Credit card approval Prediction using crx data

Importing libraries

In [208]:

```
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 [209]:

```
data = pd.read_csv('crx.data')
```

In [210]:

```
data
```

Out[210]:

	b	30.83	0	u	g	w	v	1.25	t	t.1	01	f	g.1	00202	0.1	+
0	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
1	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
2	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
3	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+
4	b	32.08	4.000	u	g	m	v	2.50	t	f	0	t	g	00360	0	+
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
684	b	21.08	10.085	y	p	e	h	1.25	f	f	0	f	g	00260	0	-
685	a	22.67	0.750	u	g	c	v	2.00	f	t	2	t	g	00200	394	-
686	a	25.25	13.500	y	p	ff	ff	2.00	f	t	1	t	g	00200	1	-
687	b	17.92	0.205	u	g	aa	v	0.04	f	f	0	f	g	00280	750	-
688	b	35.00	3.375	u	g	c	h	8.29	f	f	0	t	g	00000	0	-

689 rows × 16 columns

In [211]:
```
# shape
print(data.shape)
```

```
(689, 16)
```

In [212]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 689 entries, 0 to 688
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   b       689 non-null    object 
 1   30.83   689 non-null    object 
 2   0       689 non-null    float64
 3   u       689 non-null    object 
 4   g       689 non-null    object 
 5   w       689 non-null    object 
 6   v       689 non-null    object 
 7   1.25    689 non-null    float64
 8   t       689 non-null    object 
 9   t.1     689 non-null    object 
 10  01      689 non-null    int64  
 11  f       689 non-null    object 
 12  g.1     689 non-null    object 
 13  00202   689 non-null    object 
 14  0.1     689 non-null    int64  
 15  +       689 non-null    object 
dtypes: float64(2), int64(2), object(12)
memory usage: 86.2+ KB
```

Information collected to find the columns representing the data

In [213]:
```
## Classes 'tbl_df', 'tbl' and 'data.frame':    690 obs. of  16 variables:
##  $ Male          : chr  "b" "a" "a" "b" ...
##  $ Age           : chr  "30.83" "58.67" "24.50" "27.83" ...
##  $ Debt          : num  0 4.46 0.5 1.54 5.62 ...
##  $ Married       : chr  "u" "u" "u" "u" ...
##  $ BankCustomer  : chr  "g" "g" "g" "g" ...
##  $ EducationLevel: chr  "w" "q" "q" "w" ...
##  $ Ethnicity     : chr  "v" "h" "h" "v" ...
##  $ YearsEmployed : num  1.25 3.04 1.5 3.75 1.71 ...
##  $ PriorDefault  : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ Employed      : logi  TRUE TRUE FALSE TRUE FALSE FALSE ...
##  $ CreditScore   : int  1 6 0 5 0 0 0 0 0 0 ...
##  $ DriversLicense: logi  FALSE FALSE FALSE TRUE FALSE TRUE ...
##  $ Citizen       : chr  "g" "g" "g" "g" ...
##  $ ZipCode       : chr  "00202" "00043" "00280" "00100" ...
##  $ Income        : int  0 560 824 3 0 0 31285 1349 314 1442 ...
##  $ Approved      : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
```

Assigning relevant column names

In [214]:
```
header_list = ['sex','Age','Debt','Married','BankCustomer','EducationLevel','
               'Employed','CreditScore','DriversLicense','Citizen','ZipCode',
```

In [215]:
```
data = pd.read_csv('crx.data',names=header_list)
```

In [216]:
```
data
```

Out[216]:

	sex	Age	Debt	Married	BankCustomer	EducationLevel	Ethnicity	YearsEmployed	P
0	b	30.83	0.000	u	g	w	v	1.25	
1	a	58.67	4.460	u	g	q	h	3.04	
2	a	24.50	0.500	u	g	q	h	1.50	
3	b	27.83	1.540	u	g	w	v	3.75	
4	b	20.17	5.625	u	g	w	v	1.71	
...	...	...	...	...	...	...	...	...	
685	b	21.08	10.085	y	p	e	h	1.25	
686	a	22.67	0.750	u	g	c	v	2.00	
687	a	25.25	13.500	y	p	ff	ff	2.00	
688	b	17.92	0.205	u	g	aa	v	0.04	
689	b	35.00	3.375	u	g	c	h	8.29	

690 rows × 16 columns

In [217]:
```
data.columns
```

Out[217]:
```
Index(['sex', 'Age', 'Debt', 'Married', 'BankCustomer', 'EducationLevel',
       'Ethnicity', 'YearsEmployed', 'PriorDefault', 'Employed', 'CreditSco
re',
       'DriversLicense', 'Citizen', 'ZipCode', 'Income', 'Approved'],
      dtype='object')
```

In [218]:
```
# descriptions
print(data.describe())
```

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

In [219]:
```
# Checking for missing values
data.isna().sum()
```

Out[219]:
```
sex               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 [220]:
```
# Checking for missing values another method
pd.isnull(data).any()
```

Out[220]:
```
sex               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 [221]:
```
data.index
```

Out[221]:
```
RangeIndex(start=0, stop=690, step=1)
```

In [222]:
```
#counting sex , here male=a and female = b
data['sex'].value_counts()
```

