score(rf, X, y, cv=n_folds, scoring=metric)

# print the mean and standard deviation of the scores
print("Logistic Regression Accuracy: {:.2f} (+/- {:.2f})".format(np.mean(lr_scores),
print("Decision Tree Accuracy: {:.2f} (+/- {:.2f})".format(np.mean(dt_scores), np.st
print("Random Forest Accuracy: {:.2f} (+/- {:.2f})".format(np.mean(rf_scores), np.st

```

```
C:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:444: Conve
rgenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
C:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:444: Conve
rgenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
C:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:444: Conve
rgenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(

Logistic Regression Accuracy: 0.75 (+/- 0.00)
Decision Tree Accuracy: 1.00 (+/- 0.00)
Random Forest Accuracy: 1.00 (+/- 0.00)
```

In [ ]:

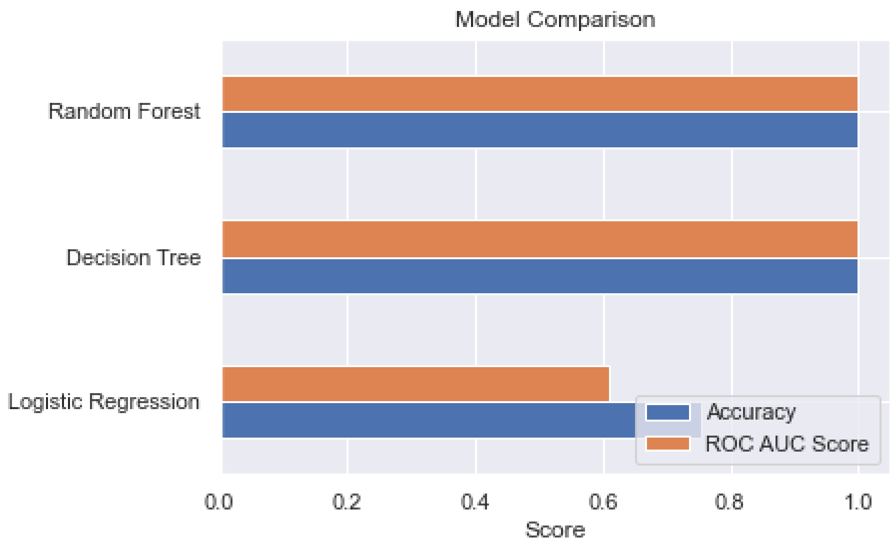
```

```

-------------MODEL RESULT COMPARISON ------------

In [53]:

```

# Create a dictionary to store the model names and their corresponding evaluation me
models = {
    'Logistic Regression': [accuracy_score(y_test, logreg.predict(X_test)),
                            roc_auc_score(y_test, logreg.predict_proba(X_test)[:, 1]
    'Decision Tree': [accuracy_score(y_test, dt.predict(X_test)),
                      roc_auc_score(y_test, dt.predict_proba(X_test)[:, 1])],
    'Random Forest': [accuracy_score(y_test, rf.predict(X_test)),
                      roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])]

}

# Create a pandas DataFrame from the models dictionary
df = pd.DataFrame(models, index=['Accuracy', 'ROC AUC Score'])

# Print the DataFrame
print(df)

# Create a horizontal bar chart
df.T.plot(kind='barh')
plt.title('Model Comparison')
plt.xlabel('Score')
plt.legend(loc='best')
plt.show()

```

	Logistic Regression	Decision Tree	Random Forest
Accuracy	0.754234	0.999898	0.999932
ROC AUC Score	0.608776	0.999932	0.999970



In [54]:

```python
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

models = ['Logistic Regression', 'Decision Tree', 'Random Forest']

approve_precision_scores = [precision_score(y_test, model.predict(X_test), pos_label
approve_recall_scores = [recall_score(y_test, model.predict(X_test), pos_label='Appr
not_approve_precision_scores = [precision_score(y_test, model.predict(X_test), pos_l
not_approve_recall_scores = [recall_score(y_test, model.predict(X_test), pos_label='
accuracy_scores = [accuracy_score(y_test, model.predict(X_test)) for model in [logre
f1_scores = [f1_score(y_test, model.predict(X_test), pos_label='Approved') for model

# Create a dataframe
data = {'Model': models,
        'Precision - Approved': approve_precision_scores,
        'Recall - Approved': approve_recall_scores,
        'Precision - Not Approved': not_approve_precision_scores,
        'Recall - Not Approved': not_approve_recall_scores,
        'Accuracy': accuracy_scores,
        'F1 Score - Approved': f1_scores}
df = pd.DataFrame(data)

# Set the index to the model names
df.set_index('Model', inplace=True)

# Print the dataframe
print(df)
```

```
C:\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1334: Und
efinedMetricWarning: Precision is ill-defined and being set to 0.0 due to
no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))

                     Precision - Approved  Recall - Approved  \
Model
Logistic Regression              0.000000                0.0
Decision Tree                    0.999586                1.0
Random Forest                    0.999724                1.0

                     Precision - Not Approved  Recall - Not Approved  \
Model
Logistic Regression                  0.754234               1.000000
Decision Tree                        1.000000               0.999865
Random Forest                        1.000000               0.999910

                     Accuracy  F1 Score - Approved
Model
Logistic Regression  0.754234             0.000000
Decision Tree        0.999898             0.999793
Random Forest        0.999932             0.999862
```

The table shows the evaluation metrics for three different models: Logistic Regression, Decision Tree, and Random Forest.

For Logistic Regression, it has a Precision - Approved score of 0.0 which means that it did not predict any approved loans correctly. The Recall - Approved score is also 0.0 which means that it missed all of the approved loans. On the other hand, it has a high Precision - Not Approved score of 0.754 which means that when it predicts a loan as not approved, it is usually correct. The Recall - Not Approved score is 1.0 which means that it correctly identified all of the not approved loans. The Accuracy score is 0.754 which means that overall it correctly classified only 75.4% of the loans. The F1 Score - Approved is 0.0 because there were no true positive predictions for approved loans.

For Decision Tree, it has a high Precision - Approved score of 0.999 which means that when it predicts a loan as approved, it is usually correct. The Recall - Approved score is 1.0 which means that it correctly identified all of the approved loans. It also has a high Precision - Not Approved score of 1.0 which means that when it predicts a loan as not approved, it is usually correct. The Recall - Not Approved score is 0.999 which means that it missed only 0.1% of the not approved loans. The Accuracy score is 0.999 which means that overall it correctly classified 99.9% of the loans. The F1 Score - Approved is 0.999 because it has high precision and recall for approved loans.

For Random Forest, it has a high Precision - Approved score of 0.999 which means that when it predicts a loan as approved, it is usually correct. The Recall - Approved score is 1.0 which means that it correctly identified all of the approved loans. It also has a high Precision - Not Approved score of 1.0 which means that when it predicts a loan as not approved, it is usually correct. The Recall - Not Approved score is 0.999 which means that it missed only 0.09% of the not approved loans. The Accuracy score is 0.999 which means that overall it correctly classified 99.9% of the loans. The F1 Score - Approved is 0.999 because it has high precision and recall for approved loans.

In summary, based on these evaluation metrics, the Decision Tree and Random Forest models performed

In [ ]:

```
 
```

In [ ]:

```
 
```