Michelle Helfman - Identifying Credit Risk.

Milestone 5 - All code used in the Term Project.

Identifying customers at-risk of defaulting on loans before approving the credit application.

Milestone 3

Data Exploration and Visualizations

In [1]:
```
# Import Functions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.figure_factory as ff
import plotly.express as px

from sklearn import tree
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from yellowbrick.classifier import ROCAUC

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Set your custom color palette

colors = ["#FF0B04", "#4374B3", "#FF7733", "#782F98", "#00FF00","#FFFF4D","#D2691E"]
cust_colors = sns.set_palette(sns.color_palette(colors))
red = "#FF0B04"
blue = "#4374B3"
purple = "#782F98"
```

In [3]:
```
# Create data frames for all records
risk_df = pd.read_csv('credit_risk_dataset.csv')

print('1st 10 Rows of Dataset')
risk_df.head(10)
```

```
1st 10 Rows of Dataset
```

Out[3]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.02	1
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	0
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	1
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.23	1
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.27	1
5	21	9900	OWN	2.0	VENTURE	A	2500	7.14	1
6	26	77100	RENT	8.0	EDUCATION	B	35000	12.42	1
7	24	78956	RENT	5.0	MEDICAL	B	35000	11.11	1
8	24	83000	RENT	8.0	PERSONAL	A	35000	8.90	1
9	21	10000	OWN	6.0	VENTURE	D	1600	14.74	1

In [4]:
```python
# Rename Headers

new_col_headers = {'person_age': 'Age',
'person_income': 'Income',
'person_home_ownership': 'Home Ownership',
'person_emp_length': 'Employment Length',
'loan_amnt': 'Loan Amount',
'loan_int_rate': 'Interest Rate',
'loan_intent': 'Intent',
'loan_status': 'Current Loan Status',
'loan_grade': 'Loan Grade',
'loan_percent_income': 'Loan Percent Income',
'cb_person_default_on_file': 'Prior Defaults',
'cb_person_cred_hist_length': 'Credit History Length'}

risk_df.rename(columns = new_col_headers, inplace = True)
print('1st 10 Rows of Dataset')
risk_df.head(10)
```

```
1st 10 Rows of Dataset
```

Out[4]:

	Age	Income	Home Ownership	Employment Length	Intent	Loan Grade	Loan Amount	Interest Rate	Current Loan Status	Loan Percent Income	Prior Defaults	Credit History Length
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.02	1	0.59	Y	3
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	0	0.10	N	2
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	1	0.57	N	3
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.23	1	0.53	N	2
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.27	1	0.55	Y	4
5	21	9900	OWN	2.0	VENTURE	A	2500	7.14	1	0.25	N	2
6	26	77100	RENT	8.0	EDUCATION	B	35000	12.42	1	0.45	N	3
7	24	78956	RENT	5.0	MEDICAL	B	35000	11.11	1	0.44	N	4
8	24	83000	RENT	8.0	PERSONAL	A	35000	8.90	1	0.42	N	2
9	21	10000	OWN	6.0	VENTURE	D	1600	14.74	1	0.16	N	3

In [5]:
```python
# Describe the data

risk_df.describe().apply(lambda s: s.apply('{0:.5f}'.format))
```

Out[5]:

	Age	Income	Employment Length	Loan Amount	Interest Rate	Current Loan Status	Loan Percent Income	Credit History Length
count	32581.00000	32581.00000	31686.00000	32581.00000	29465.00000	32581.00000	32581.00000	32581.00000
mean	27.73460	66074.84847	4.78969	9589.37111	11.01169	0.21816	0.17020	5.80421
std	6.34808	61983.11917	4.14263	6322.08665	3.24046	0.41301	0.10678	4.05500
min	20.00000	4000.00000	0.00000	500.00000	5.42000	0.00000	0.00000	2.00000
25%	23.00000	38500.00000	2.00000	5000.00000	7.90000	0.00000	0.09000	3.00000
50%	26.00000	55000.00000	4.00000	8000.00000	10.99000	0.00000	0.15000	4.00000
75%	30.00000	79200.00000	7.00000	12200.00000	13.47000	0.00000	0.23000	8.00000
max	144.00000	6000000.00000	123.00000	35000.00000	23.22000	1.00000	0.83000	30.00000

In [6]:
```python
# Column datatypes and non-null counts

print('Credit Risk Column Information')

risk_df.info()
```

```
Credit Risk Column Information
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    32581 non-null  int64
 1   Income                 32581 non-null  int64
 2   Home Ownership         32581 non-null  object
 3   Employment Length      31686 non-null  float64
 4   Intent                 32581 non-null  object
 5   Loan Grade             32581 non-null  object
 6   Loan Amount            32581 non-null  int64
 7   Interest Rate          29465 non-null  float64
 8   Current Loan Status    32581 non-null  int64
 9   Loan Percent Income    32581 non-null  float64
 10  Prior Defaults         32581 non-null  object
 11  Credit History Length  32581 non-null  int64
dtypes: float64(3), int64(5), object(4)
memory usage: 3.0+ MB
```

In [7]:
```python
# Number of rows and columns

risk_df.shape
```

Out[7]:
```
(32581, 12)
```

In [8]:
```python
# The percentages of column nulls

null_counts = risk_df.isna().sum()
total = len(risk_df)

print('Percentage Columns with NULLS')
pct_null = round((null_counts / total) * 100,1)
pct_null
```

