Problem Statement

Although different banks set their own approval standards, they generally look out for the same type of customers. From machine learning, we can predict the indicators that will drive a successful credit card application.

Import neccessary libraries

In [302…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Import Dataset

In [303…

```
#data set from: https://www.kaggle.com/datasets/samuelcortinhas/credit-card-approve

df = pd.read_csv('/Users/alisonc/Documents/PERSONAL/Data Science Bootcamp/credit_ca
```

Data cleaning & EDA

In [304…

```
#get info

df.shape
df.head()
df.info() #check for strings that might relate to the approval
df.describe()
```

Out[304]: (690, 16)

Out[304]:

	Gender	Age	Debt	Married	BankCustomer	Industry	Ethnicity	YearsEmployed	PriorDefau
0	1	30.83	0.000	1	1	Industrials	White	1.25	
1	0	58.67	4.460	1	1	Materials	Black	3.04	
2	0	24.50	0.500	1	1	Materials	Black	1.50	
3	1	27.83	1.540	1	1	Industrials	White	3.75	
4	1	20.17	5.625	1	1	Industrials	White	1.71	

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Gender          690 non-null    int64  
 1   Age             690 non-null    float64
 2   Debt            690 non-null    float64
 3   Married         690 non-null    int64  
 4   BankCustomer    690 non-null    int64  
 5   Industry        690 non-null    object 
 6   Ethnicity       690 non-null    object 
 7   YearsEmployed   690 non-null    float64
 8   PriorDefault    690 non-null    int64  
 9   Employed        690 non-null    int64  
 10  CreditScore     690 non-null    int64  
 11  DriversLicense  690 non-null    int64  
 12  Citizen         690 non-null    object 
 13  ZipCode         690 non-null    int64  
 14  Income          690 non-null    int64  
 15  Approved        690 non-null    int64  
dtypes: float64(3), int64(10), object(3)
memory usage: 86.4+ KB
```

Out[304]:

	Gender	Age	Debt	Married	BankCustomer	YearsEmployed	PriorDefau
count	690.000000	690.000000	690.000000	690.000000	690.000000	690.000000	690.00000
mean	0.695652	31.514116	4.758725	0.760870	0.763768	2.223406	0.52318
std	0.460464	11.860245	4.978163	0.426862	0.425074	3.346513	0.49982
min	0.000000	13.750000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.000000	22.670000	1.000000	1.000000	1.000000	0.165000	0.00000
50%	1.000000	28.460000	2.750000	1.000000	1.000000	1.000000	1.00000
75%	1.000000	37.707500	7.207500	1.000000	1.000000	2.625000	1.00000

Legend

```
Gender: 0=Male 1=Female
Debt: Outstanding Debt
Married: 0=Single/Divorced 1=Married
BankCustomer: 0=Not customer 1=Customer
PriorDefault: 0=Not employed 1=Employed
DriversLicense: 0=No license 1=With license
Approved: 0=Not approved 1=Approved
```

Rename column header for better understanding

In [305…

```
df = df.rename({'Debt':'OutsDebt','Approved':'ApprovalStatus'},axis=1)
```

Identify the objects and find out the value count of them

In [306…

```
string_cols = df.select_dtypes('object').columns

string_cols
```

```python
value_count = df[string_cols].apply(pd.value_counts, normalize=True)
value_count
value_count.index # preview to multiindex, you also get this when doing df.groupby

# we don't need such high granularity usually, so do in loop
for col in string_cols:
    df[col].value_counts()
```

Out[306]:
```
Index(['Industry', 'Ethnicity', 'Citizen'], dtype='object')
```

Out[306]:

	Industry	Ethnicity	Citizen
Asian	NaN	0.085507	NaN
Black	NaN	0.200000	NaN
ByBirth	NaN	NaN	0.905797
ByOtherMeans	NaN	NaN	0.082609
CommunicationServices	0.055072	NaN	NaN
ConsumerDiscretionary	0.085507	NaN	NaN
ConsumerStaples	0.078261	NaN	NaN
Education	0.036232	NaN	NaN
Energy	0.211594	NaN	NaN
Financials	0.073913	NaN	NaN
Healthcare	0.076812	NaN	NaN
Industrials	0.092754	NaN	NaN
InformationTechnology	0.059420	NaN	NaN
Latino	NaN	0.082609	NaN
Materials	0.113043	NaN	NaN
Other	NaN	0.040580	NaN
Real Estate	0.043478	NaN	NaN
Research	0.014493	NaN	NaN
Temporary	NaN	NaN	0.011594
Transport	0.004348	NaN	NaN
Utilities	0.055072	NaN	NaN
White	NaN	0.591304	NaN

Out[306]:
```
Index(['Asian', 'Black', 'ByBirth', 'ByOtherMeans', 'CommunicationServices',
       'ConsumerDiscretionary', 'ConsumerStaples', 'Education', 'Energy',
       'Financials', 'Healthcare', 'Industrials', 'InformationTechnology',
       'Latino', 'Materials', 'Other', 'Real Estate', 'Research', 'Temporary',
       'Transport', 'Utilities', 'White'],
      dtype='object')
```

Out[306]:
```
Energy                   146
Materials                 78
Industrials               64
ConsumerDiscretionary     59
ConsumerStaples           54
Healthcare                53
Financials                51
InformationTechnology     41
CommunicationServices     38
Utilities                 38
Real Estate               30
Education                 25
Research                  10
Transport                  3
Name: Industry, dtype: int64
```

Out[306]:
```
White     408
Black     138
Asian      59
Latino     57
Other      28
Name: Ethnicity, dtype: int64
```

Out[306]:
```
ByBirth         625
ByOtherMeans     57
Temporary         8
Name: Citizen, dtype: int64
```

In [307…
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

for col in df:
    if df[col].dtypes=='object':
        df[col]=le.fit_transform(df[col])

df
```

