Credit card approval Prediction

Importing libraries

In [1]:
```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

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

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor


%matplotlib inline
```

Reading Exploring Dataset

In [3]:
```
data = pd.read_csv('data3.csv')
```

In [4]:
```
# shape
print(data.shape)
```

```
(690, 16)
```

In [5]:
```
data.dtypes
```

Out[5]:
```
Gender             object
Age               float64
Debt              float64
Married            object
BankCustomer       object
EducationLevel     object
Ethnicity          object
YearsEmployed     float64
PriorDefault       object
Employed           object
CreditScore         int64
DriversLicense     object
Citizen            object
ZipCode            object
Income              int64
Approved           object
dtype: object
```

In [6]:
```
# descriptions
print(data.describe())
```

```
              Age        Debt  YearsEmployed  CreditScore         Income
count  690.000000  690.000000     690.000000    690.00000     690.000000
mean    31.568171    4.758725       2.223406      2.40000    1017.385507
std     11.853273    4.978163       3.346513      4.86294    5210.102598
min     13.750000    0.000000       0.000000      0.00000       0.000000
25%     22.670000    1.000000       0.165000      0.00000       0.000000
50%     28.625000    2.750000       1.000000      0.00000       5.000000
75%     37.707500    7.207500       2.625000      3.00000     395.500000
max     80.250000   28.000000      28.500000     67.00000  100000.000000
```

In [7]:
```
# Checking for missing values
data.isna().sum()
```

Out[7]:
```
Gender            0
Age               0
Debt              0
Married           0
BankCustomer      0
EducationLevel    0
Ethnicity         0
YearsEmployed     0
PriorDefault      0
Employed          0
CreditScore       0
DriversLicense    0
Citizen           0
ZipCode           0
Income            0
Approved          0
dtype: int64
```

In [8]:
```
# Checking for missing values another method
pd.isnull(data).any()
```

Out[8]:
```
Gender            False
Age               False
Debt              False
Married           False
BankCustomer      False
EducationLevel    False
Ethnicity         False
YearsEmployed     False
PriorDefault      False
Employed          False
CreditScore       False
DriversLicense    False
Citizen           False
ZipCode           False
Income            False
Approved          False
dtype: bool
```

In [12]:
```
#counting sex , here male=a and female = b
data['Gender'].value_counts()
```

Out[12]:
```
b    480
a    210
Name: Gender, dtype: int64
```

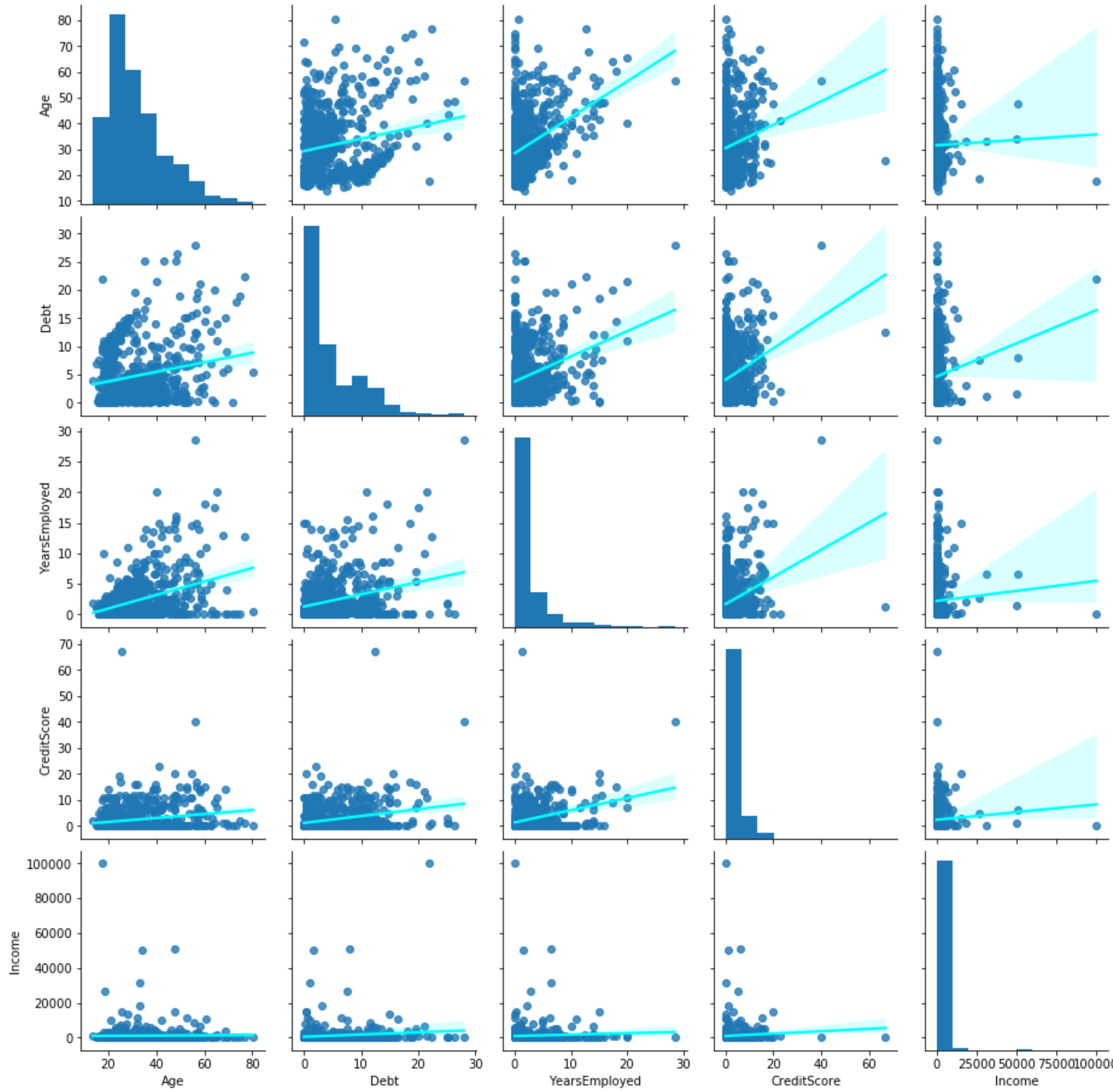
In [16]:
```
# count by presence of Approved  , here + for yes and - for no
data['Approved'].value_counts()

#print(data.groupby('target').size())   #this is another method
```

Out[16]:
```
-    383
+    307
Name: Approved, dtype: int64
```

Initial visualization

In [14]:

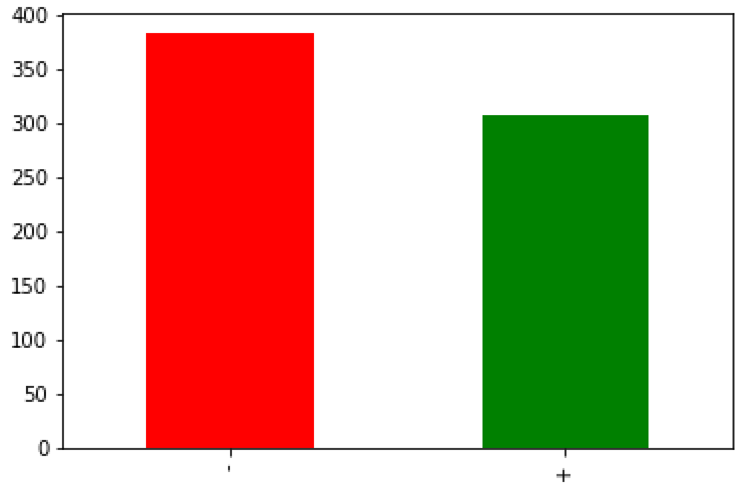
```
%%time
sns.pairplot(data, kind='reg', plot_kws={'line_kws':{'color': 'cyan'}})
plt.show()
```




```
Wall time: 8.01 s
```

In [17]:
```
data.Approved.value_counts().plot(kind="bar", color=["red", "green"])
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x17e153398c8>



In [18]:
```
data.dtypes
```

Out[18]:
```
Gender             object
Age               float64
Debt              float64
Married            object
BankCustomer       object
EducationLevel     object
Ethnicity          object
YearsEmployed     float64
PriorDefault       object
Employed           object
CreditScore         int64
DriversLicense     object
Citizen            object
ZipCode            object
Income              int64
Approved           object
dtype: object
```

In [19]:

```
# Import LabelEncoder
from sklearn.preprocessing import LabelEncoder
# Instantiate LabelEncoder
le=LabelEncoder()

# Iterate over all the values of each column and extract their dtypes
for col in data.columns:
    # Compare if the dtype is object
    if data[col].dtypes=='object':
    # Use LabelEncoder to do the numeric transformation
        data[col]=le.fit_transform(data[col])
data.info()
```