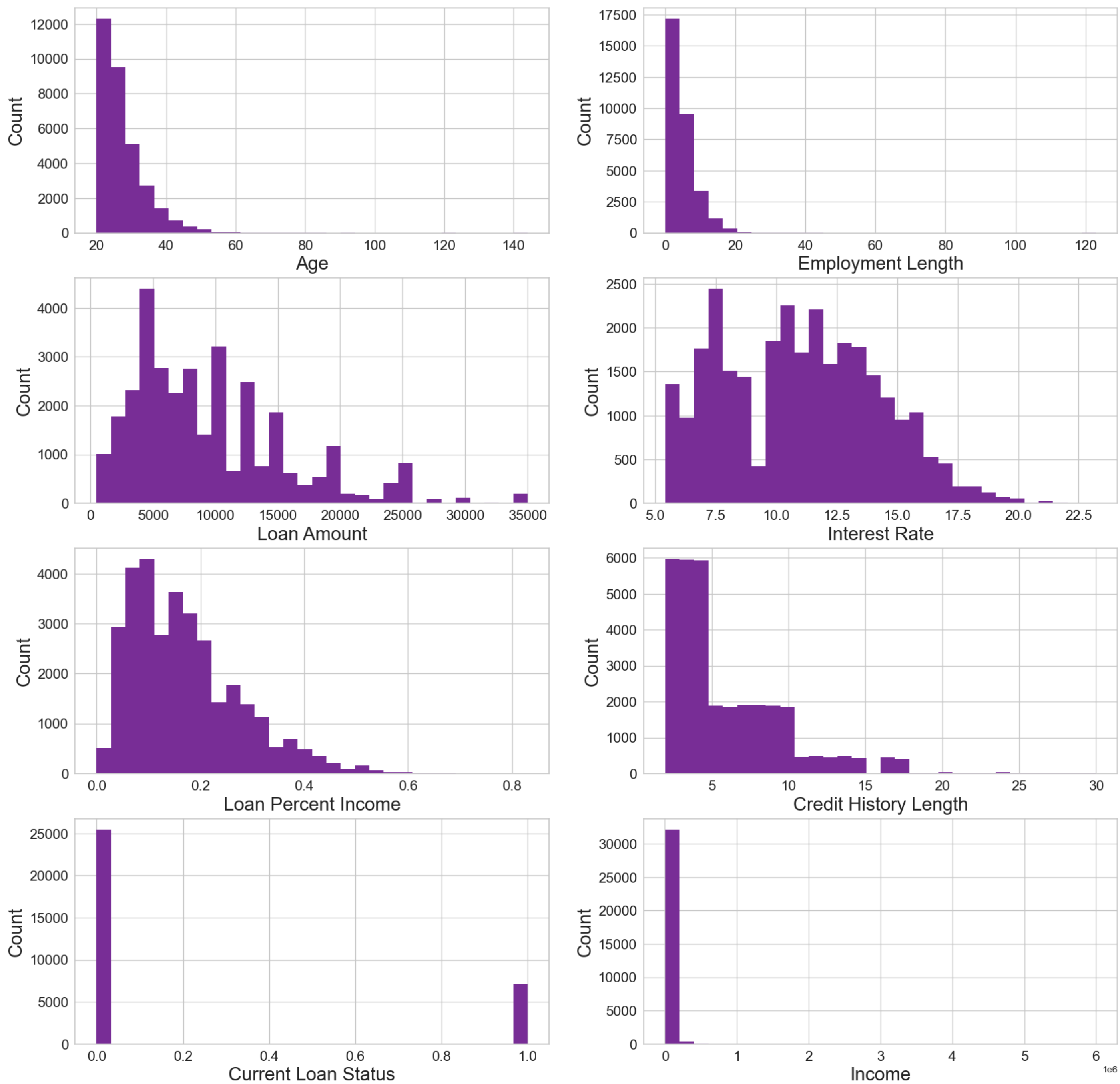
```
Percentage Columns with NULLS
```

Out[8]:
```
Age                      0.0
Income                   0.0
Home Ownership           0.0
Employment Length        2.7
Intent                   0.0
Loan Grade               0.0
Loan Amount              0.0
Interest Rate            9.6
Current Loan Status      0.0
Loan Percent Income      0.0
Prior Defaults           0.0
Credit History Length    0.0
dtype: float64
```

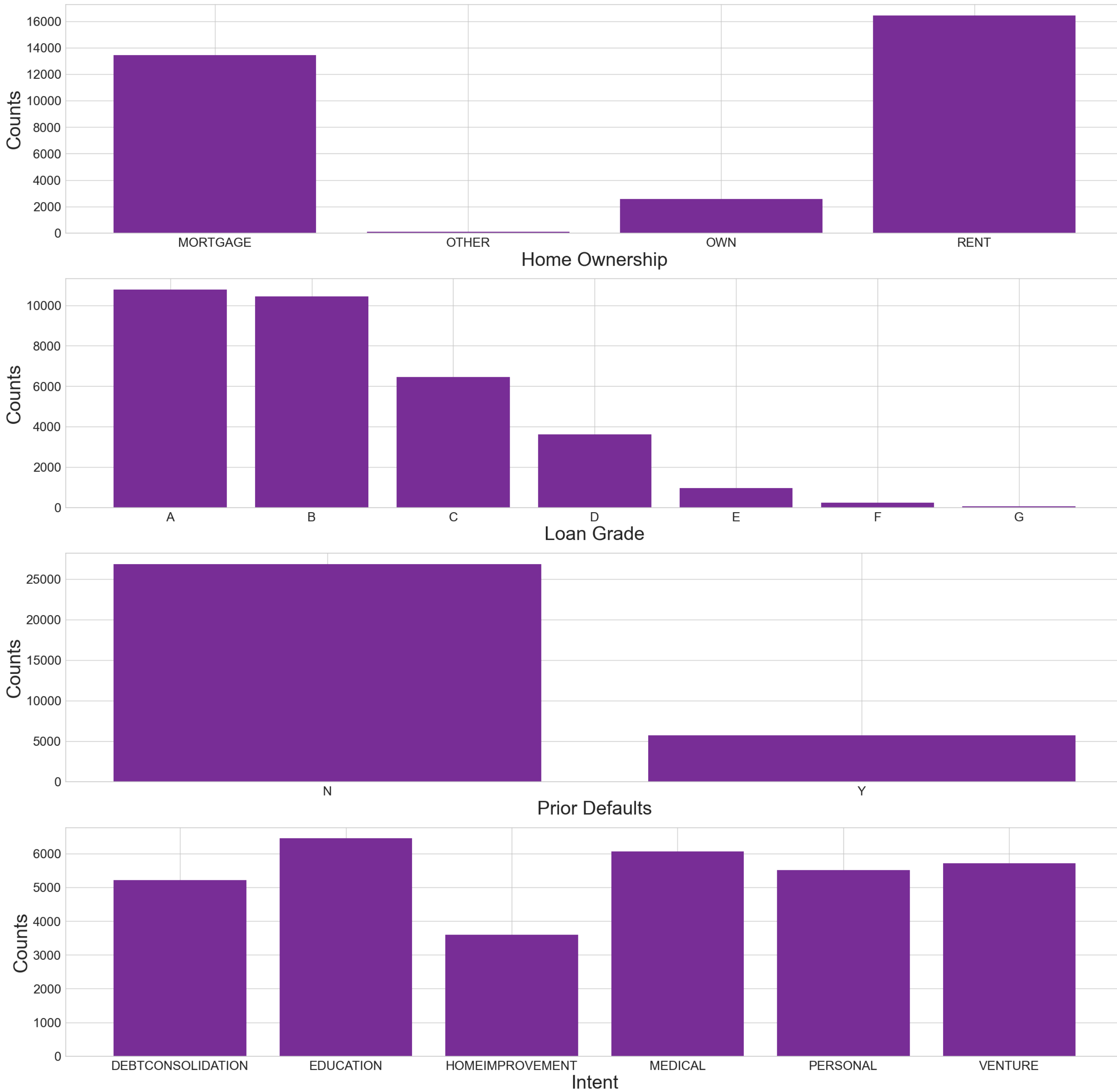
In [9]:
```python
# Nulls counts

null_counts
```

Out[9]:
```
Age                        0
Income                     0
Home Ownership             0
Employment Length        895
Intent                     0
Loan Grade                 0
Loan Amount                0
Interest Rate           3116
Current Loan Status        0
Loan Percent Income        0
Prior Defaults             0
Credit History Length      0
dtype: int64
```

In [10]:
```python
# get histograms of the numerical features
# set the figure size
plt.rcParams['figure.figsize'] = (20, 20)

# make subplots
fig, axes = plt.subplots(nrows = 4, ncols = 2)

# Specify the features of interest
num_features = ['Age', 'Employment Length', 'Loan Amount', 'Interest Rate', 'Loan Percent Income',
                'Credit History Length', 'Current Loan Status', 'Income']
xaxes = num_features
yaxes = ['Count', 'Count', 'Count', 'Count', 'Count', 'Count', 'Count', 'Count']

# draw histograms
axes = axes.ravel()
for idx, ax in enumerate(axes):
    ax.hist(risk_df[num_features[idx]].dropna(), bins=30, color=purple)
    ax.set_xlabel(xaxes[idx], fontsize=20)
    ax.set_ylabel(yaxes[idx], fontsize=20)
    ax.tick_params(axis='both', labelsize=15) #colors='white')

plt.show()
```