Out[307]:

	Gender	Age	OutsDebt	Married	BankCustomer	Industry	Ethnicity	YearsEmployed	Prior
0	1	30.83	0.000	1	1	7	4	1.25	
1	0	58.67	4.460	1	1	9	1	3.04	
2	0	24.50	0.500	1	1	9	1	1.50	
3	1	27.83	1.540	1	1	7	4	3.75	
4	1	20.17	5.625	1	1	7	4	1.71	
...	...	...	...	...	...	...	...	...	
685	1	21.08	10.085	0	0	3	1	1.25	
686	0	22.67	0.750	1	1	4	4	2.00	
687	0	25.25	13.500	0	0	6	2	2.00	
688	1	17.92	0.205	1	1	2	4	0.04	
689	1	35.00	3.375	1	1	4	1	8.29	

690 rows × 16 columns

Check for duplicates

In [308…
```
df.duplicated().sum()

#Result: 0 duplicates to drop
```
Out[308]:
```
0
```

Results: No duplicates to drop

Check for missing data

In [309…
```
df.isnull().sum().sum()

#Result: 0 null to fill
```
Out[309]:
```
0
```

Results: No missing data to fill

Observe the mean of the varients compared to the approval status

In [310…
```
df.groupby('ApprovalStatus').mean()
```
Out[310]:

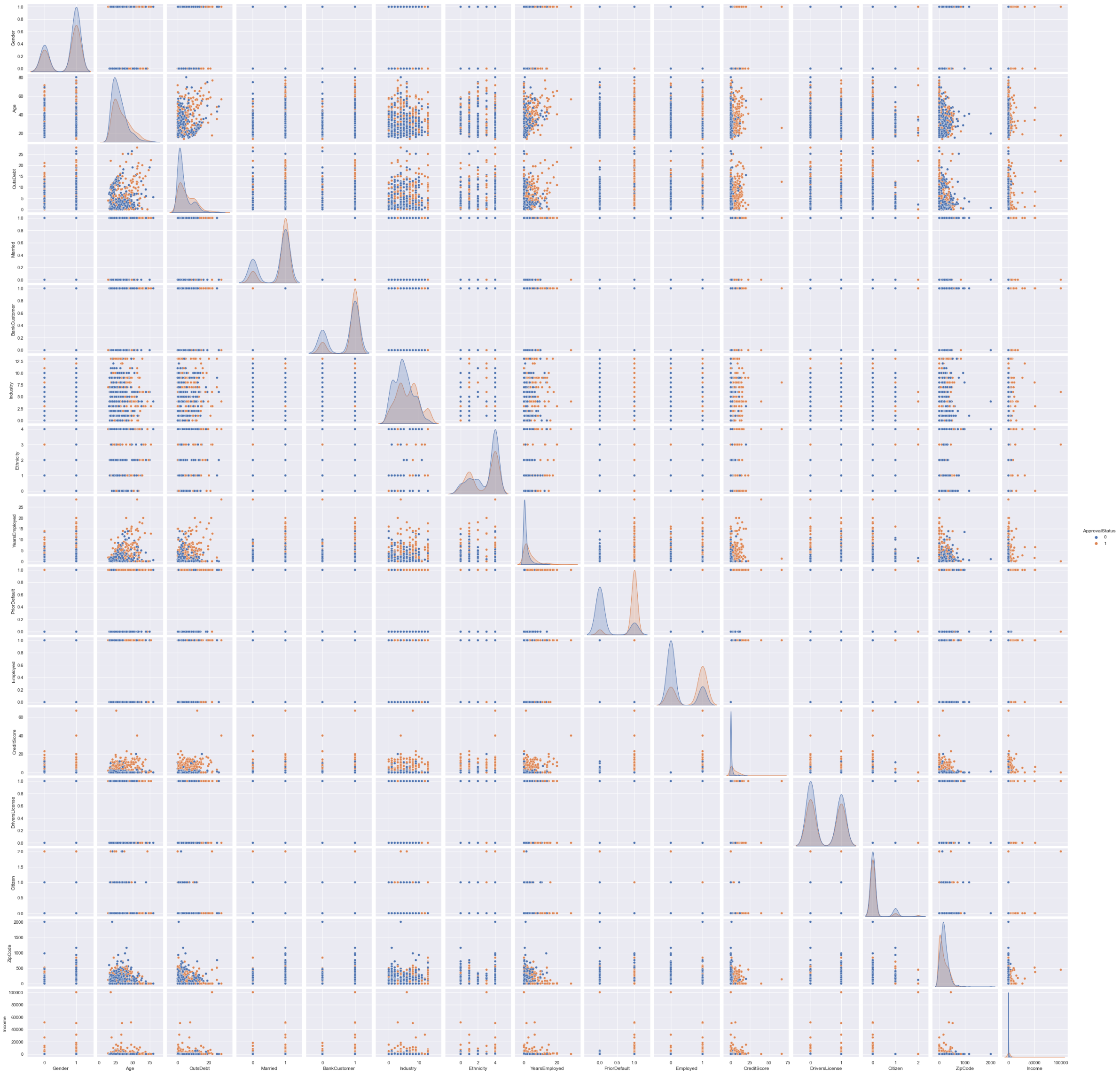
	Gender	Age	OutsDebt	Married	BankCustomer	Industry	Ethnicity	Year
ApprovalStatus								
0	0.707572	29.773029	3.839948	0.691906	0.691906	4.913838	2.953003	
1	0.680782	33.686221	5.904951	0.846906	0.853420	6.302932	2.726384	

Basic observation through mean

- The mean of Gender and Age in relations to the approval status does not have a big variance. It does not seem to affect approval status.
- A higher Outstanding Debt did not seem to affect the approval rate. It could be that the bank also consider on time debt repayment, as long as the repayment is done on time, it will not affect the approval rate.
- Being Married or a Customer of the bank have very little affect on the approval rate as well.
- Longer Years Employed, Being Employed, good Credit Score and a high income does seem to have push more approvals
- Having Prior Payment seem to lower the approval rate
- Having a Drivers License and Zipcode location dont seem to have any relations to the approval status

Use Pairplot to identify correlations of the different varients

In [313…
```
sns.pairplot(df, hue ='ApprovalStatus')
plt.show()
```
Out[313]:
```
<seaborn.axisgrid.PairGrid at 0x1a491adc2b0>
```



Identify correlations of the varients through pairplot

- Gender, Marital Status, Citizen, Bank Customer, Employed, Ethnicity and Drivers License does not seem to have any correlations to any other variants
- Age, Industry, Oustanding Debt, Years Employed, Prior Default, Credit Score, Zipcode and Income seem to have come correslations to each other that we can explore and understand better.