```
<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    int32  
 1   Age             690 non-null    float64
 2   Debt            690 non-null    float64
 3   Married         690 non-null    int32  
 4   BankCustomer    690 non-null    int32  
 5   EducationLevel  690 non-null    int32  
 6   Ethnicity       690 non-null    int32  
 7   YearsEmployed   690 non-null    float64
 8   PriorDefault    690 non-null    int32  
 9   Employed        690 non-null    int32  
 10  CreditScore     690 non-null    int64  
 11  DriversLicense  690 non-null    int32  
 12  Citizen         690 non-null    int32  
 13  ZipCode         690 non-null    int32  
 14  Income          690 non-null    int64  
 15  Approved        690 non-null    int32  
dtypes: float64(3), int32(11), int64(2)
memory usage: 56.7 KB
```

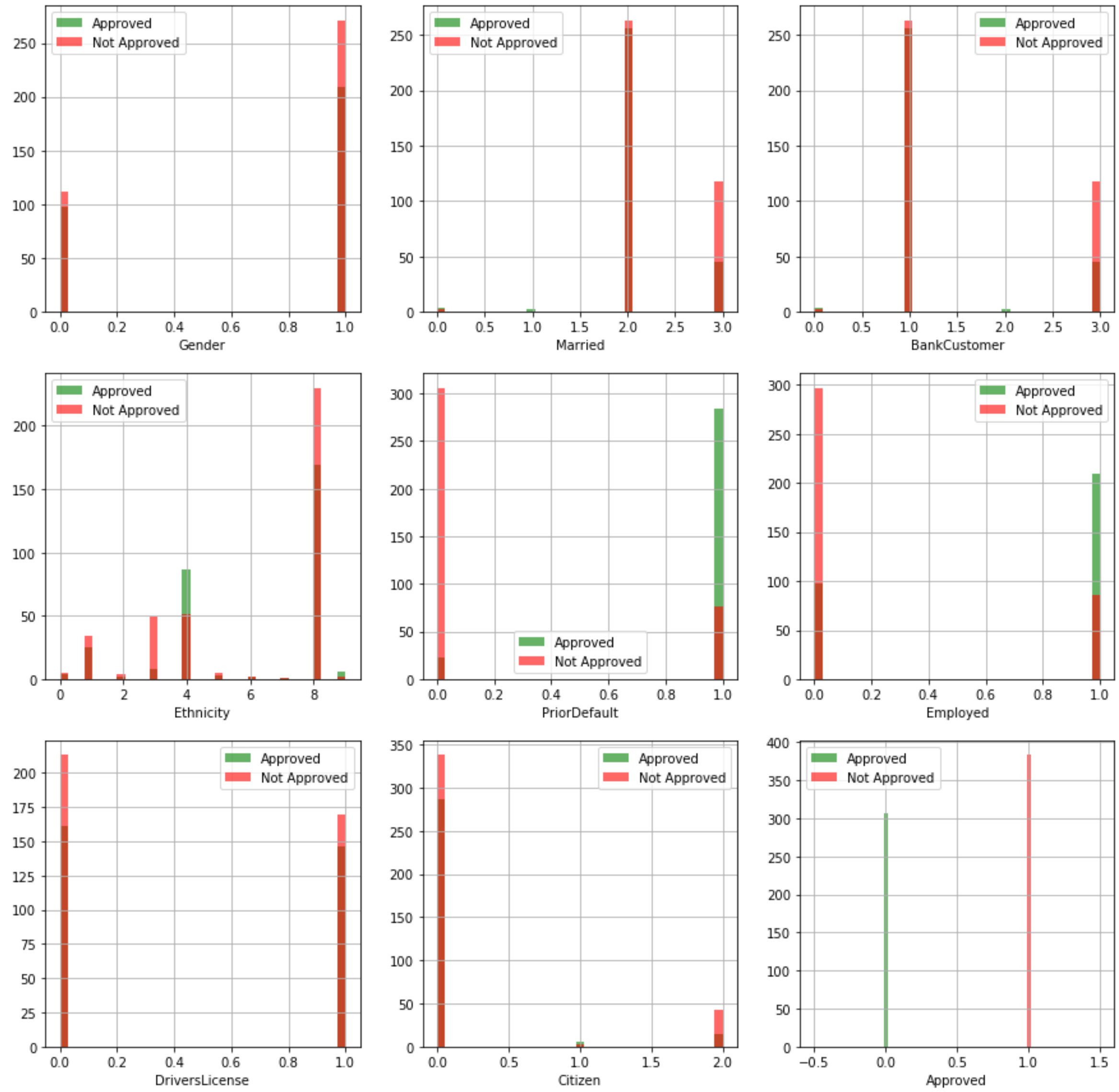
In [20]:
```
categorical_val = []
continous_val = []
for column in data.columns:
    print('==============================')
    print(f"{column} : {data[column].unique()}")
    if len(data[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

```
==============================
PriorDefault : [1 0]
==============================
Employed : [1 0]
==============================
CreditScore : [ 1  6  0  5  7 10  3 17  2  9  8 15 11 12 40 23  4 20 67
14 16 13 19]
==============================
DriversLicense : [0 1]
==============================
Citizen : [0 2 1]
==============================
ZipCode : [ 42 118  74   1   8  96  25 159  34 140  10  67   0  86 108 1
67  40  80
  18 136  27 121 146  79  61 152 127  83  49 134 111  59  23 153  62 141
 138 116 170 168 123  15 166  97  75 164  37   5  30  69  26 156  20  32
  87  48 113  71 102 112  93  43 110  35 109 122 165 147 107 106  45  76
   2 119  99  60  39  11 137  38  17 103 133 135 104  19   7 144  47  56
  22  50  55 139  64 132  51 162 155   9 130  14  13  78  21  70  90   3
 154 125  58  29   6 114 128  91 131 150 101  72  52  94  16  33 148 143
```

In [21]:
```
categorical_val
```

Out[21]:
```
['Gender',
 'Married',
 'BankCustomer',
 'Ethnicity',
 'PriorDefault',
 'Employed',
 'DriversLicense',
 'Citizen',
 'Approved']
```

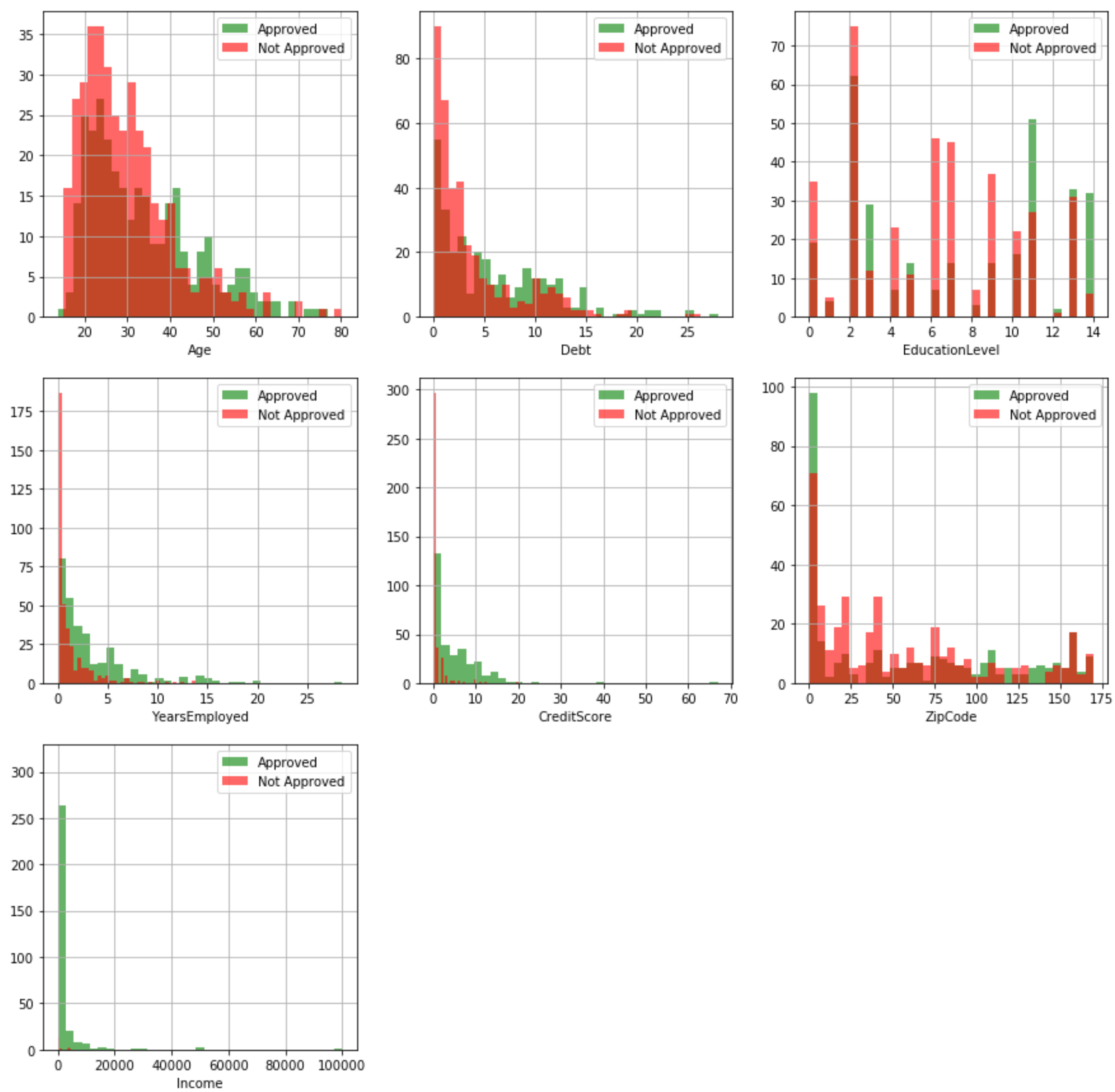
In [23]:
```
plt.figure(figsize=(15, 15))

for i, column in enumerate(categorical_val, 1):
    plt.subplot(3, 3, i)
    data[data["Approved"] == 0][column].hist(bins=35, color='green', label='A
    data[data["Approved"] == 1][column].hist(bins=35, color='red', label='Not
    plt.legend()
    plt.xlabel(column)
```



In [24]:
```
continous_val
```

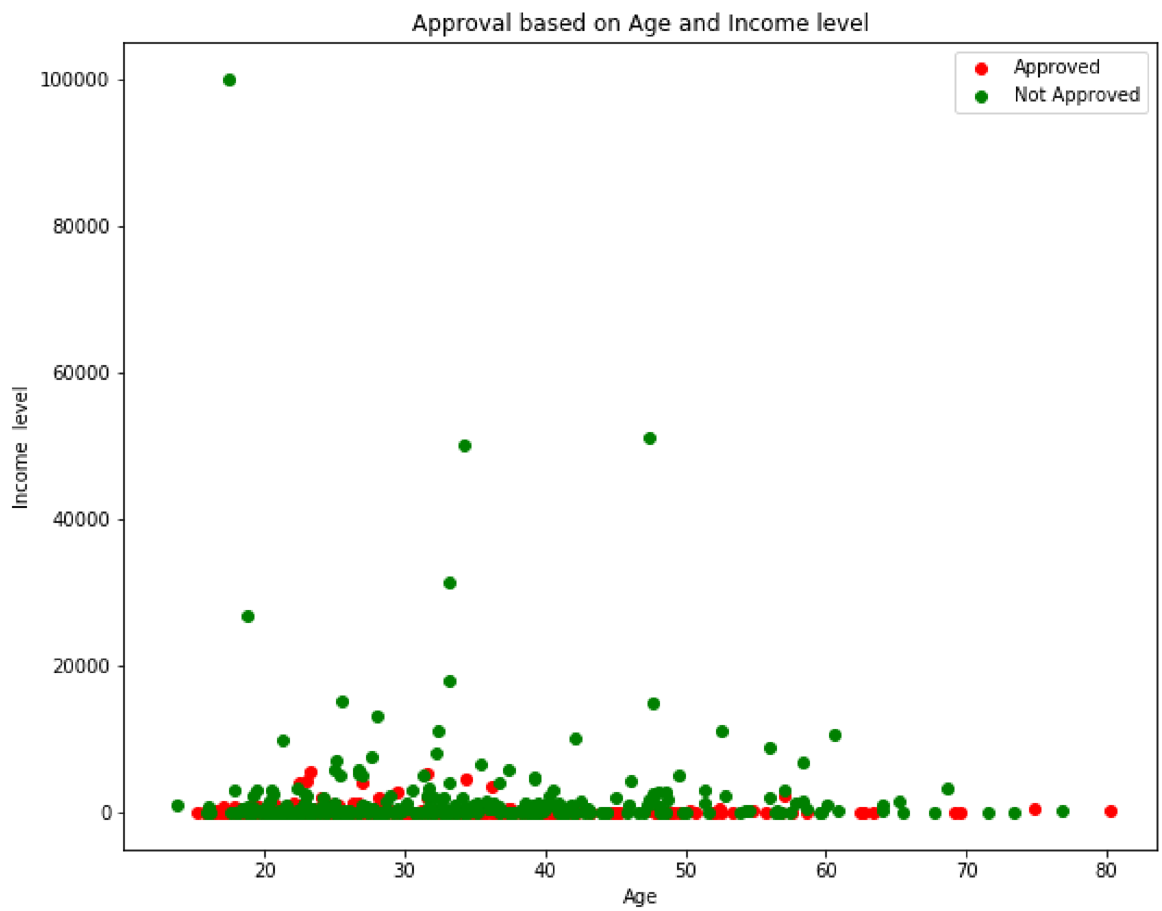
Out[24]:
```
['Age',
 'Debt',
 'EducationLevel',
 'YearsEmployed',
 'CreditScore',
 'ZipCode',
 'Income']
```