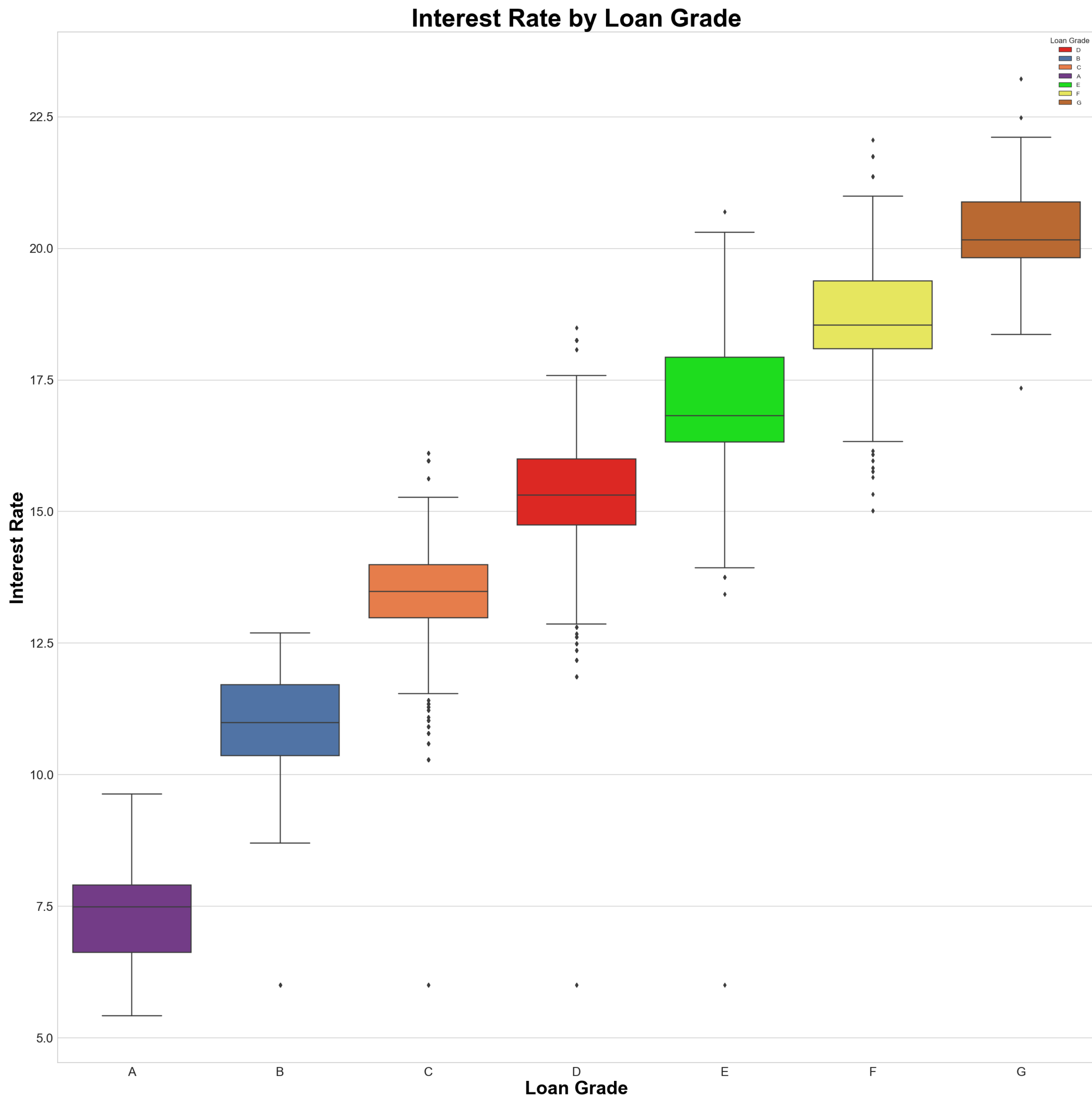


In [11]:
```
# get histograms of the Categorical features
# set the figure size
plt.rcParams['figure.figsize'] = (30, 30)

# make subplots
fig, axes = plt.subplots(nrows = 4, ncols = 1)
#fig.tight_layout()

# make the data read to feed into the visulizer
X_home_o = risk_df.groupby('Home Ownership').size().reset_index(name='Counts')['Home Ownership']
Y_home_o = risk_df.groupby('Home Ownership').size().reset_index(name='Counts')['Counts']
# make the bar plot
axes[0].bar(X_home_o, Y_home_o, color=purple)
axes[0].set_xlabel('Home Ownership', fontsize=30)
axes[0].set_ylabel('Counts', fontsize=30)
axes[0].tick_params(axis='both', labelsize=20)

# make the data read to feed into the visulizer
X_lg = risk_df.groupby('Loan Grade').size().reset_index(name='Counts')['Loan Grade']
Y_lg = risk_df.groupby('Loan Grade').size().reset_index(name='Counts')['Counts']
# make the bar plot
axes[1].bar(X_lg, Y_lg, color=purple)
axes[1].set_xlabel('Loan Grade', fontsize=30)
axes[1].set_ylabel('Counts', fontsize=30)
axes[1].tick_params(axis='both', labelsize=20)

X_pd = risk_df.groupby('Prior Defaults').size().reset_index(name='Counts')['Prior Defaults']
Y_pd = risk_df.groupby('Prior Defaults').size().reset_index(name='Counts')['Counts']
# make the bar plot
axes[2].bar(X_pd, Y_pd, color=purple)
axes[2].set_xlabel('Prior Defaults', fontsize=30)
axes[2].set_ylabel('Counts', fontsize=30)
axes[2].tick_params(axis='both', labelsize=20)

X_intent = risk_df.groupby('Intent').size().reset_index(name='Counts')['Intent']
Y_intent = risk_df.groupby('Intent').size().reset_index(name='Counts')['Counts']
# make the bar plot
axes[3].bar(X_intent, Y_intent, color=purple)
axes[3].set_xlabel('Intent', fontsize=30)
axes[3].set_ylabel('Counts', fontsize=30)
axes[3].tick_params(axis='both', labelsize=20)


plt.show()
```


Counts
16000
14000
12000
10000
8000
6000
4000
2000
0
MORTGAGE
OTHER
OWN
RENT
Home Ownership
Counts
10000
8000
6000
4000
2000
0
A
B
C
D
E
F
G
Loan Grade
Counts
25000
20000
15000
10000
5000
0
N
Y
Prior Defaults
Counts
6000
5000
4000
3000
2000
1000
0
DEBTCONSOLIDATION
EDUCATION
HOMEIMPROVEMENT
MEDICAL
PERSONAL
VENTURE
Intent

In [12]:

```
# High School Graduation Rate by Gender

ir_bp = sns.boxplot(data = risk_df, x = 'Loan Grade', y = 'Interest Rate', hue = 'Loan Grade',
                    order = ['A','B','C','D','E','F','G'], orient = 'v', dodge=False,
                    palette=cust_colors)
ir_bp.set_title('Interest Rate by Loan Grade',
                fontdict={'size': 40, 'weight': 'bold', 'color': 'black'})
ir_bp.set_xlabel('Loan Grade', fontdict={'size': 30, 'weight': 'bold', 'color': 'black'})
ir_bp.set_ylabel('Interest Rate', fontdict={'size': 30, 'weight': 'bold', 'color': 'black'})
plt.tick_params(axis='both', which='major', labelsize=20)

plt.show()
```



Milestone 4

Finalizing Your Results

In [13]:
```
print('1st 10 Rows of Dataset')
risk_df.head(10)
```

```
1st 10 Rows of Dataset
```

Out[13]:

	Age	Income	Home Ownership	Employment Length	Intent	Loan Grade	Loan Amount	Interest Rate	Current Loan Status	Loan Percent Income	Prior Defaults	Credit History Length
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.02	1	0.59	Y	3
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	0	0.10	N	2
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	1	0.57	N	3
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.23	1	0.53	N	2
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.27	1	0.55	Y	4
5	21	9900	OWN	2.0	VENTURE	A	2500	7.14	1	0.25	N	2
6	26	77100	RENT	8.0	EDUCATION	B	35000	12.42	1	0.45	N	3
7	24	78956	RENT	5.0	MEDICAL	B	35000	11.11	1	0.44	N	4
8	24	83000	RENT	8.0	PERSONAL	A	35000	8.90	1	0.42	N	2
9	21	10000	OWN	6.0	VENTURE	D	1600	14.74	1	0.16	N	3

In [14]:
```
# Describe the data

risk_df.describe().apply(lambda s: s.apply('{0:.5f}'.format))
```

Out[14]:

	Age	Income	Employment Length	Loan Amount	Interest Rate	Current Loan Status	Loan Percent Income	Credit History Length
count	32581.00000	32581.00000	31686.00000	32581.00000	29465.00000	32581.00000	32581.00000	32581.00000
mean	27.73460	66074.84847	4.78969	9589.37111	11.01169	0.21816	0.17020	5.80421
std	6.34808	61983.11917	4.14263	6322.08665	3.24046	0.41301	0.10678	4.05500
min	20.00000	4000.00000	0.00000	500.00000	5.42000	0.00000	0.00000	2.00000
25%	23.00000	38500.00000	2.00000	5000.00000	7.90000	0.00000	0.09000	3.00000
50%	26.00000	55000.00000	4.00000	8000.00000	10.99000	0.00000	0.15000	4.00000
75%	30.00000	79200.00000	7.00000	12200.00000	13.47000	0.00000	0.23000	8.00000
max	144.00000	6000000.00000	123.00000	35000.00000	23.22000	1.00000	0.83000	30.00000

In [15]:
```
# Datatypes and non-null counts

print('Credit Risk Column Information')

risk_df.info()
```

```
Credit Risk Column Information
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Age                    32581 non-null  int64  
 1   Income                 32581 non-null  int64  
 2   Home Ownership         32581 non-null  object 
 3   Employment Length      31686 non-null  float64
 4   Intent                 32581 non-null  object 
 5   Loan Grade             32581 non-null  object 
 6   Loan Amount            32581 non-null  int64  
 7   Interest Rate          29465 non-null  float64
 8   Current Loan Status    32581 non-null  int64  
 9   Loan Percent Income    32581 non-null  float64
 10  Prior Defaults         32581 non-null  object 
 11  Credit History Length  32581 non-null  int64  
dtypes: float64(3), int64(5), object(4)
memory usage: 3.0+ MB
```

In [16]:
```
# Number of rows and columns

risk_df.shape
```

Out[16]:
```
(32581, 12)
```

In [17]:
```
# Percentage of nulls per columns

null_counts = risk_df.isna().sum()
total = len(risk_df)

print('Percentage Columns with NULLS')
pct_null = round((null_counts / total) * 100,1)
pct_null
```

```
Percentage Columns with NULLS
```

Out[17]:
```
Age                      0.0
Income                   0.0
Home Ownership           0.0
Employment Length        2.7
Intent                   0.0
Loan Grade               0.0
Loan Amount              0.0
Interest Rate            9.6
Current Loan Status      0.0
Loan Percent Income      0.0
Prior Defaults           0.0
Credit History Length    0.0
dtype: float64
```

In [18]:
```
# Null Counts

null_counts
```

Out[18]:
```
Age                         0
Income                      0
Home Ownership              0
Employment Length         895
Intent                      0
Loan Grade                  0
Loan Amount                 0
Interest Rate            3116
Current Loan Status         0
Loan Percent Income         0
Prior Defaults              0
Credit History Length       0
dtype: int64
```

In [19]:
```
# Fill in the interest rate with average by loan grade

risk_df['Interest Rate'] = risk_df.groupby('Loan Grade')['Interest Rate'].transform(lambda x: x.fillna(x.mean()))

risk_df['Employment Length'] = risk_df['Employment Length'].fillna(risk_df['Employment Length'].mean())

# Remove Outliers 1 Outlier at a time
# Remove rows with age > 84
risk_df = risk_df[(risk_df.Age <= 84)]

# Remove row with Employment Length = 123 years
risk_df = risk_df[(risk_df['Employment Length'] != 123)]

# Remove rows with income >= 300000
risk_df = risk_df[(risk_df.Income < 300000)]
```

In [20]:
```
# Check for Nulls again

null_counts_a = risk_df.isna().sum()
total_a = len(risk_df)

print('Percentage Columns with NULLS')
pct_null_a = round((null_counts_a / total_a) * 100,1)
pct_null_a
```

```
Percentage Columns with NULLS
```

Out[20]:
```
Age                     0.0
Income                  0.0
Home Ownership          0.0
Employment Length       0.0
Intent                  0.0
Loan Grade              0.0
Loan Amount             0.0
Interest Rate           0.0
Current Loan Status     0.0
Loan Percent Income     0.0
Prior Defaults          0.0
Credit History Length   0.0
dtype: float64
```

In [21]:
```
# Null counts after cleanup

null_counts_a
```

Out[21]:
```
Age                     0
Income                  0
Home Ownership          0
Employment Length       0
Intent                  0
Loan Grade              0
Loan Amount             0
Interest Rate           0
Current Loan Status     0
Loan Percent Income     0
Prior Defaults          0
Credit History Length   0
dtype: int64
```

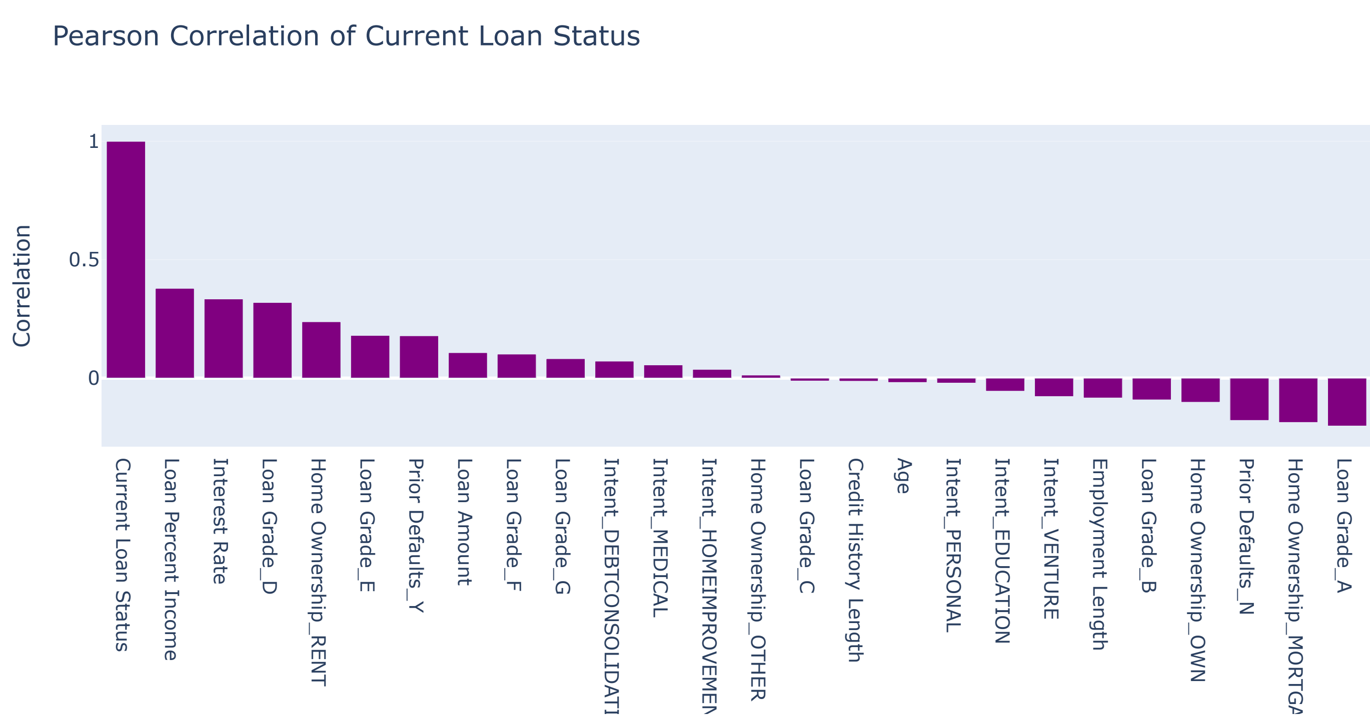
In [22]:
```
# Get Categorical Columns and Create Dummy Columns

# Make a copy of the Credit Risk df
risk_df1 = risk_df.copy()

# Get the categorical columns
object_cols = risk_df1.select_dtypes("object").columns
object_cols = list(set(object_cols))

# Create dummy records
risk_dummy_df = pd.get_dummies(risk_df1, columns = object_cols)
```