Further use of heatmap to observe correlations

In [314…

```
df.corr()
```

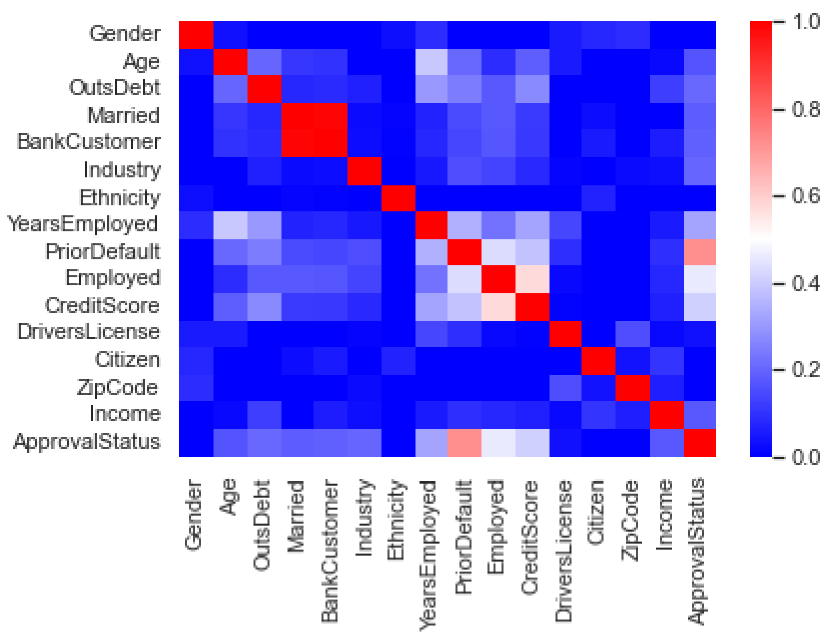
Out[314]:

	Gender	Age	OutsDebt	Married	BankCustomer	Industry	Ethnicity	Y
Gender	1.000000	0.035044	-0.041746	-0.068062	-0.071250	-0.111889	0.029492	
Age	0.035044	1.000000	0.202177	0.106929	0.099477	-0.038746	-0.179534	
OutsDebt	-0.041746	0.202177	1.000000	0.074649	0.083781	0.064553	-0.075789	
Married	-0.068062	0.106929	0.074649	1.000000	0.992033	0.021652	0.008226	
BankCustomer	-0.071250	0.099477	0.083781	0.992033	1.000000	0.024677	0.006648	
Industry	-0.111889	-0.038746	0.064553	0.021652	0.024677	1.000000	-0.013881	
Ethnicity	0.029492	-0.179534	-0.075789	0.008226	0.006648	-0.013881	1.000000	
YearsEmployed	0.086544	0.391464	0.298902	0.069945	0.075905	0.048689	-0.177111	
PriorDefault	-0.026047	0.204434	0.244317	0.145073	0.138535	0.154645	-0.114148	
Employed	-0.077784	0.086037	0.174846	0.175428	0.170268	0.134769	-0.030250	
CreditScore	-0.024630	0.187327	0.271207	0.113968	0.111077	0.080107	-0.068072	
DriversLicense	0.051674	0.053599	-0.013023	-0.009784	-0.002402	0.011010	-0.035688	
Citizen	0.075413	-0.006481	-0.116975	0.024319	0.052141	-0.122211	0.070235	
ZipCode	0.086007	-0.078690	-0.217903	-0.017074	-0.009513	0.020551	-0.040003	
Income	-0.002063	0.018719	0.123121	-0.006899	0.057273	0.027820	-0.034251	
ApprovalStatus	-0.028934	0.164086	0.206294	0.180583	0.188964	0.202158	-0.075558	

In [315…

```
sns.heatmap(df.corr(),
            cmap='bwr',
            vmin=0,
            vmax=1
           )
plt.show()
```

Out[315]: <AxesSubplot:>



- Looking at both the pair plot as well as the heatmap, we can roughly determine the variables that most affect the approval status. - Namely, Age, Industry, Outstanding Debt, Years Employed, Credit Score and

Income. - We can look at dropping the rest of the columns that does not seem to have a significant effect to the approval rate.