In [25]:
```
plt.figure(figsize=(15, 15))

for i, column in enumerate(continous_val, 1):
    plt.subplot(3, 3, i)
    data[data["Approved"] == 0][column].hist(bins=35, color='green', label='A
    data[data["Approved"] == 1][column].hist(bins=35, color='red', label='Not
    plt.legend()
    plt.xlabel(column)
```



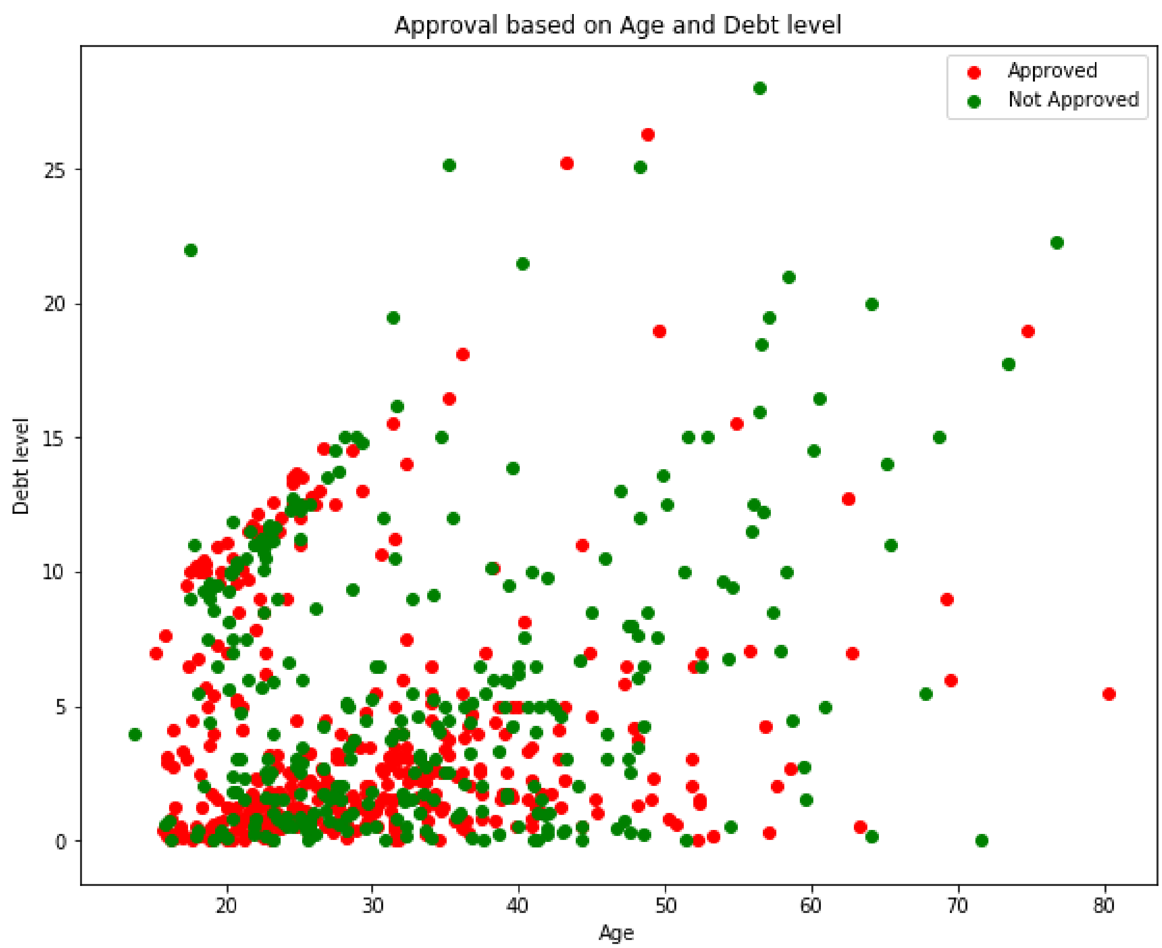
In [26]:

```
# Create another figure
plt.figure(figsize=(10, 8))

# Scatter with postivie examples
plt.scatter(data.Age[data.Approved==1],
            data.Income[data.Approved==1],
            c="red")

# Scatter with negative examples
plt.scatter(data.Age[data.Approved==0],
            data.Income[data.Approved==0],
            c="green")
# Add some helpful info
plt.title("Approval based on Age and Income level")
plt.xlabel("Age")
plt.ylabel("Income  level")
plt.legend(["Approved", "Not Approved"]);
```



In [27]:

```
# Create another figure
plt.figure(figsize=(10, 8))

# Scatter with postivie examples
plt.scatter(data.Age[data.Approved==1],
            data.Debt[data.Approved==1],
            c="red")

# Scatter with negative examples
plt.scatter(data.Age[data.Approved==0],
            data.Debt[data.Approved==0],
            c="green")
# Add some helpful info
plt.title("Approval based on Age and Debt level")
plt.xlabel("Age")
plt.ylabel("Debt level")
plt.legend(["Approved", "Not Approved"]);
```



In [28]:

```
data.corr() # Pearson Correlation Coefficients
```

Out[28]:

	Gender	Age	Debt	Married	BankCustomer	EducationLevel	Et
Gender	1.000000	0.035604	-0.041746	0.068365	0.069627	-0.012486	0.
Age	0.035604	1.000000	0.201316	-0.092745	-0.104618	0.014077	-0.
Debt	-0.041746	0.201316	1.000000	-0.047608	-0.068773	0.023428	-0.
Married	0.068365	-0.092745	-0.047608	1.000000	0.942463	0.001738	0.
BankCustomer	0.069627	-0.104618	-0.068773	0.942463	1.000000	-0.041508	-0.
EducationLevel	-0.012486	0.014077	0.023428	0.001738	-0.041508	1.000000	0.
Ethnicity	0.044686	-0.194310	-0.016451	0.063158	-0.003989	0.038218	1.
YearsEmployed	0.086544	0.392787	0.298902	-0.048423	-0.065497	0.041492	-0.
PriorDefault	-0.026047	0.204342	0.244317	-0.078851	-0.129863	0.107793	-0.
Employed	-0.077784	0.083681	0.174846	-0.114926	-0.162464	0.132133	0.
CreditScore	-0.024630	0.185575	0.271207	-0.077948	-0.106457	0.012271	-0.
DriversLicense	0.051674	0.054778	-0.013023	0.029057	0.015342	0.075946	0.
Citizen	0.085488	-0.014584	-0.122233	-0.094585	-0.036095	-0.010663	-0.
ZipCode	0.018101	-0.092842	-0.100937	-0.120559	-0.062667	0.053078	-0.
Income	-0.002063	0.018539	0.123121	-0.101102	-0.022904	0.007381	-0.
Approved	0.028934	-0.161627	-0.206294	0.191431	0.187520	-0.130026	-0.

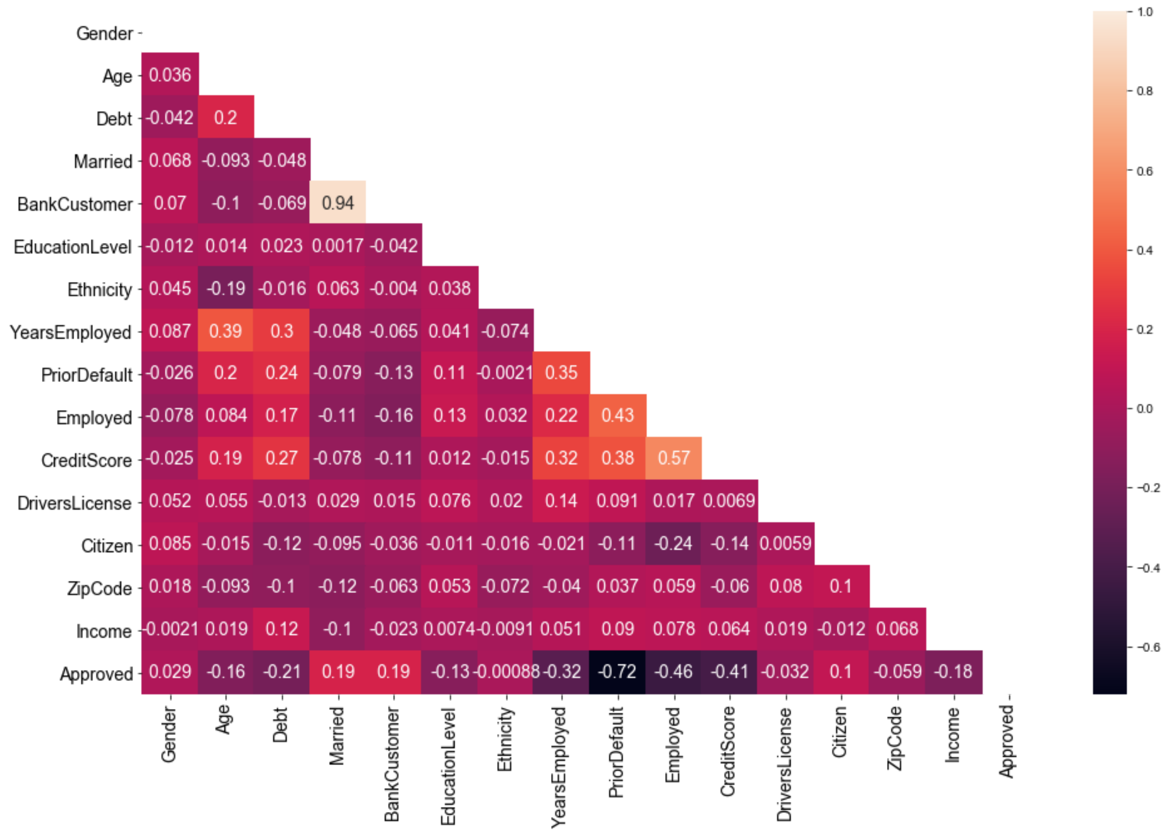
In [29]:

```
mask = np.zeros_like(data.corr())
triangle_indices = np.triu_indices_from(mask)
mask[triangle_indices] = True
mask
```