In [23]:
```python
# Describe the data after cleanup

round(risk_df1.apply(lambda x: x.factorize()[0]).corr(), 2)
```

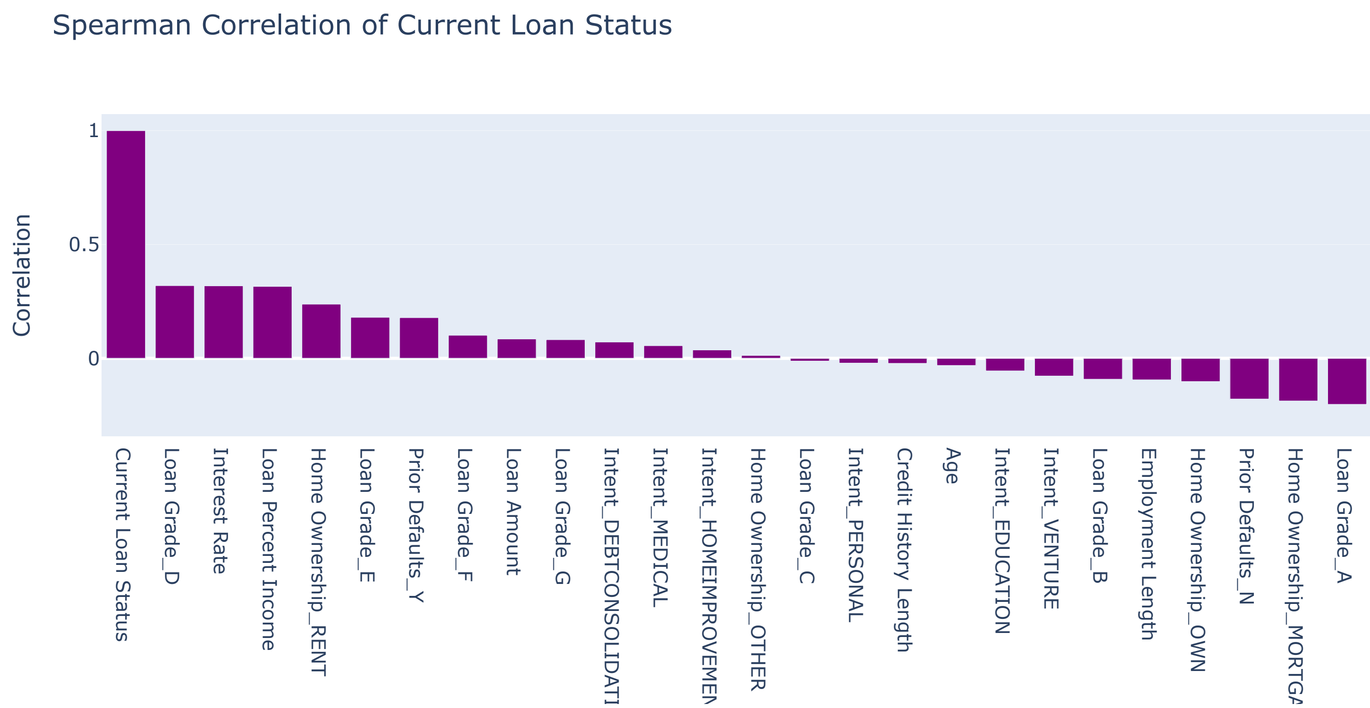
Out[23]:

	Age	Income	Home Ownership	Employment Length	Intent	Loan Grade	Loan Amount	Interest Rate	Current Loan Status	Loan Percent Income	Prior Defaults	Credit History Length
Age	1.00	0.14	-0.02	0.19	0.06	0.01	0.03	-0.00	-0.02	0.01	0.01	0.89
Income	0.14	1.00	-0.03	0.07	-0.00	0.02	0.04	0.02	0.02	0.01	0.02	0.13
Home Ownership	-0.02	-0.03	1.00	-0.11	0.01	-0.02	0.00	0.01	0.23	-0.02	0.05	-0.02
Employment Length	0.19	0.07	-0.11	1.00	0.03	0.01	-0.00	-0.01	-0.03	0.01	-0.01	0.18
Intent	0.06	-0.00	0.01	0.03	1.00	0.02	-0.01	-0.00	0.07	0.00	0.01	0.06
Loan Grade	0.01	0.02	-0.02	0.01	0.02	1.00	-0.00	0.16	0.25	-0.00	0.25	0.01
Loan Amount	0.03	0.04	0.00	-0.00	-0.01	-0.00	1.00	-0.01	0.00	-0.02	-0.01	0.03
Interest Rate	-0.00	0.02	0.01	-0.01	-0.00	0.16	-0.01	1.00	0.07	0.01	0.11	-0.01
Current Loan Status	-0.02	0.02	0.23	-0.03	0.07	0.25	0.00	0.07	1.00	-0.08	0.18	-0.01
Loan Percent Income	0.01	0.01	-0.02	0.01	0.00	-0.00	-0.02	0.01	-0.08	1.00	-0.02	0.00
Prior Defaults	0.01	0.02	0.05	-0.01	0.01	0.25	-0.01	0.11	0.18	-0.02	1.00	0.00
Credit History Length	0.89	0.13	-0.02	0.18	0.06	0.01	0.03	-0.01	-0.01	0.00	0.00	1.00

In [24]:
```python
# Using the Pearson method, Correlate all columns including
# dummy variables to the target variable (Current Loan Status)

p_corr_cls = risk_dummy_df.corr()['Current Loan Status'].sort_values(ascending=False)

p_corr_bp = px.bar(x = p_corr_cls.keys(), y = p_corr_cls.values,
                   color_discrete_sequence = ["purple"],
                   title = 'Pearson Correlation of Current Loan Status',
                   labels = {'x':'Features', 'y': 'Correlation'})

p_corr_bp.show()
```



In [25]:
```
# Using the Spearman method, Correlate all columns including
# dummy variables to the target variable (Current Loan Status)

sp_corr_cls = risk_dummy_df.corr(method = 'spearman')['Current Loan Status'].sort_values(ascending=False)

sp_corr_bp = px.bar(x = sp_corr_cls.keys(), y = sp_corr_cls.values,
                 color_discrete_sequence = ["purple"],
                 title = 'Spearman Correlation of Current Loan Status',
                 labels = {'x':'Features', 'y': 'Correlation'})

sp_corr_bp.show()
```



Build and Evaluate Models

In [26]:
```
# Split credit risk data into training and testing data sets

# Delete the Status columns
no_status_df = risk_dummy_df.drop(['Current Loan Status'], axis = 1)

# Create Status Target column
status_df = risk_dummy_df['Current Loan Status']

# Split data into 80/20 sets
x_train, x_test, y_train, y_test = train_test_split(no_status_df, status_df,
                                                    test_size = 0.2, random_state = 1)

print('Number of x_train Rows and Columns = ', x_train.shape)
print('Number of y_train Rows and Columns = ', y_train.shape)
print('Number of x_test Rows and Columns = ', x_test.shape)
print('Number of y_test Rows and Columns = ', y_test.shape)
```

```
Number of x_train Rows and Columns =  (25924, 26)
Number of y_train Rows and Columns =  (25924,)
Number of x_test Rows and Columns =  (6482, 26)
Number of y_test Rows and Columns =  (6482,)
```

Decision Tree Classifier

In [27]:
```
# Decision Tree Classifier

# Create the Decision Tree Classifier,
# train the model, and test the results

# Create a Decision Tree object
#dtc = DecisionTreeClassifier()
dtc = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)

# Train the Decision Tree model
dtc.fit(x_train, y_train)

# Train model to make predictions
y_pred = dtc.predict(x_test)

# Calculate accuracy
ac_score = accuracy_score(y_test, y_pred)
print('The Accuracy Score For Decision Tree Classifier = ', round(100 * ac_score, 2), '%', sep = '')
```