In [316…
```
df
```

Out[316]:

	Gender	Age	OutsDebt	Married	BankCustomer	Industry	Ethnicity	YearsEmployed	Prior
0	1	30.83	0.000	1	1	7	4	1.25	
1	0	58.67	4.460	1	1	9	1	3.04	
2	0	24.50	0.500	1	1	9	1	1.50	
3	1	27.83	1.540	1	1	7	4	3.75	
4	1	20.17	5.625	1	1	7	4	1.71	
...	...	...	...	...	...	...	...	...	
685	1	21.08	10.085	0	0	3	1	1.25	
686	0	22.67	0.750	1	1	4	4	2.00	
687	0	25.25	13.500	0	0	6	2	2.00	
688	1	17.92	0.205	1	1	2	4	0.04	
689	1	35.00	3.375	1	1	4	1	8.29	

690 rows × 16 columns

In [317…
```
df = df.drop(['Gender', 'BankCustomer', 'Married', 'DriversLicense', 'Citizen', 'Z
```

In [318…
```
df
```

Out[318]:

	Age	OutsDebt	Industry	YearsEmployed	PriorDefault	Employed	CreditScore	Income	Ap
0	30.83	0.000	7	1.25	1	1	1	0	
1	58.67	4.460	9	3.04	1	1	6	560	
2	24.50	0.500	9	1.50	1	0	0	824	
3	27.83	1.540	7	3.75	1	1	5	3	
4	20.17	5.625	7	1.71	1	0	0	0	
...	...	...	...	...	...	...	...	...	
685	21.08	10.085	3	1.25	0	0	0	0	
686	22.67	0.750	4	2.00	0	1	2	394	
687	25.25	13.500	6	2.00	0	1	1	1	
688	17.92	0.205	2	0.04	0	0	0	750	
689	35.00	3.375	4	8.29	0	0	0	0	

690 rows × 9 columns

How does Age affect Approval Status

Pulling out potential correlations between different variants from the pairplot earlier

In [319…

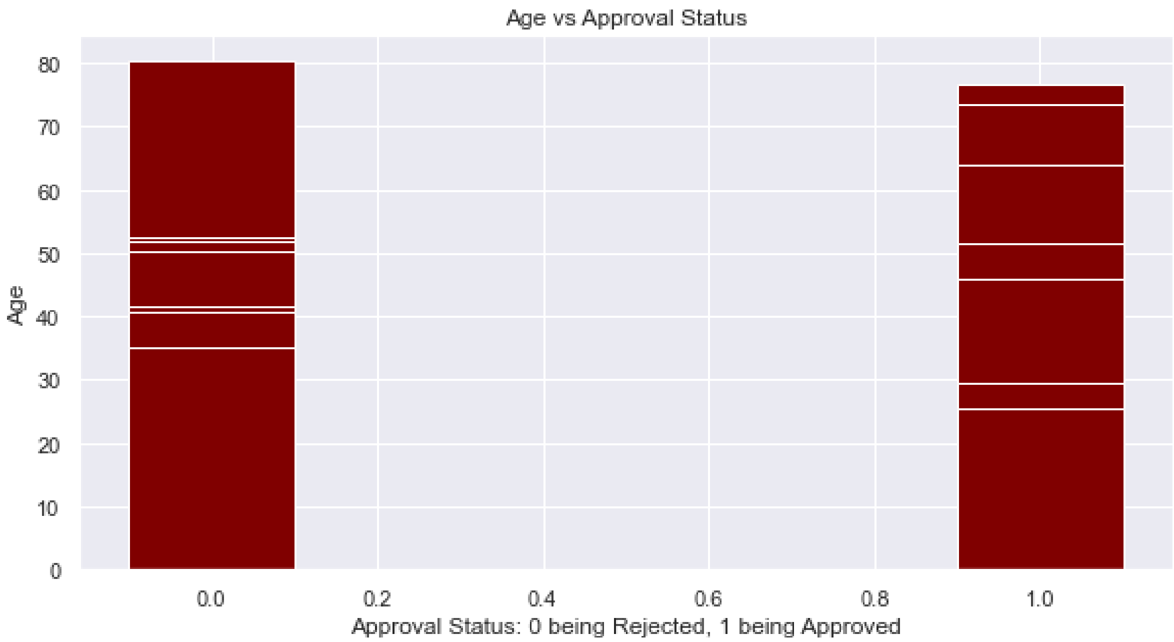
```
fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(x=df['ApprovalStatus'], height=df['Age'], color ='maroon',
        width = 0.2)

plt.xlabel('Approval Status: 0 being Rejected, 1 being Approved')
plt.ylabel('Age')
plt.title('Age vs Approval Status')
plt.show()
```

Out[319]: `<BarContainer object of 690 artists>`

Out[319]: `Text(0.5, 0, 'Approval Status: 0 being Rejected, 1 being Approved')`

Out[319]: `Text(0, 0.5, 'Age')`

Out[319]: `Text(0.5, 1.0, 'Age vs Approval Status')`



- From the bar chart, age does not seem to affect approval status

How does Credit Score affect Approval Status

In [320…

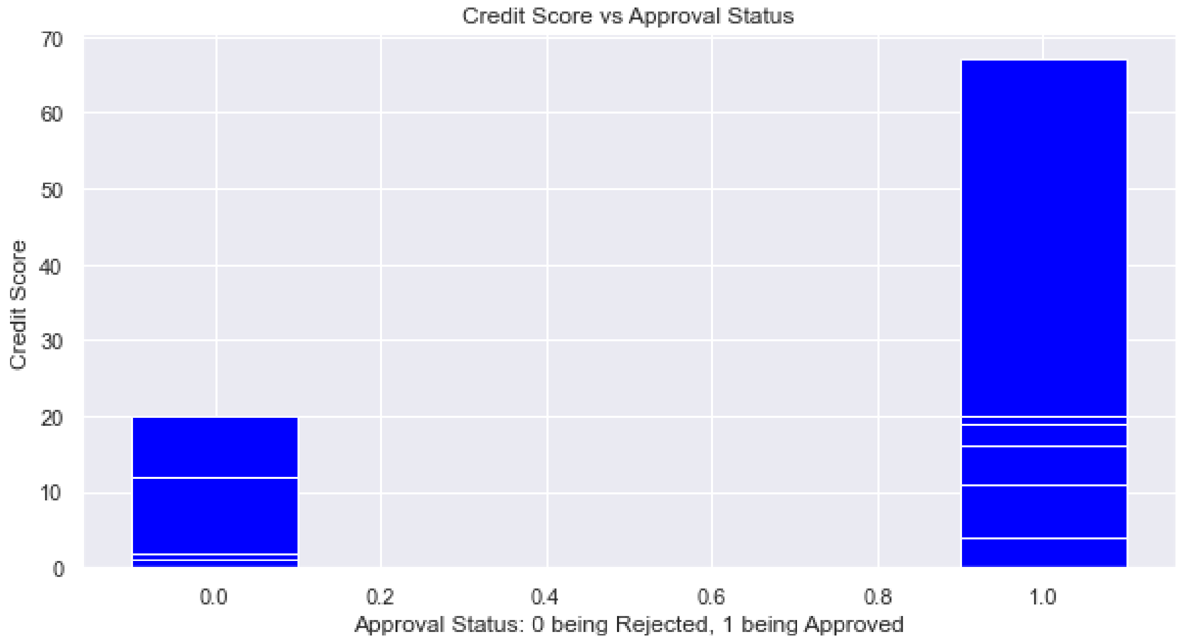
```
fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(x=df['ApprovalStatus'], height=df['CreditScore'], color ='blue',
        width = 0.2)

plt.xlabel('Approval Status: 0 being Rejected, 1 being Approved')
plt.ylabel('Credit Score')
plt.title('Credit Score vs Approval Status')
plt.show()
```