Out[29]:

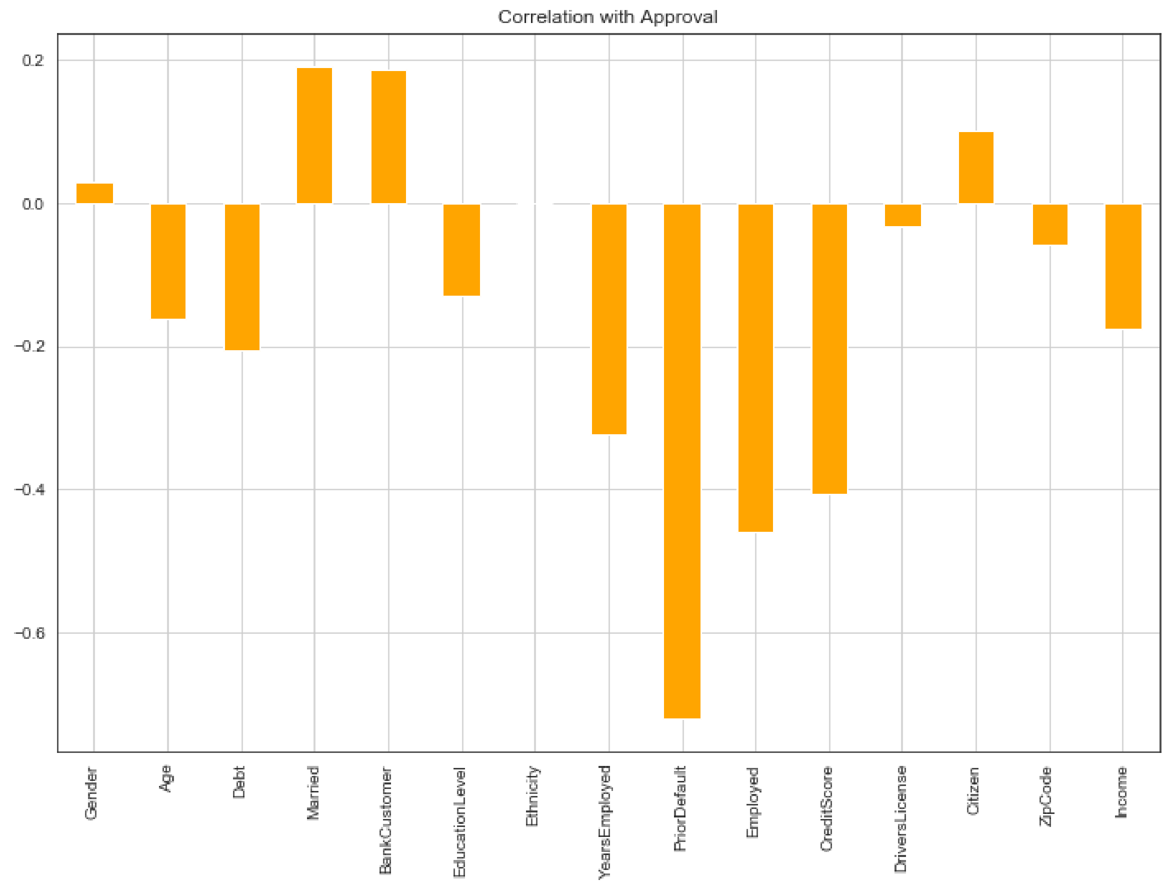
```
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

In [30]:
```
plt.figure(figsize=(16,10))
sns.heatmap(data.corr(), mask=mask, annot=True, annot_kws={"size": 14})
sns.set_style('white')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



In [31]:
```
data.drop('Approved', axis=1).corrwith(data.Approved).plot(kind='bar', grid=T
                                                            title="Correlation with Ap
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x17e139ae608>



In [32]:
```
categorical_val.remove('Approved')
dataset = pd.get_dummies(data, columns = categorical_val)
```

In [33]:
```
dataset.head()
```

Out[33]:

	Age	Debt	EducationLevel	YearsEmployed	CreditScore	ZipCode	Income	Approved	G
0	30.83	0.000	13	1.25	1	42	0	0	
1	58.67	4.460	11	3.04	6	118	560	0	
2	24.50	0.500	11	1.50	0	74	824	0	
3	27.83	1.540	13	3.75	5	1	3	0	
4	20.17	5.625	13	1.71	0	8	0	0	

5 rows × 37 columns

In [34]:
```
print(data.columns)
print(dataset.columns)
```

```
Index(['Gender', 'Age', 'Debt', 'Married', 'BankCustomer', 'EducationLeve
l',
       'Ethnicity', 'YearsEmployed', 'PriorDefault', 'Employed', 'CreditSco
re',
       'DriversLicense', 'Citizen', 'ZipCode', 'Income', 'Approved'],
      dtype='object')
Index(['Age', 'Debt', 'EducationLevel', 'YearsEmployed', 'CreditScore',
       'ZipCode', 'Income', 'Approved', 'Gender_0', 'Gender_1', 'Married_
0',
       'Married_1', 'Married_2', 'Married_3', 'BankCustomer_0',
       'BankCustomer_1', 'BankCustomer_2', 'BankCustomer_3', 'Ethnicity_0',
       'Ethnicity_1', 'Ethnicity_2', 'Ethnicity_3', 'Ethnicity_4',
       'Ethnicity_5', 'Ethnicity_6', 'Ethnicity_7', 'Ethnicity_8',
       'Ethnicity_9', 'PriorDefault_0', 'PriorDefault_1', 'Employed_0',
       'Employed_1', 'DriversLicense_0', 'DriversLicense_1', 'Citizen_0',
       'Citizen_1', 'Citizen_2'],
      dtype='object')
```

In [35]:
```
dataset.describe()
```

Out[35]:

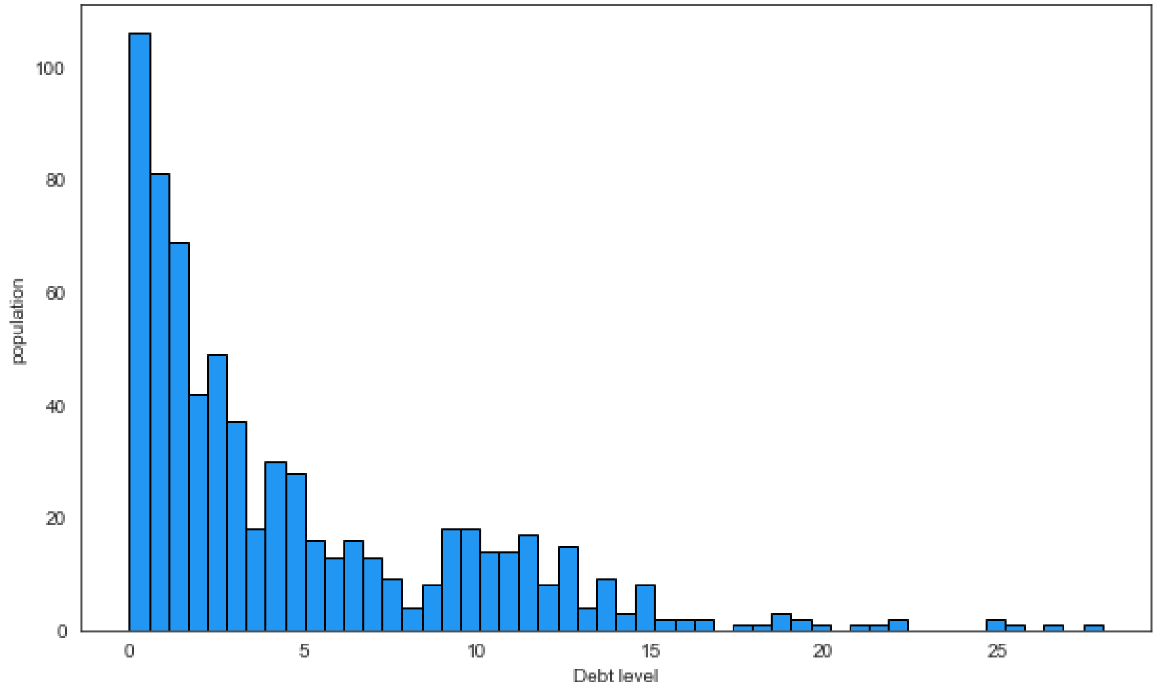
	Age	Debt	EducationLevel	YearsEmployed	CreditScore	ZipCode	
count	690.000000	690.000000	690.000000	690.000000	690.00000	690.000000	6
mean	31.568171	4.758725	6.607246	2.223406	2.40000	56.927536	10
std	11.853273	4.978163	4.412110	3.346513	4.86294	54.813265	52
min	13.750000	0.000000	0.000000	0.000000	0.00000	0.000000	
25%	22.670000	1.000000	2.000000	0.165000	0.00000	7.250000	
50%	28.625000	2.750000	6.000000	1.000000	0.00000	40.000000	
75%	37.707500	7.207500	11.000000	2.625000	3.00000	95.750000	3
max	80.250000	28.000000	14.000000	28.500000	67.00000	170.000000	1000

8 rows × 37 columns

Visualising Data - Histograms, Distributions and Bar Charts

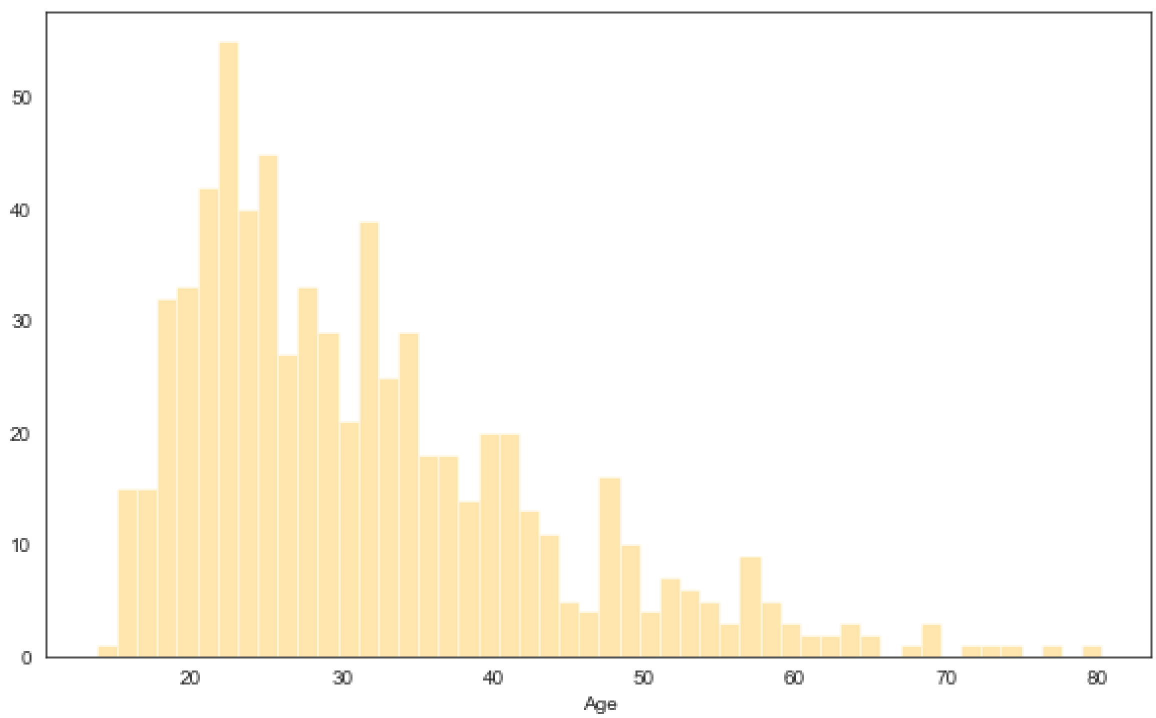
In [36]:

```
plt.figure(figsize=(10, 6))
plt.hist(dataset['Debt'], bins=50, ec='black', color='#2196f3')
plt.xlabel('Debt level')
plt.ylabel('population')
plt.show()
```



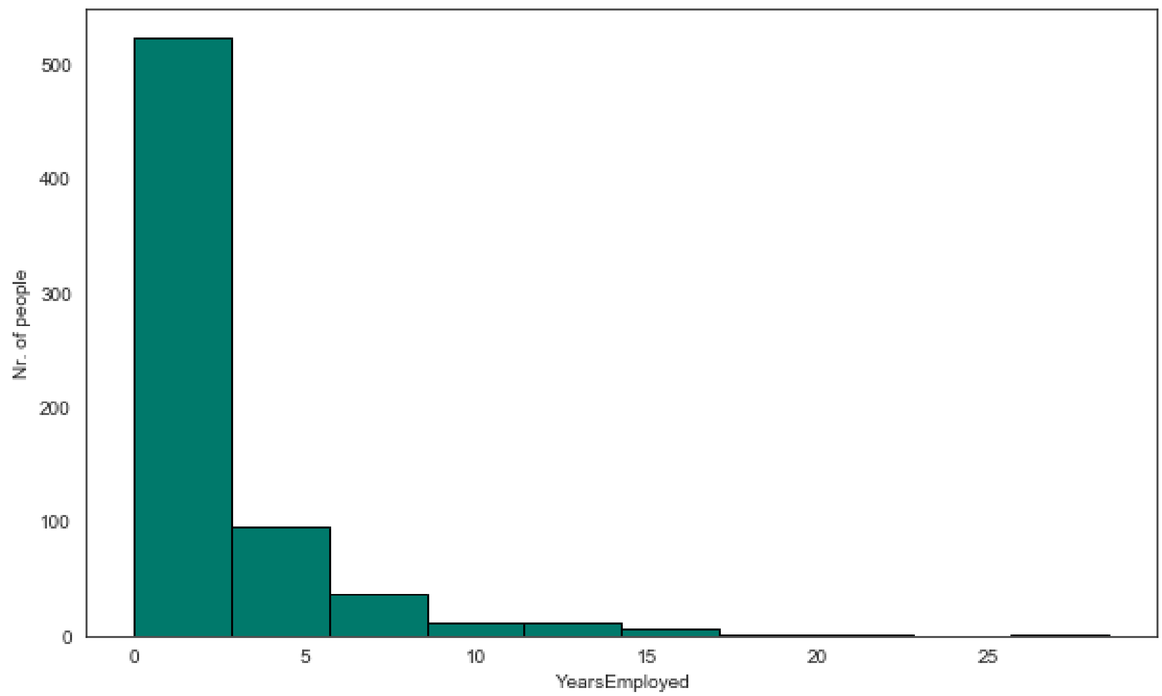
In [37]:

```
plt.figure(figsize=(10, 6))
sns.distplot(dataset['Age'], bins=50, hist=True, kde=False, color='#fbc02d')
plt.show()
```



In [38]:

```
plt.figure(figsize=(10, 6))
plt.hist(dataset['YearsEmployed'], ec='black', color='#00796b')
plt.xlabel('YearsEmployed')
plt.ylabel('Nr. of people')
plt.show()
```



In [39]:

```
from sklearn.preprocessing import StandardScaler

s_sc = StandardScaler()
col_to_scale = ['Age','Debt','EducationLevel','YearsEmployed','CreditScore','
dataset[col_to_scale] = s_sc.fit_transform(dataset[col_to_scale])
```

In [40]:

```
dataset.head()
```

Out[40]:

	Age	Debt	EducationLevel	YearsEmployed	CreditScore	ZipCode	Income	Ap
0	-0.062321	-0.956613	1.449962	-0.291083	-0.288101	-0.272532	-0.195413	
1	2.288101	-0.060051	0.996335	0.244190	0.740830	1.115000	-0.087852	
2	-0.596738	-0.856102	0.996335	-0.216324	-0.493887	0.311692	-0.037144	
3	-0.315599	-0.647038	1.449962	0.456505	0.535044	-1.021069	-0.194837	
4	-0.962303	0.174141	1.449962	-0.153526	-0.493887	-0.893270	-0.195413	

5 rows × 37 columns

In [41]:
```
dataset.describe()
```

Out[41]:

	Age	Debt	EducationLevel	YearsEmployed	CreditScore	ZipC
count	6.900000e+02	6.900000e+02	6.900000e+02	6.900000e+02	6.900000e+02	6.900000e
mean	-4.158509e-16	1.605801e-16	-1.010464e-16	1.673380e-16	1.153667e-15	-2.606611e
std	1.000725e+00	1.000725e+00	1.000725e+00	1.000725e+00	1.000725e+00	1.000725e
min	-1.504318e+00	-9.566132e-01	-1.498612e+00	-6.648767e-01	-4.938866e-01	-1.039326e
25%	-7.512378e-01	-7.555902e-01	-1.044985e+00	-6.155359e-01	-4.938866e-01	-9.069625e
50%	-2.484804e-01	-4.037999e-01	-1.377316e-01	-3.658414e-01	-4.938866e-01	-3.090459e
75%	5.183195e-01	4.922602e-01	9.963352e-01	1.200908e-01	1.234717e-01	7.087815e
max	4.110016e+00	4.672031e+00	1.676775e+00	7.857628e+00	1.329378e+01	2.064363e

8 rows × 37 columns

In [42]:

```
dataset.corr() # Pearson Correlation Coefficients
```

Out[42]:

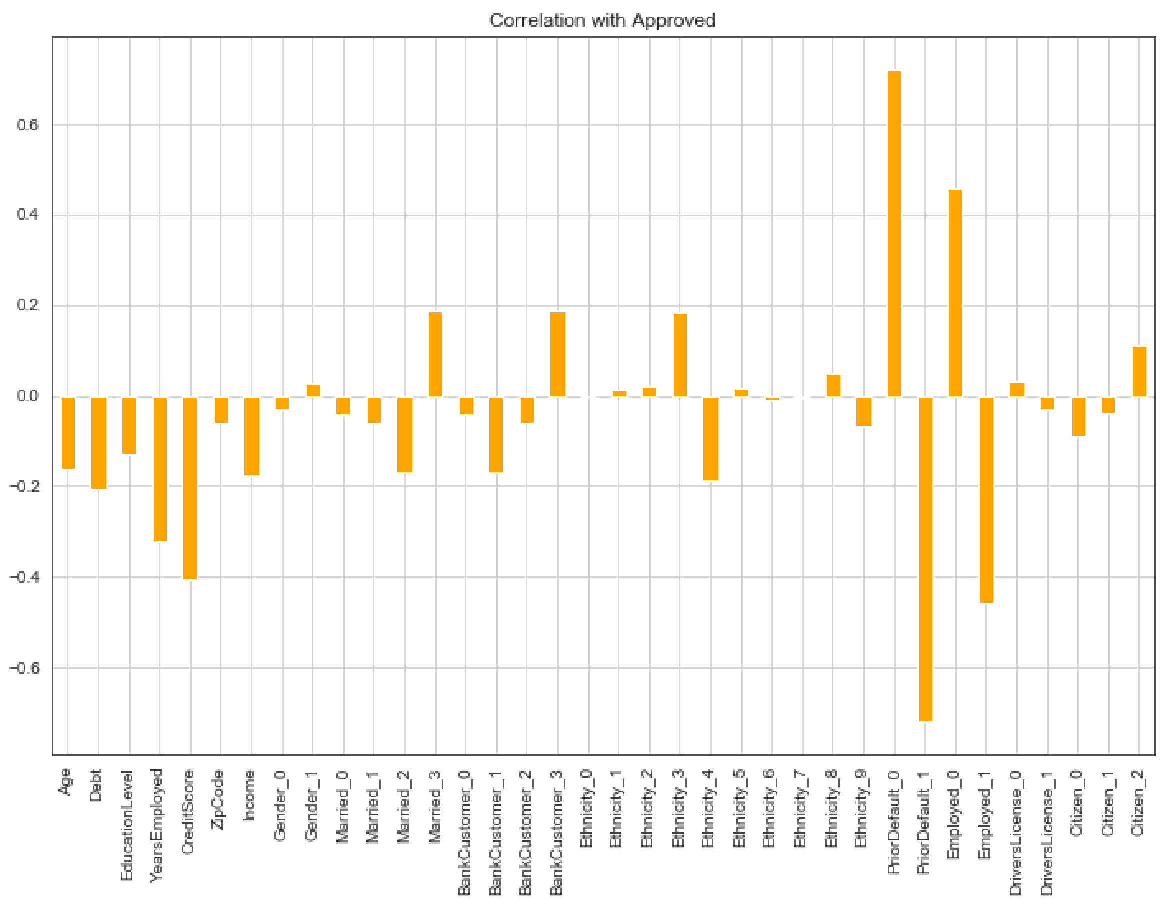
	Age	Debt	EducationLevel	YearsEmployed	CreditScore	ZipCode
Age	1.000000	0.201316	0.014077	0.392787	0.185575	-0.092842
Debt	0.201316	1.000000	0.023428	0.298902	0.271207	-0.100937
EducationLevel	0.014077	0.023428	1.000000	0.041492	0.012271	0.053078
YearsEmployed	0.392787	0.298902	0.041492	1.000000	0.322330	-0.039743
CreditScore	0.185575	0.271207	0.012271	0.322330	1.000000	-0.060216
ZipCode	-0.092842	-0.100937	0.053078	-0.039743	-0.060216	1.000000
Income	0.018539	0.123121	0.007381	0.051345	0.063692	0.067609
Approved	-0.161627	-0.206294	-0.130026	-0.322475	-0.406410	-0.058633
Gender_0	-0.035604	0.041746	0.012486	-0.086544	0.024630	-0.018101
Gender_1	0.035604	-0.041746	-0.012486	0.086544	-0.024630	0.018101
Married_0	0.030971	-0.089595	-0.119115	-0.062271	-0.046257	0.193345
Married_1	-0.062718	0.069678	-0.025769	0.044767	-0.026629	0.044859
Married_2	0.098048	0.093017	0.081342	0.082493	0.122543	-0.002782
Married_3	-0.098492	-0.083781	-0.053383	-0.075905	-0.111077	-0.045111
BankCustomer_0	0.030971	-0.089595	-0.119115	-0.062271	-0.046257	0.193345
BankCustomer_1	0.098048	0.093017	0.081342	0.082493	0.122543	-0.002782
BankCustomer_2	-0.062718	0.069678	-0.025769	0.044767	-0.026629	0.044859
BankCustomer_3	-0.098492	-0.083781	-0.053383	-0.075905	-0.111077	-0.045111
Ethnicity_0	0.083896	-0.084398	-0.146206	-0.074047	-0.056777	0.159896
Ethnicity_1	0.125157	-0.003667	-0.059749	0.074188	0.032423	0.022546
Ethnicity_2	-0.064904	0.010693	-0.034143	-0.046331	-0.023771	-0.001016
Ethnicity_3	0.166328	0.037302	-0.032972	-0.073321	-0.033368	-0.023343
Ethnicity_4	0.017277	0.061269	0.124254	0.178414	0.065464	0.032347
Ethnicity_5	-0.012042	-0.033528	0.015789	-0.062901	-0.020059	-0.073018
Ethnicity_6	-0.054689	-0.002532	0.045771	-0.009126	0.005500	-0.052875
Ethnicity_7	0.005767	0.074880	-0.001312	-0.033147	-0.026629	0.005485
Ethnicity_8	-0.226782	-0.095540	-0.010178	-0.143370	-0.049884	-0.018489
Ethnicity_9	0.242260	0.203351	-0.039482	0.194173	0.096952	-0.095264
PriorDefault_0	-0.204342	-0.244317	-0.107793	-0.345689	-0.379532	-0.036880
PriorDefault_1	0.204342	0.244317	0.107793	0.345689	0.379532	0.036880
Employed_0	-0.083681	-0.174846	-0.132133	-0.222982	-0.571498	-0.058852
Employed_1	0.083681	0.174846	0.132133	0.222982	0.571498	0.058852
DriversLicense_0	-0.054778	0.013023	-0.075946	-0.138139	-0.006944	-0.080080
DriversLicense_1	0.054778	-0.013023	0.075946	0.138139	0.006944	0.080080

	Age	Debt	EducationLevel	YearsEmployed	CreditScore	ZipCode
Citizen_0	0.011500	0.123569	0.032039	0.031670	0.142938	-0.127782
Citizen_1	0.013184	-0.037842	-0.119320	-0.065938	-0.053491	0.172419
Citizen_2	-0.017329	-0.116404	0.012403	-0.007965	-0.130871	0.068542

37 rows × 37 columns

In [43]:
```
dataset.drop('Approved', axis=1).corrwith(dataset.Approved).plot(kind='bar',
                                                          title="Correlation with Ap
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x17e13d15948>



In [44]:
```
dataset['Approved'].corr(dataset['Age'])
```

Out[44]: -0.1616273697063664

In [45]:
```
dataset['Approved'].corr(dataset['Debt'])
```

Out[45]: -0.20629373864503894

In [46]:
```
dataset['Approved'].corr(dataset['EducationLevel'])
```

Out[46]:
```
-0.13002568654880345
```

In [49]:
```
dataset['Approved'].corr(dataset['YearsEmployed'])
```

Out[49]:
```
-0.3224753582553844
```

In [50]:
```
dataset['Approved'].corr(dataset['CreditScore'])
```

Out[50]:
```
-0.4064100087639563
```

In [51]:
```
dataset['Approved'].corr(dataset['ZipCode'])
```

Out[51]:
```
-0.05863341377788577
```

In [52]:
```
dataset['Approved'].corr(dataset['Income'])
```

Out[52]:
```
-0.17565720099350488
```

machine learning algorithms

In [53]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        print("Train Result:\n================================================
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print("Classification Report:", end='')
        print(f"\tPrecision Score: {precision_score(y_train, pred) * 100:.2f}
        print(f"\t\t\tRecall Score: {recall_score(y_train, pred) * 100:.2f}%"
        print(f"\t\t\tF1 score: {f1_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        print("Test Result:\n=================================================
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print("Classification Report:", end='')
        print(f"\tPrecision Score: {precision_score(y_test, pred) * 100:.2f}%
        print(f"\t\t\tRecall Score: {recall_score(y_test, pred) * 100:.2f}%")
        print(f"\t\t\tF1 score: {f1_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

In [108]:

```
from sklearn.model_selection import train_test_split

X = dataset.drop('Approved', axis=1)
y = dataset.Approved

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ranc
```

In [109]:

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(solver='liblinear')
log_reg.fit(X_train, y_train)
```

Out[109]:

```
LogisticRegression(solver='liblinear')
```

In [110]:

```
print_score(log_reg, X_train, y_train, X_test, y_test, train=True)
print_score(log_reg, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 88.61%
_______________________________________________
Classification Report:  Precision Score: 91.60%
                        Recall Score: 87.91%
                        F1 score: 89.72%
_______________________________________________
Confusion Matrix:
 [[188  22]
 [ 33 240]]

Test Result:
================================================
Accuracy Score: 83.09%
_______________________________________________
Classification Report:  Precision Score: 87.13%
                        Recall Score: 80.00%
                        F1 score: 83.41%
_______________________________________________
Confusion Matrix:
 [[84 13]
 [22 88]]
```

In [111]:

```
test_score = accuracy_score(y_test, log_reg.predict(X_test)) * 100
train_score = accuracy_score(y_train, log_reg.predict(X_train)) * 100

results_df = pd.DataFrame(data=[["Logistic Regression", train_score, test_scc
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df
```

Out[111]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	88.612836	83.091787

K-nearest neighbors

In [112]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)

print_score(knn_classifier, X_train, y_train, X_test, y_test, train=True)
print_score(knn_classifier, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 89.86%
_______________________________________________
Classification Report:  Precision Score: 88.62%
                        Recall Score: 94.14%
                        F1 score: 91.30%
_______________________________________________
Confusion Matrix:
 [[177  33]
 [ 16 257]]

Test Result:
================================================
Accuracy Score: 83.57%
_______________________________________________
Classification Report:  Precision Score: 82.76%
                        Recall Score: 87.27%
                        F1 score: 84.96%
_______________________________________________
Confusion Matrix:
 [[77 20]
 [14 96]]
```

In [113]:
```
test_score = accuracy_score(y_test, knn_classifier.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_classifier.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["K-nearest neighbors", train_score, test_s
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[113]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	88.612836	83.091787
1	K-nearest neighbors	89.855072	83.574879

Support Vector machine

In [114]:
```
from sklearn.svm import SVC


svm_model = SVC(kernel='rbf', gamma=0.1, C=1.0)
svm_model.fit(X_train, y_train)
```

Out[114]:
```
SVC(gamma=0.1)
```

In [115]:
```
print_score(svm_model, X_train, y_train, X_test, y_test, train=True)
print_score(svm_model, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 90.68%
_______________________________________________
Classification Report:  Precision Score: 94.53%
                        Recall Score: 88.64%
                        F1 score: 91.49%
_______________________________________________
Confusion Matrix:
 [[196  14]
 [ 31 242]]

Test Result:
================================================
Accuracy Score: 84.54%
_______________________________________________
Classification Report:  Precision Score: 86.79%
                        Recall Score: 83.64%
                        F1 score: 85.19%
_______________________________________________
Confusion Matrix:
 [[83 14]
 [18 92]]
```

In [116]:
```
test_score = accuracy_score(y_test, svm_model.predict(X_test)) * 100
train_score = accuracy_score(y_train, svm_model.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Support Vector Machine", train_score, tes
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[116]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	88.612836	83.091787
1	K-nearest neighbors	89.855072	83.574879
2	Support Vector Machine	90.683230	84.541063

Decision Tree Classifier

In [117]:
```
from sklearn.tree import DecisionTreeClassifier


tree = DecisionTreeClassifier(random_state=42)
tree.fit(X_train, y_train)

print_score(tree, X_train, y_train, X_test, y_test, train=True)
print_score(tree, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[210   0]
 [  0 273]]

Test Result:
================================================
Accuracy Score: 85.02%
_______________________________________________
Classification Report:  Precision Score: 81.60%
                        Recall Score: 92.73%
                        F1 score: 86.81%
_______________________________________________
Confusion Matrix:
 [[ 74  23]
 [  8 102]]
```