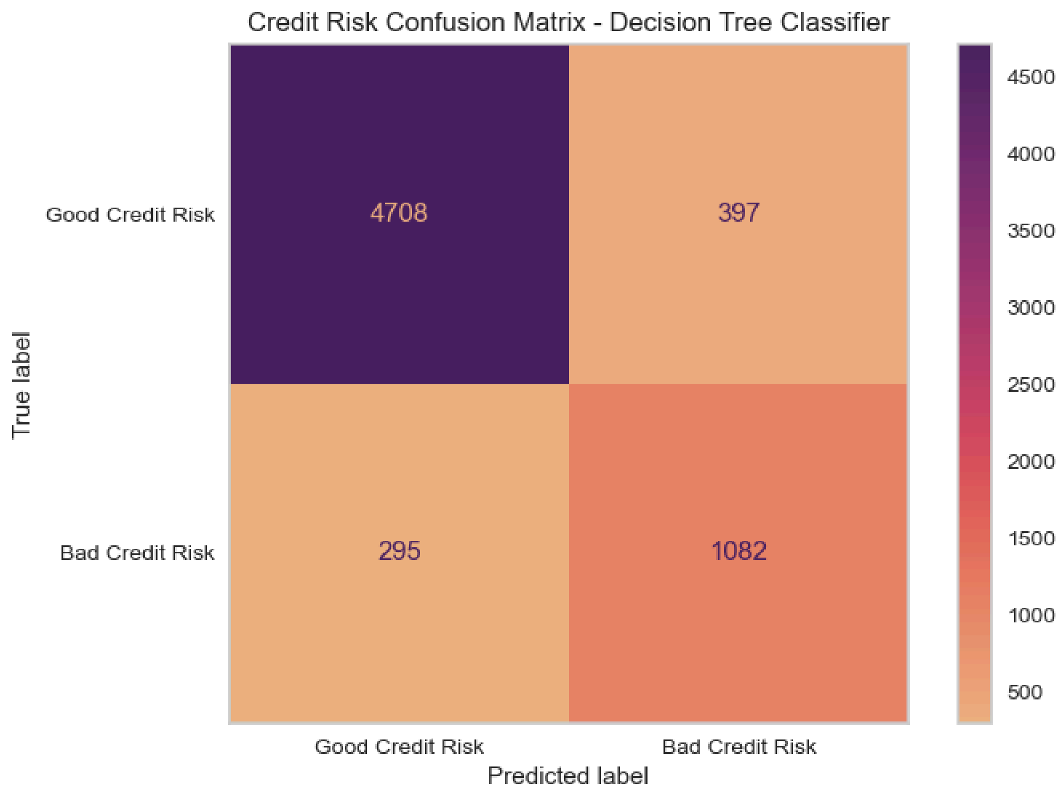
```
The Accuracy Score For Decision Tree Classifier = 89.32%
```

In [28]:
```
# Double check the accuracy

target_names = ['Good Credit Risk', 'Bad Credit Risk']
class_rpt = classification_report(y_test, y_pred, target_names=target_names)

print('Classification Report For Decision Tree Classifier')
print(class_rpt)
```

```
Classification Report For Decision Tree Classifier
                  precision    recall  f1-score   support

Good Credit Risk       0.94      0.92      0.93      5105
 Bad Credit Risk       0.73      0.79      0.76      1377

        accuracy                           0.89      6482
       macro avg       0.84      0.85      0.84      6482
    weighted avg       0.90      0.89      0.89      6482
```

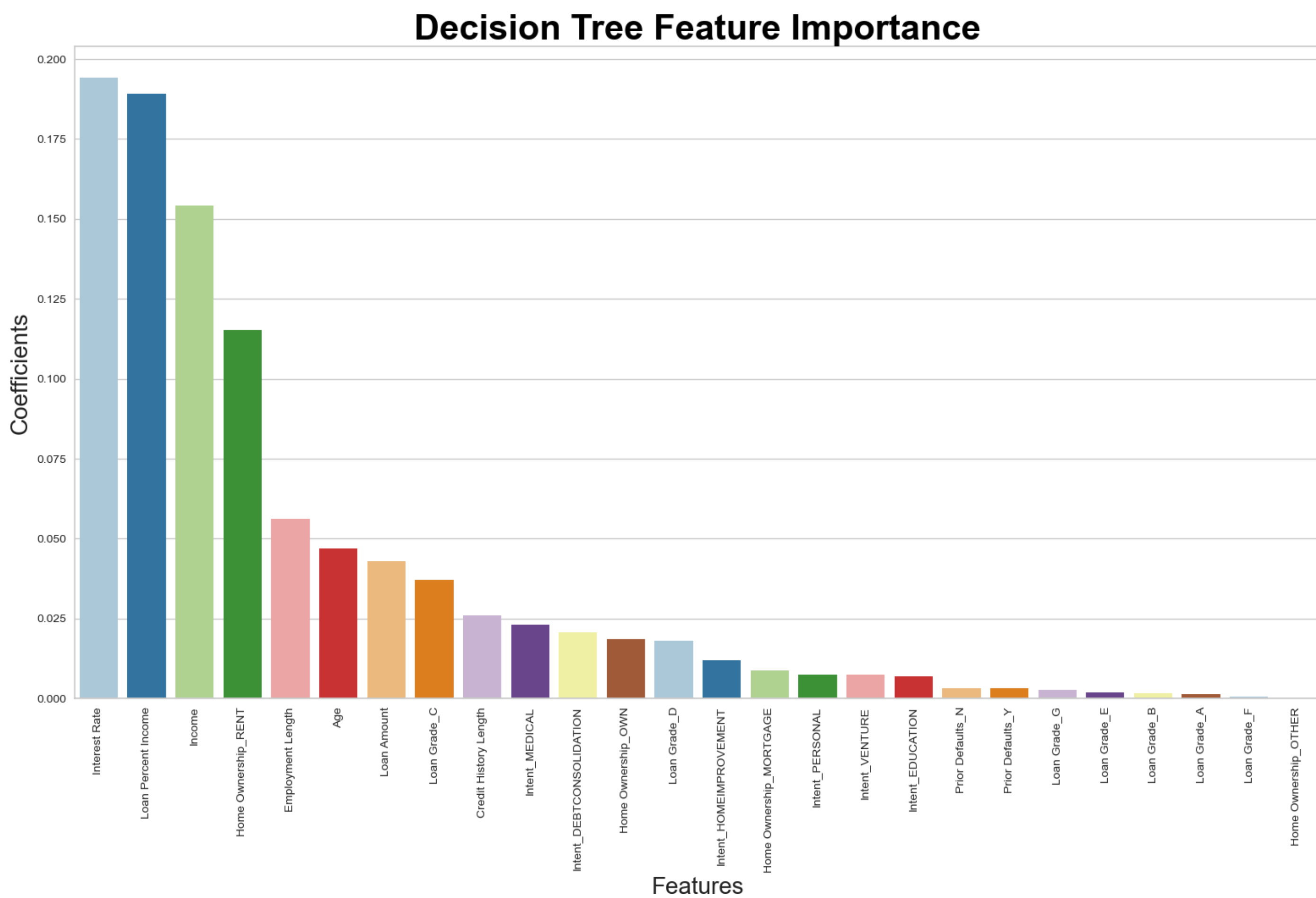
In [29]:
```
# create and plot a confusion matrix

# Reset figure size
plt.rcParams["figure.figsize"] = (8.0, 5.5)

# Set up labels
labels = ['Good Credit Risk', 'Bad Credit Risk']

# Plot confusion matrix
fig = plt.figure()
plot_confusion_matrix(dtc, x_test, y_test, display_labels = labels, cmap = "flare")
plt.title('Credit Risk Confusion Matrix - Decision Tree Classifier')
plt.grid(False)

plt.show()
```

```
<Figure size 800x550 with 0 Axes>
```



Features of the Decision Tree Classifier

In [30]:
```
# Display the features for the Decision Tree in order of importance

# Identify and rank the important features
importances_df = pd.DataFrame(data={
    'Attribute': x_train.columns,
    'Importance': dtc.feature_importances_
})
importances_df = importances_df.sort_values(by = 'Importance', ascending = False)

# Display the features in order of importance
fig, axes = plt.subplots(figsize = (19, 10))

import_bp = sns.barplot(x = 'Attribute', y = 'Importance', data = importances_df,
                        ci = None, palette = 'Paired')

import_bp.set_title('Decision Tree Feature Importance',
                    fontdict={'size': 30, 'weight': 'bold', 'color': 'black'})
import_bp.set_xlabel('Features', fontdict={'size': 20})
import_bp.set_ylabel('Coefficients', fontdict={'size': 20})

# rotate x-axis labels by 40 degrees
plt.xticks(rotation = 90)

# Show the plot
plt.show()
```



Identify the Best 5 Features.