Out[320]: `<BarContainer object of 690 artists>`

Out[320]: `Text(0.5, 0, 'Approval Status: 0 being Rejected, 1 being Approved')`

Out[320]: `Text(0, 0.5, 'Credit Score')`

Out[320]: Text(0.5, 1.0, 'Credit Score vs Approval Status')



- We can see that when credit score is higher, , there is higher approval rate

How does Prior Default affect Approval Status

In [321…

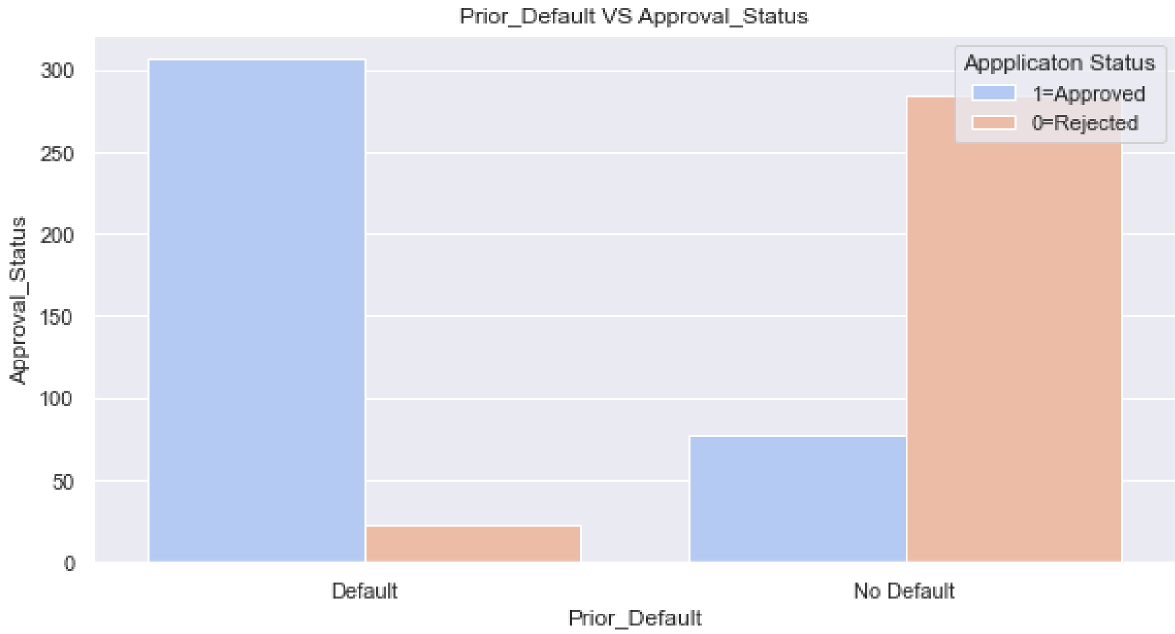
```
sns.set_theme(style='darkgrid')

fig = plt.subplots(figsize=(10,5))
PriorDefaultPlot = sns.countplot(data=df,x='PriorDefault', hue='ApprovalStatus', pa
PriorDefaultPlot.set(xlabel='Prior_Default', ylabel='Approval_Status')
PriorDefaultPlot.set_xticklabels(['Default', 'No Default'])
plt.legend(title='Appplicaton Status', labels=['1=Approved', '0=Rejected'], loc='up
plt.title('Prior_Default VS Approval_Status')
plt.show()
```

Out[321]: [Text(0.5, 0, 'Prior_Default'), Text(0, 0.5, 'Approval_Status')]

Out[321]: [Text(0, 0, 'Default'), Text(1, 0, 'No Default')]

Out[321]: <matplotlib.legend.Legend at 0x1a4b3c11f10>

Out[321]: Text(0.5, 1.0, 'Prior_Default VS Approval_Status')



In [ ]:
```
- When there is no default, there is higher approval rate and when there are defaul
```