In [118]:
```
test_score = accuracy_score(y_test, tree.predict(X_test)) * 100
train_score = accuracy_score(y_train, tree.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Decision Tree Classifier", train_score, t
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[118]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	88.612836	83.091787
1	K-nearest neighbors	89.855072	83.574879
2	Support Vector Machine	90.683230	84.541063
3	Decision Tree Classifier	100.000000	85.024155

Random Forest

In [119]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

rand_forest = RandomForestClassifier(n_estimators=1000, random_state=42)
rand_forest.fit(X_train, y_train)

print_score(rand_forest, X_train, y_train, X_test, y_test, train=True)
print_score(rand_forest, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[210   0]
 [  0 273]]

Test Result:
================================================
Accuracy Score: 85.02%
_______________________________________________
Classification Report:  Precision Score: 86.92%
                        Recall Score: 84.55%
                        F1 score: 85.71%
_______________________________________________
Confusion Matrix:
 [[83 14]
 [17 93]]
```

In [120]:
```
test_score = accuracy_score(y_test, rand_forest.predict(X_test)) * 100
train_score = accuracy_score(y_train, rand_forest.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Random Forest Classifier", train_score, t
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[120]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	88.612836	83.091787
1	K-nearest neighbors	89.855072	83.574879
2	Support Vector Machine	90.683230	84.541063
3	Decision Tree Classifier	100.000000	85.024155
4	Random Forest Classifier	100.000000	85.024155

XGBoost Classifer

XGBoost Classifier

In [121]:

```
#pip install xgboost     installing xgboost
```

In [122]:

```
from xgboost import XGBClassifier

xgb = XGBClassifier()
xgb.fit(X_train, y_train)

print_score(xgb, X_train, y_train, X_test, y_test, train=True)
print_score(xgb, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:	Precision Score: 100.00%
			Recall Score: 100.00%
			F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[210   0]
 [  0 273]]

Test Result:
================================================
Accuracy Score: 86.96%
_______________________________________________
Classification Report:	Precision Score: 85.47%
			Recall Score: 90.91%
			F1 score: 88.11%
_______________________________________________
Confusion Matrix:
 [[ 80  17]
 [ 10 100]]
```

In [123]:
```
test_score = accuracy_score(y_test, xgb.predict(X_test)) * 100
train_score = accuracy_score(y_train, xgb.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["XGBoost Classifier", train_score, test_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[123]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	88.612836	83.091787
1	K-nearest neighbors	89.855072	83.574879
2	Support Vector Machine	90.683230	84.541063
3	Decision Tree Classifier	100.000000	85.024155
4	Random Forest Classifier	100.000000	85.024155
5	XGBoost Classifier	100.000000	86.956522

Using Hyperparameter Tuning

Logistic Regression Hyperparameter Tuning

In [124]:
```
from sklearn.model_selection import GridSearchCV

params = {"C": np.logspace(-4, 4, 20),
          "solver": ["liblinear"]}

log_reg = LogisticRegression()

grid_search_cv = GridSearchCV(log_reg, params, scoring="accuracy", n_jobs=-1,
# grid_search_cv.fit(X_train, y_train)
```

In [125]:
```
# grid_search_cv.best_estimator_
```

In [126]:
```
log_reg = LogisticRegression(C=0.615848211066026,
                             solver='liblinear')

log_reg.fit(X_train, y_train)

print_score(log_reg, X_train, y_train, X_test, y_test, train=True)
print_score(log_reg, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 88.61%
_______________________________________________
Classification Report:  Precision Score: 91.29%
                        Recall Score: 88.28%
                        F1 score: 89.76%
_______________________________________________
Confusion Matrix:
 [[187  23]
 [ 32 241]]

Test Result:
================================================
Accuracy Score: 83.09%
_______________________________________________
Classification Report:  Precision Score: 87.13%
                        Recall Score: 80.00%
                        F1 score: 83.41%
_______________________________________________
Confusion Matrix:
 [[84 13]
 [22 88]]
```

In [127]:
```
test_score = accuracy_score(y_test, log_reg.predict(X_test)) * 100
train_score = accuracy_score(y_train, log_reg.predict(X_train)) * 100

tuning_results_df = pd.DataFrame(data=[["Tuned Logistic Regression", train_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df
```

Out[127]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.612836	83.091787

K-nearest neighbors Hyperparameter Tuning

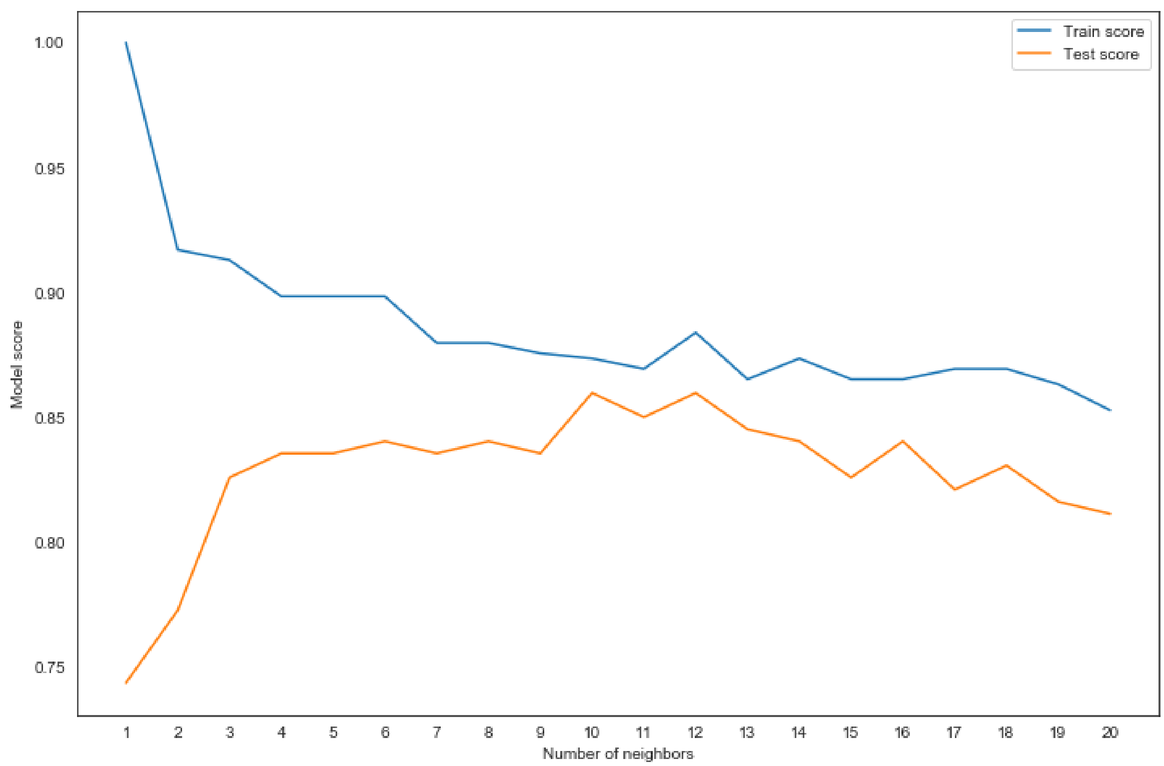
In [128]:

```
train_score = []
test_score = []
neighbors = range(1, 21)

for k in neighbors:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    train_score.append(accuracy_score(y_train, model.predict(X_train)))
    test_score.append(accuracy_score(y_test, model.predict(X_test)))
```

In [129]:

```
plt.figure(figsize=(12, 8))

plt.plot(neighbors, train_score, label="Train score")
plt.plot(neighbors, test_score, label="Test score")
plt.xticks(np.arange(1, 21, 1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model score")
plt.legend()

print(f"Maximum KNN score on the test data: {max(test_score)*100:.2f}%")
```

```
Maximum KNN score on the test data: 85.99%
```



In [130]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=19)
knn_classifier.fit(X_train, y_train)

print_score(knn_classifier, X_train, y_train, X_test, y_test, train=True)
print_score(knn_classifier, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 86.34%
_______________________________________________
Classification Report:	Precision Score: 84.39%
			Recall Score: 93.04%
			F1 score: 88.50%
_______________________________________________
Confusion Matrix:
 [[163  47]
 [ 19 254]]

Test Result:
================================================
Accuracy Score: 81.64%
_______________________________________________
Classification Report:	Precision Score: 77.69%
			Recall Score: 91.82%
			F1 score: 84.17%
_______________________________________________
Confusion Matrix:
 [[ 68  29]
 [  9 101]]
```

In [131]:
```
test_score = accuracy_score(y_test, knn_classifier.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_classifier.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned K-nearest neighbors", train_score,
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[131]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.612836	83.091787
1	Tuned K-nearest neighbors	86.335404	81.642512

In [132]:
```
### Support Vector Machine Hyperparameter Tuning
```

In [133]:
```
svm_model = SVC(kernel='rbf', gamma=0.1, C=1.0)

params = {"C":(0.1, 0.5, 1, 2, 5, 10, 20),
          "gamma":(0.001, 0.01, 0.1, 0.25, 0.5, 0.75, 1),
          "kernel":('linear', 'poly', 'rbf')}

svm_grid = GridSearchCV(svm_model, params, n_jobs=-1, cv=5, verbose=1, scorir
# svm_grid.fit(X_train, y_train)
```

In [134]:
```
# svm_grid.best_estimator_
```

In [135]:
```
svm_model = SVC(C=5, gamma=0.01, kernel='rbf')
svm_model.fit(X_train, y_train)

print_score(svm_model, X_train, y_train, X_test, y_test, train=True)
print_score(svm_model, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 86.75%
_______________________________________________
Classification Report:  Precision Score: 94.47%
                        Recall Score: 81.32%
                        F1 score: 87.40%
_______________________________________________
Confusion Matrix:
 [[197  13]
 [ 51 222]]

Test Result:
================================================
Accuracy Score: 84.06%
_______________________________________________
Classification Report:  Precision Score: 91.40%
                        Recall Score: 77.27%
                        F1 score: 83.74%
_______________________________________________
Confusion Matrix:
 [[89  8]
 [25 85]]
```