In [31]:
```
# Create target and feature_names list
# to identify the Best 5 Features
features = x_train
target = y_train

# Select 5 features with highest chi-squared statistics
chi2_selector = SelectKBest(chi2, k = 5)
best_5_features = chi2_selector.fit_transform(features, target)

# Show results
print ("Original Number of Features:", features.shape[1])
print ("Number of Features After Best 5:", best_5_features.shape[1])
```

```
Original Number of Features: 26
Number of Features After Best 5: 5
```

In [32]:
```
# Create a dataframe with the Best 5 features

best_5_cols = chi2_selector.get_support(indices = True)
best_5_df = features.iloc[:,best_5_cols]

print('1st 10 Rows of the Best 5 Features')
best_5_df.head(10)
```

```
1st 10 Rows of the Best 5 Features
```

Out[32]:

	Income	Loan Amount	Interest Rate	Loan Grade_D	Loan Grade_E
9388	70000	8000	11.990000	0	0
4898	39000	5000	7.490000	0	0
32033	62400	2400	7.490000	0	0
9196	60000	11500	11.260000	0	0
31689	121200	1450	6.760000	0	0
7253	37000	6300	10.990000	0	0
9719	63996	4000	5.420000	0	0
1551	27600	9600	8.590000	0	0
12844	85000	8000	7.327651	0	0
22549	55640	20000	10.995555	0	0

In [33]:
```
# Split the data into a training and test set for the Best 5 Features

x_best_5 = risk_dummy_df[['Income', 'Loan Amount', 'Interest Rate',
                'Loan Grade_D', 'Loan Grade_E']]
# Create the Target column
y_best_5 = risk_dummy_df['Current Loan Status']

# Split data into 80/20 sets
x_train_best_5, x_test_best_5, y_train_best_5, y_test_best_5 = train_test_split(x_best_5,
                                                                y_best_5, test_size = 0.2)

print('Number of x_train_best_5 Rows and Columns = ', x_train_best_5.shape)
print('Number of y_train_best_5 Rows and Columns = ', y_train_best_5.shape)
print('Number of x_test_best_5 Rows and Columns = ', x_test_best_5.shape)
print('Number of y_test_best_5 Rows and Columns = ', y_test_best_5.shape)
```

```
Number of x_train_best_5 Rows and Columns =  (25924, 5)
Number of y_train_best_5 Rows and Columns =  (25924,)
Number of x_test_best_5 Rows and Columns =  (6482, 5)
Number of y_test_best_5 Rows and Columns =  (6482,)
```

In [34]:
```
# Decision Tree Classifier for Best 5 Features

# Create the Decision Tree Classifier,
# train the model, and test the results

# Create a Decision Tree object
dtc_5 = DecisionTreeClassifier()

# Train the Decision Tree model
dtc_5.fit(x_train_best_5, y_train_best_5)

# Train model to make predictions
y_pred_5 = dtc_5.predict(x_test_best_5)

# Calculate accuracy
ac_score_5 = accuracy_score(y_test_best_5, y_pred_5)
print('The Accuracy Score For Decision Tree Classifier = ', round(100 * ac_score_5, 2), '%', sep = '')
```

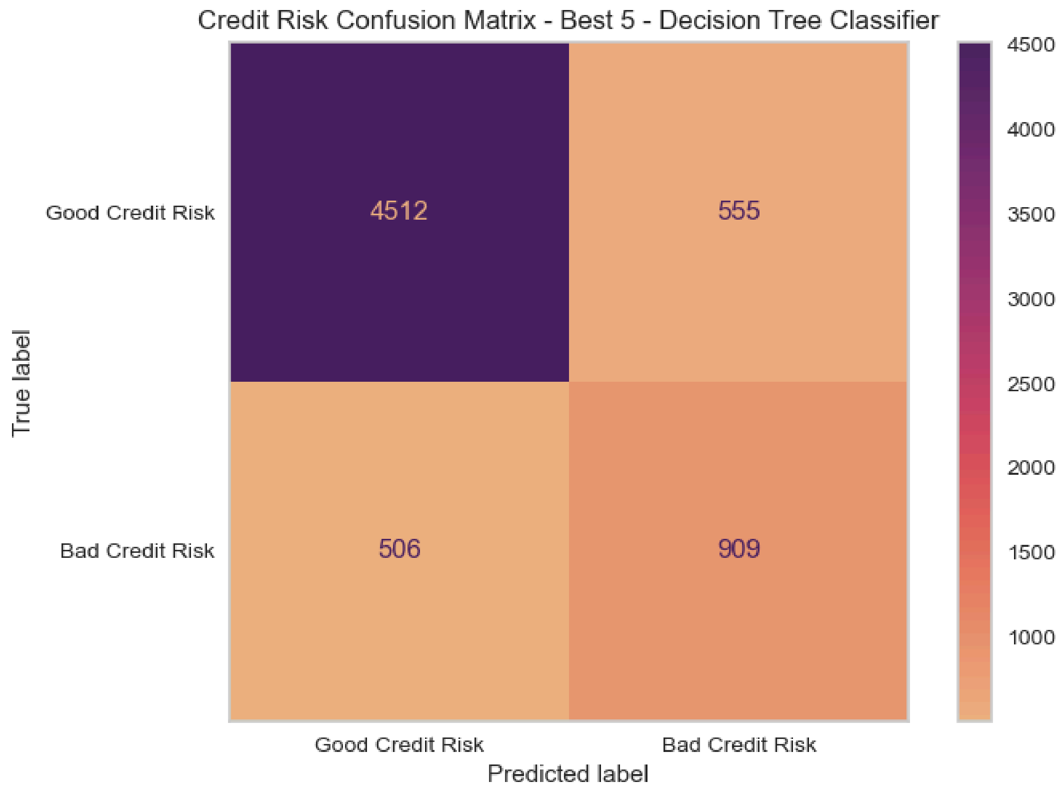
```
The Accuracy Score For Decision Tree Classifier = 83.63%
```

In [35]:
```
# Double check the accuracy

class_rpt_5 = classification_report(y_test_best_5, y_pred_5, target_names = target_names)

print('Classification Report For Best 5 - Decision Tree Classifier')
print(class_rpt_5)
```

```
Classification Report For Best 5 - Decision Tree Classifier
                  precision    recall  f1-score   support

Good Credit Risk       0.90      0.89      0.89      5067
 Bad Credit Risk       0.62      0.64      0.63      1415

        accuracy                           0.84      6482
       macro avg       0.76      0.77      0.76      6482
    weighted avg       0.84      0.84      0.84      6482
```

In [36]:
```
# create and plot a confusion matrix

# Plot confusion matrix
fig = plt.figure()
plot_confusion_matrix(dtc_5, x_test_best_5, y_test_best_5,
                      display_labels = labels, cmap = "flare")
plt.title('Credit Risk Confusion Matrix - Best 5 - Decision Tree Classifier')
plt.grid(False)

plt.show()
```

```
<Figure size 800x550 with 0 Axes>
```