How does Income affect Approval Status

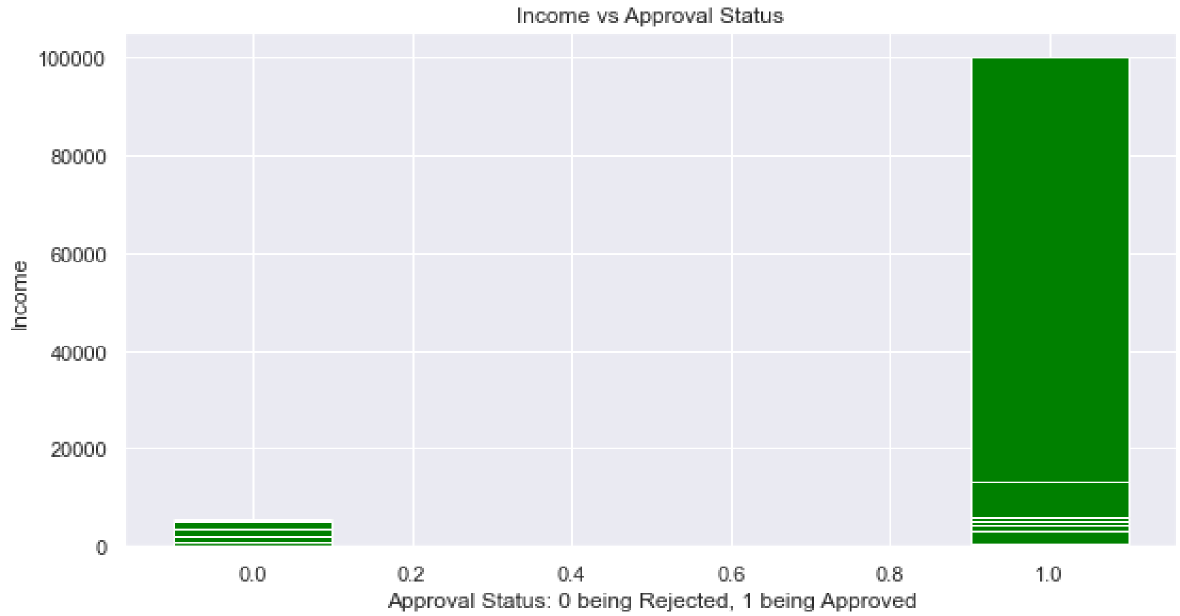
In [322…
```
fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(x=df['ApprovalStatus'], height=df['Income'], color ='green',
        width = 0.2)

plt.xlabel('Approval Status: 0 being Rejected, 1 being Approved')
plt.ylabel('Income')
plt.title('Income vs Approval Status')
plt.show()
```

Out[322]: <BarContainer object of 690 artists>

Out[322]: Text(0.5, 0, 'Approval Status: 0 being Rejected, 1 being Approved')

Out[322]: Text(0, 0.5, 'Income')

Out[322]: Text(0.5, 1.0, 'Income vs Approval Status')



- High income increases approval rate

How does being Employed affect Approval Status

In [323…

```
sns.set_theme(style='darkgrid')

fig = plt.subplots(figsize=(10,5))
EmployedPlot = sns.countplot(data=df,x='Employed', hue='ApprovalStatus', palette='
EmployedPlot.set(xlabel='Employed', ylabel='Approval_Status')
EmployedPlot.set_xticklabels(['Employed', 'Not Employed'])
plt.legend(title='Appplicaton Status', labels=['1=Approved', '0=Rejected'], loc='u
plt.title('Employed VS Approval_Status')
plt.show()
```

Out[323]:

```
[Text(0.5, 0, 'Employed'), Text(0, 0.5, 'Approval_Status')]
```

Out[323]:

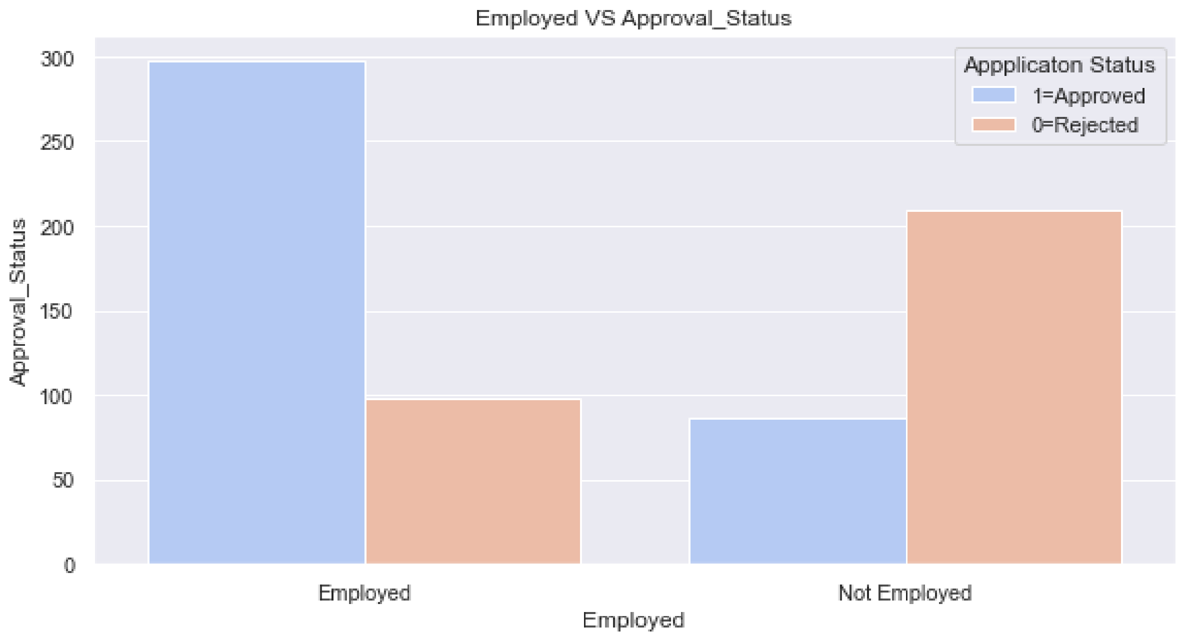
```
[Text(0, 0, 'Employed'), Text(1, 0, 'Not Employed')]
```