In [136]:
```
test_score = accuracy_score(y_test, svm_model.predict(X_test)) * 100
train_score = accuracy_score(y_train, svm_model.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned Support Vector Machine", train_scor
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[136]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.612836	83.091787
1	Tuned K-nearest neighbors	86.335404	81.642512
2	Tuned Support Vector Machine	86.749482	84.057971

Decision Tree Classifier Hyperparameter Tuning

In [137]:
```
params = {"criterion":("gini", "entropy"),
          "splitter":("best", "random"),
          "max_depth":(list(range(1, 20))),
          "min_samples_split":[2, 3, 4],
          "min_samples_leaf":list(range(1, 20))
          }

tree = DecisionTreeClassifier(random_state=42)
grid_search_cv = GridSearchCV(tree, params, scoring="accuracy", n_jobs=-1, ve
# grid_search_cv.fit(X_train, y_train)
```

In [138]:
```
# grid_search_cv.best_estimator_
```

In [139]:

```
tree = DecisionTreeClassifier(criterion='gini',
                              max_depth=3,
                              min_samples_leaf=2,
                              min_samples_split=2,
                              splitter='random')
tree.fit(X_train, y_train)

print_score(tree, X_train, y_train, X_test, y_test, train=True)
print_score(tree, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 87.78%
_______________________________________________
Classification Report:  Precision Score: 90.53%
                        Recall Score: 87.55%
                        F1 score: 89.01%
_______________________________________________
Confusion Matrix:
 [[185  25]
 [ 34 239]]

Test Result:
================================================
Accuracy Score: 82.61%
_______________________________________________
Classification Report:  Precision Score: 82.46%
                        Recall Score: 85.45%
                        F1 score: 83.93%
_______________________________________________
Confusion Matrix:
 [[77 20]
 [16 94]]
```

In [140]:

```
test_score = accuracy_score(y_test, tree.predict(X_test)) * 100
train_score = accuracy_score(y_train, tree.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned Decision Tree Classifier", train_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[140]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.612836	83.091787
1	Tuned K-nearest neighbors	86.335404	81.642512
2	Tuned Support Vector Machine	86.749482	84.057971
3	Tuned Decision Tree Classifier	87.784679	82.608696

Random Forest Classifier Hyperparameter Tuning

In [141]:
```
from sklearn.model_selection import RandomizedSearchCV

n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num=11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

random_grid = {'n_estimators': n_estimators, 'max_features': max_features,
               'max_depth': max_depth, 'min_samples_split': min_samples_split
               'min_samples_leaf': min_samples_leaf, 'bootstrap': bootstrap}


rand_forest = RandomForestClassifier(random_state=42)

rf_random = RandomizedSearchCV(estimator=rand_forest, param_distributions=rar
                               verbose=2, random_state=42, n_jobs=-1)
# rf_random.fit(X_train, y_train)
```

In [142]:
```
# rf_random.best_estimator_
```

In [143]:
```
rand_forest = RandomForestClassifier(bootstrap=True,
                                     max_depth=70,
                                     max_features='auto',
                                     min_samples_leaf=4,
                                     min_samples_split=10,
                                     n_estimators=400)
rand_forest.fit(X_train, y_train)
```

Out[143]:
```
RandomForestClassifier(max_depth=70, min_samples_leaf=4, min_samples_split=
10,
                       n_estimators=400)
```

In [144]:

```
print_score(rand_forest, X_train, y_train, X_test, y_test, train=True)
print_score(rand_forest, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 92.75%
_______________________________________________
Classification Report:	Precision Score: 94.74%
			Recall Score: 92.31%
			F1 score: 93.51%
_______________________________________________
Confusion Matrix:
 [[196  14]
 [ 21 252]]

Test Result:
================================================
Accuracy Score: 85.51%
_______________________________________________
Classification Report:	Precision Score: 88.46%
			Recall Score: 83.64%
			F1 score: 85.98%
_______________________________________________
Confusion Matrix:
 [[85 12]
 [18 92]]
```

In [145]:

```
test_score = accuracy_score(y_test, rand_forest.predict(X_test)) * 100
train_score = accuracy_score(y_train, rand_forest.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned Random Forest Classifier", train_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[145]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.612836	83.091787
1	Tuned K-nearest neighbors	86.335404	81.642512
2	Tuned Support Vector Machine	86.749482	84.057971
3	Tuned Decision Tree Classifier	87.784679	82.608696
4	Tuned Random Forest Classifier	92.753623	85.507246

XGBoost Classifier Hyperparameter Tuning

In [146]:

```
n_estimators = [100, 500, 900, 1100, 1500]
max_depth = [2, 3, 5, 10, 15]
booster = ['gbtree', 'gblinear']
base_score = [0.25, 0.5, 0.75, 0.99]
learning_rate = [0.05, 0.1, 0.15, 0.20]
min_child_weight = [1, 2, 3, 4]

hyperparameter_grid = {'n_estimators': n_estimators, 'max_depth': max_depth,
                       'learning_rate' : learning_rate, 'min_child_weight' :
                       'booster' : booster, 'base_score' : base_score
                        }

xgb_model = XGBClassifier()

xgb_cv = RandomizedSearchCV(estimator=xgb_model, param_distributions=hyperpar
                               cv=5, n_iter=650, scoring = 'accuracy',n_jobs
                               verbose=1, return_train_score = True, random_s


# xgb_cv.fit(X_train, y_train)
```

In [147]:

```
# xgb_cv.best_estimator_
```

In [148]:

```
xgb_best = XGBClassifier(base_score=0.25,
                         booster='gbtree',
                         learning_rate=0.05,
                         max_depth=5,
                         min_child_weight=2,
                         n_estimators=100)
xgb_best.fit(X_train, y_train)
```

Out[148]:

```
XGBClassifier(base_score=0.25, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.05, max_delta_step=0, max_depth=5,
              min_child_weight=2, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state
=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [149]:

```
print_score(xgb_best, X_train, y_train, X_test, y_test, train=True)
print_score(xgb_best, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 96.48%
_______________________________________________
Classification Report:	Precision Score: 96.04%
			Recall Score: 97.80%
			F1 score: 96.91%
_______________________________________________
Confusion Matrix:
 [[199  11]
 [  6 267]]

Test Result:
================================================
Accuracy Score: 85.02%
_______________________________________________
Classification Report:	Precision Score: 86.24%
			Recall Score: 85.45%
			F1 score: 85.84%
_______________________________________________
Confusion Matrix:
 [[82 15]
 [16 94]]
```

In [150]:

```
test_score = accuracy_score(y_test, xgb_best.predict(X_test)) * 100
train_score = accuracy_score(y_train, xgb_best.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned XGBoost Classifier", train_score, t
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[150]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.612836	83.091787
1	Tuned K-nearest neighbors	86.335404	81.642512
2	Tuned Support Vector Machine	86.749482	84.057971
3	Tuned Decision Tree Classifier	87.784679	82.608696
4	Tuned Random Forest Classifier	92.753623	85.507246
5	Tuned XGBoost Classifier	96.480331	85.024155

In [151]:
```
results_df
```

Out[151]:

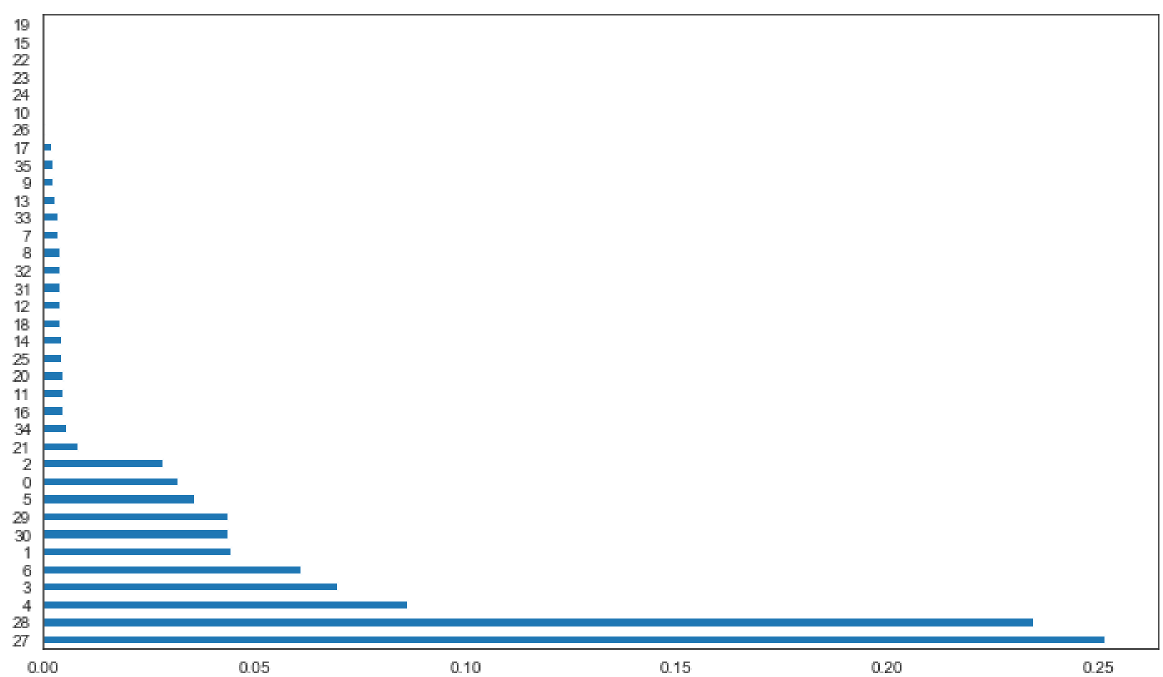
	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	88.612836	83.091787
1	K-nearest neighbors	89.855072	83.574879
2	Support Vector Machine	90.683230	84.541063
3	Decision Tree Classifier	100.000000	85.024155
4	Random Forest Classifier	100.000000	85.024155
5	XGBoost Classifier	100.000000	86.956522

Features Importance According to Random Forest and XGBoost

In [152]:
```
def feature_imp(df, model):
    fi = pd.DataFrame()
    fi["feature"] = df.columns
    fi["importance"] = model.feature_importances_
    return fi.sort_values(by="importance", ascending=False)
```

In [153]:
```
feature_imp(X, rand_forest).plot(kind='barh', figsize=(12,7), legend=False)
```

Out[153]: <matplotlib.axes._subplots.AxesSubplot at 0x17e1582b348>



In [154]:
```
feature_imp(X, xgb_best).plot(kind='barh', figsize=(12,7), legend=False)
```

Out[154]: <matplotlib.axes._subplots.AxesSubplot at 0x17e15524208>