Logistic Regression Model

In [37]:
```
# Create the Logistic Regression Model,
# train the model, and test the results

# Create logistic regression object
logit = LogisticRegression(solver="liblinear", random_state=0)

# Train the logistic regression model
logit.fit(x_train, y_train)

# Train model to make predictions
y_pred = logit.predict(x_test)

# Calculate accuracy
ac_score = accuracy_score(y_test, y_pred)
print('The Accuracy Score For Logistic Regression = ', round(100 * ac_score, 2), '%', sep = '')
```

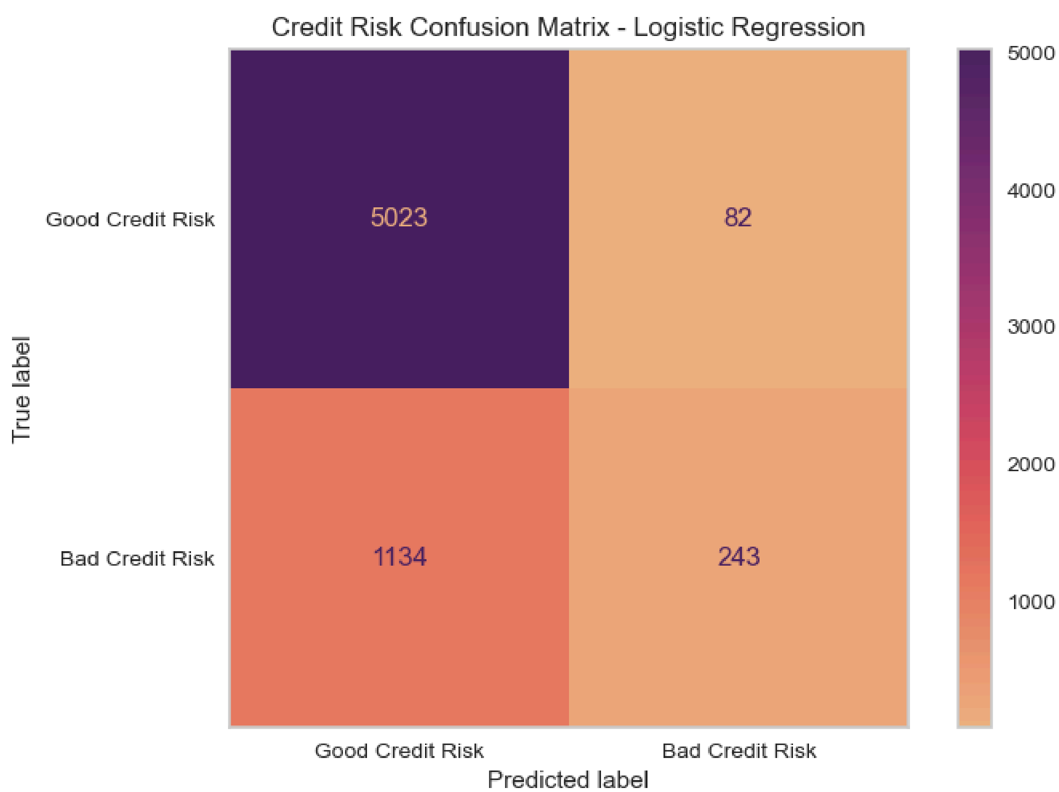
```
The Accuracy Score For Logistic Regression = 81.24%
```

In [38]:
```python
# Double check the accuracy

class_rpt = classification_report(y_test, y_pred, target_names=target_names)

print('Classification Report For Logistic Regression')
print(class_rpt)
```

```
Classification Report For Logistic Regression
                  precision    recall  f1-score   support

Good Credit Risk       0.82      0.98      0.89      5105
 Bad Credit Risk       0.75      0.18      0.29      1377

        accuracy                           0.81      6482
       macro avg       0.78      0.58      0.59      6482
    weighted avg       0.80      0.81      0.76      6482
```

In [39]:
```python
# create and plot a confusion matrix

# Plot confusion matrix using the labels in the Decision Tree
fig = plt.figure()
plot_confusion_matrix(logit, x_test, y_test, display_labels = labels, cmap = "flare")
plt.title('Credit Risk Confusion Matrix - Logistic Regression')
plt.grid(False)

plt.show()
```

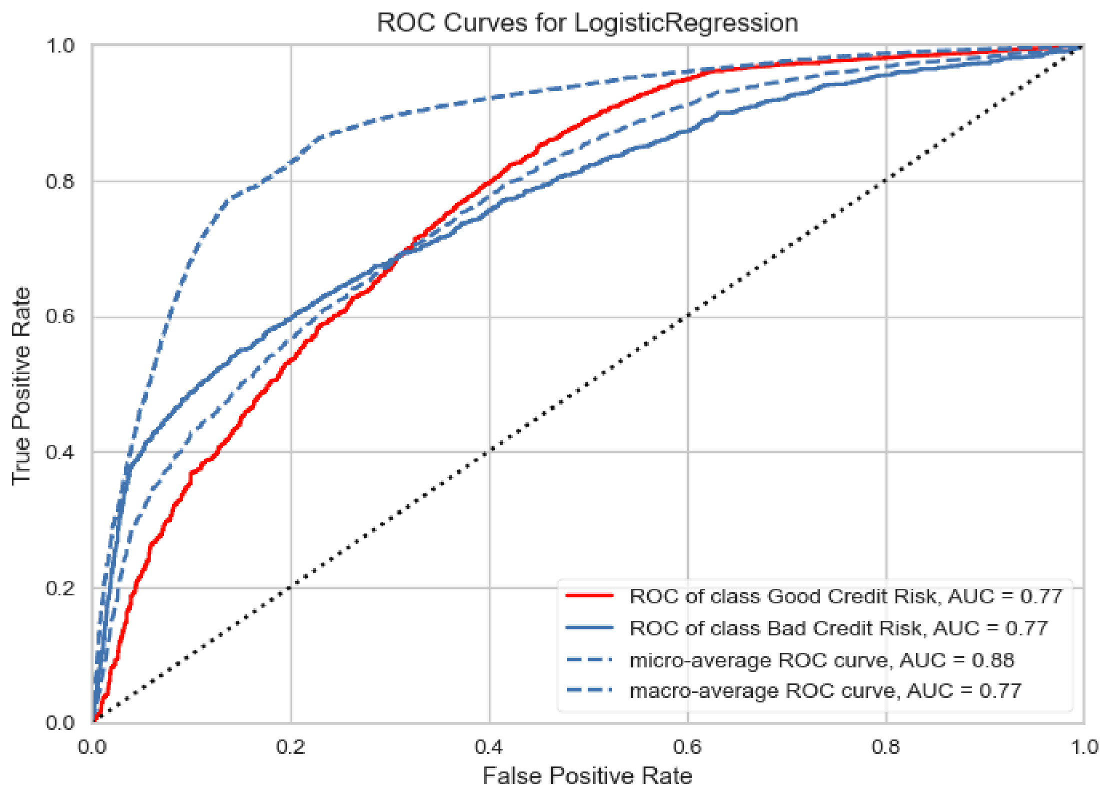
```
<Figure size 800x550 with 0 Axes>
```



Receiver Operating Characteristic (ROC) with Area Under the Curve (AUC) for the Logistic Regression Model

In [40]:
```
# Create Receiver Operating Characteristic (ROC) with Area Under the Curve (AUC)
# For the Logistic Regression Model

#fig, axes = plt.subplots(figsize = (20, 10))

# Set up the ROC visualizer
ROC_labels = {0: 'Good Credit Risk', 1: 'Bad Credit Risk'}
roc_curve = ROCAUC(logit, encoder = ROC_labels, solver = 'liblinear')

# Fit the test data
roc_curve.fit(x_test, y_test)

# Display the model on the test data
roc_score = roc_curve.score(x_test, y_test)
print(roc_score)
roc_curve.show()
```

```
0.7673002898800382
```



Out[40]:
```
<AxesSubplot:title={'center':'ROC Curves for LogisticRegression'}, xlabel='False Positive Rate', ylabel='True Pos
itive Rate'>
```