Out[323]:

```
<matplotlib.legend.Legend at 0x1a4b1a34fd0>
```

Out[323]:

```
Text(0.5, 1.0, 'Employed VS Approval_Status')
```



- Being employed increases the approval rate compared to not being employed

How does being Years Employed affect Approval Status

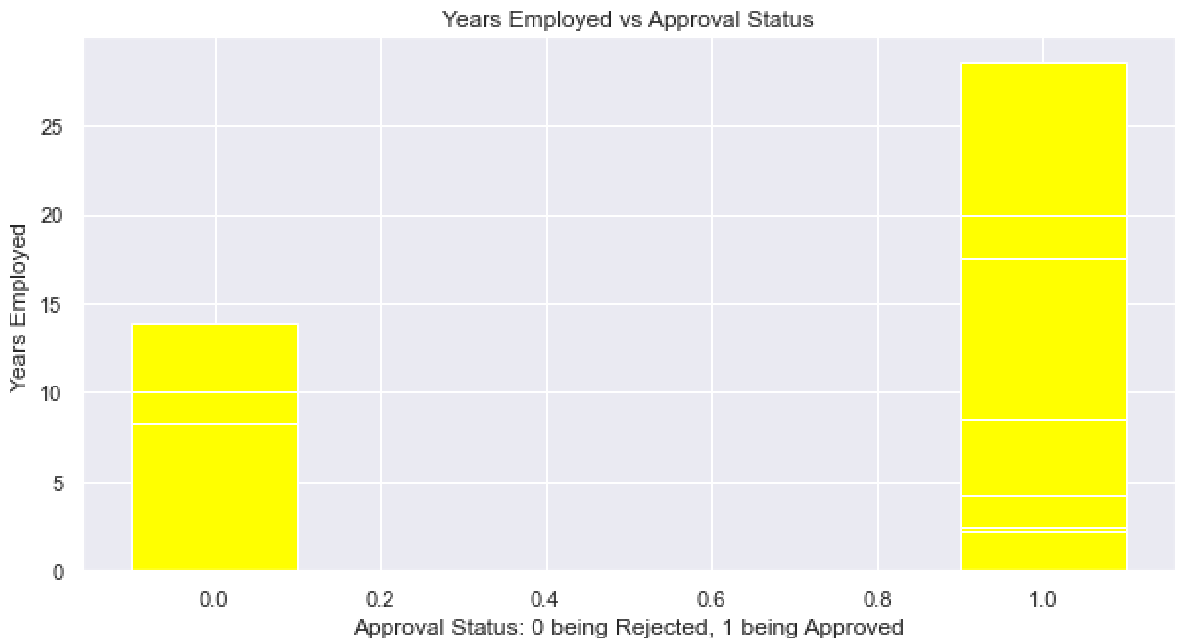
In [324...
```
fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(x=df['ApprovalStatus'], height=df['YearsEmployed'], color ='yellow',
        width = 0.2)

plt.xlabel('Approval Status: 0 being Rejected, 1 being Approved')
plt.ylabel('Years Employed')
plt.title('Years Employed vs Approval Status')
plt.show()
```

Out[324]: <BarContainer object of 690 artists>

Out[324]: Text(0.5, 0, 'Approval Status: 0 being Rejected, 1 being Approved')

Out[324]: Text(0, 0.5, 'Years Employed')

Out[324]: Text(0.5, 1.0, 'Years Employed vs Approval Status')



- Longer history of employment increases approval rate

How does being Industry affect Approval Status

In [292...
```
fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(x=df['ApprovalStatus'], height=df['Industry'], color ='pink',
        width = 0.2)

plt.xlabel('Approval Status: 0 being Rejected, 1 being Approved')
plt.ylabel('Industry')
plt.title('Industry vs Approval Status')
plt.show()
```

Out[292]: <BarContainer object of 690 artists>

Out[292]: Text(0.5, 0, 'Approval Status: 0 being Rejected, 1 being Approved')

Out[292]: Text(0, 0.5, 'Industry')

Out[292]: Text(0.5, 1.0, 'Industry vs Approval Status')

- Approval status does not seem to related to industry the person is working in

How does being Years Outstanding Debt affect Approval Status

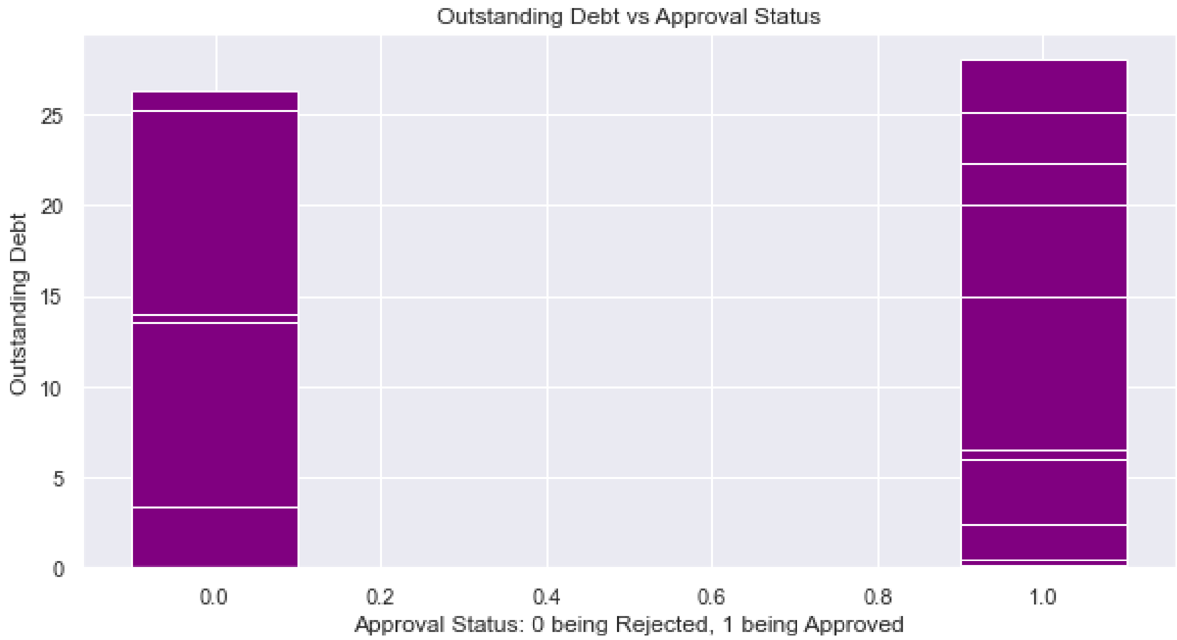
In [325…
```
fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(x=df['ApprovalStatus'], height=df['OutsDebt'], color ='purple',
        width = 0.2)

plt.xlabel('Approval Status: 0 being Rejected, 1 being Approved')
plt.ylabel('Outstanding Debt')
plt.title('Outstanding Debt vs Approval Status')
plt.show()
```

Out[325]: <BarContainer object of 690 artists>

Out[325]: Text(0.5, 0, 'Approval Status: 0 being Rejected, 1 being Approved')

Out[325]: Text(0, 0.5, 'Outstanding Debt')

Out[325]: Text(0.5, 1.0, 'Outstanding Debt vs Approval Status')



- Surprised that outstanding debt does not affect approval status much. The bank likely also take into account the timely repayment. If repayment is done on time, the amount of debt does not affect the status.

Further Clean the Data

Drop column that does not affect the Approval Status

In [326…

```
df = df.drop(['Industry', 'OutsDebt', 'Age'], axis = 1)
```

In [327…

```
df
```

Out[327]:

	YearsEmployed	PriorDefault	Employed	CreditScore	Income	ApprovalStatus
0	1.25	1	1	1	0	1
1	3.04	1	1	6	560	1
2	1.50	1	0	0	824	1
3	3.75	1	1	5	3	1
4	1.71	1	0	0	0	1
...	...	...	...	...	...	...
685	1.25	0	0	0	0	0
686	2.00	0	1	2	394	0
687	2.00	0	1	1	1	0
688	0.04	0	0	0	750	0
689	8.29	0	0	0	0	0

690 rows × 6 columns

Machine Learning

Determine the classifier with the highest accuracy

Knearest neighbors

In [328…

```
#import libraries for KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

In [329…

```
# y=df.ApprovalStatus
# X=df.drop('ApprovalStatus',axis=1)
```

In [334…

```
# STEP 1: split X and y into training and testing sets (using random_state for repı
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=99)

# STEP 2: train the model on the training set (using K=1)
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train, y_train) #first stage is always train, dont use test

# STEP 3: test the model on the testing set, and check the accuracy
y_pred_class = knn.predict(X_test)
print(accuracy_score(y_test, y_pred_class))
```

Out[334]:

```
KNeighborsClassifier(n_neighbors=1)

0.5606936416184971
```

Optimise hyperparameter

In [335…

```
# test with 50 neighbors
knn = KNeighborsClassifier(n_neighbors=50)
knn.fit(X_train, y_train)
y_pred_class = knn.predict(X_test)
print(accuracy_score(y_test, y_pred_class))
```

Out[335]:

```
KNeighborsClassifier(n_neighbors=50)

0.6705202312138728
```

In [338…

```
# test with 80 neighbors
knn = KNeighborsClassifier(n_neighbors=80)
knn.fit(X_train, y_train)
y_pred_class = knn.predict(X_test)
print(accuracy_score(y_test, y_pred_class))
```

Out[338]:

```
KNeighborsClassifier(n_neighbors=80)

0.6589595375722543
```

In [342…

```
# test with 46 neighbors
knn = KNeighborsClassifier(n_neighbors=46)
knn.fit(X_train, y_train)
y_pred_class = knn.predict(X_test)
print(accuracy_score(y_test, y_pred_class))
```

Out[342]:

```
KNeighborsClassifier(n_neighbors=46)

0.653179190751445
```

Highest Accuracy using n_neighbors=50: ~65.31%

Logistic Regression

In [343…

```
df_standardized = (df-df.mean())/df.std()
df_standardized.mean()
df_standardized.std()
```

Out[343]:

```
YearsEmployed     1.879334e-16
PriorDefault     -5.399867e-16
Employed         -7.852012e-16
CreditScore       2.067589e-16
Income           -1.448117e-18
ApprovalStatus   -5.766724e-16
dtype: float64
```

Out[343]:

```
YearsEmployed     1.0
PriorDefault      1.0
Employed          1.0
CreditScore       1.0
Income            1.0
ApprovalStatus    1.0
dtype: float64
```

In [348…

```
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

logreg = LogisticRegression()

X = df.drop('ApprovalStatus',axis=1)

y = df.ApprovalStatus

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

print('Accuracy: ', accuracy_score(y_test, y_pred))
```

```
C:\Users\alisonc\Documents\DataScience\lib\site-packages\sklearn\linear_model\_log
istic.py:814: ConvergenceWarning: 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 in:
    https://scikit-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-regression
  n_iter_i = _check_optimize_result(
```

Out[348]:

```
LogisticRegression()

Accuracy:  0.8728323699421965
```

Accuracy: ~87.28%

Decision Tree

In [349…

```
X = df.drop('ApprovalStatus',axis=1)

y = df.ApprovalStatus
```

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

#still using the same train size 80%, test size 20%
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=2, test_size=

#scaling the features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.fit_transform(X_test)

dt.fit(scaled_X_train,y_train)

y_preds = dt.predict(scaled_X_test)

accuracy_score(y_test,y_preds)
```

Out[349]:
```
DecisionTreeClassifier()
```

Out[349]:
```
0.8043478260869565
```

Accuracy: ~80.43%

Summary

- KNN, Logistic Regression and Decision Tree were applied to predict a new customer's credit card approval rate. - Among all 3 classifiers, Logistic Regression's accuracy is the highest. - I feel like i might have dropped to many columns while doing EDA. I should consider exploring different chart types to plot different types of variants to make a better decision.