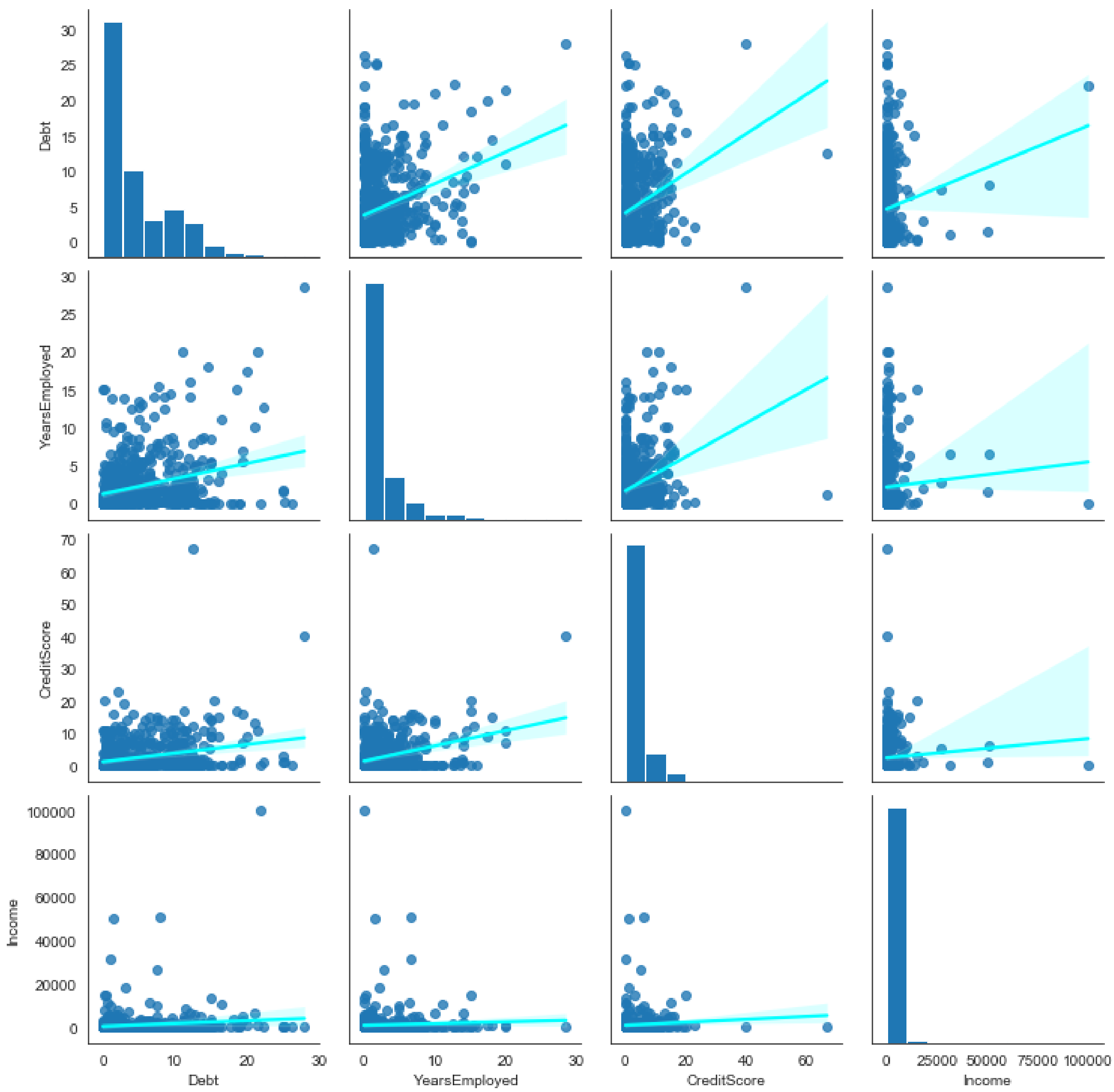
Out[222]:
```
b    468
a    210
?     12
Name: sex, dtype: int64
```

In [223]:
```
# count by presence of Approved  , here + for yes and - for no
data['Approved'].value_counts()

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

Out[223]:
```
-    383
+    307
Name: Approved, dtype: int64
```

Initial visualization

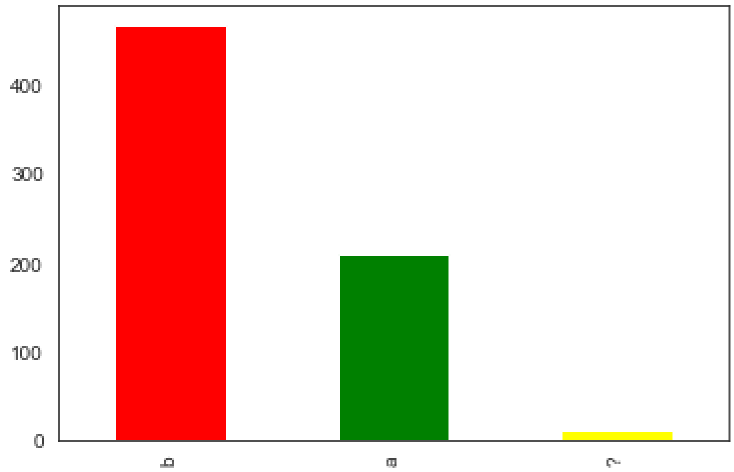
In [224]:
```
%%time
sns.pairplot(data, kind='reg', plot_kws={'line_kws':{'color': 'cyan'}})
plt.show()
```




```
Wall time: 5.7 s
```

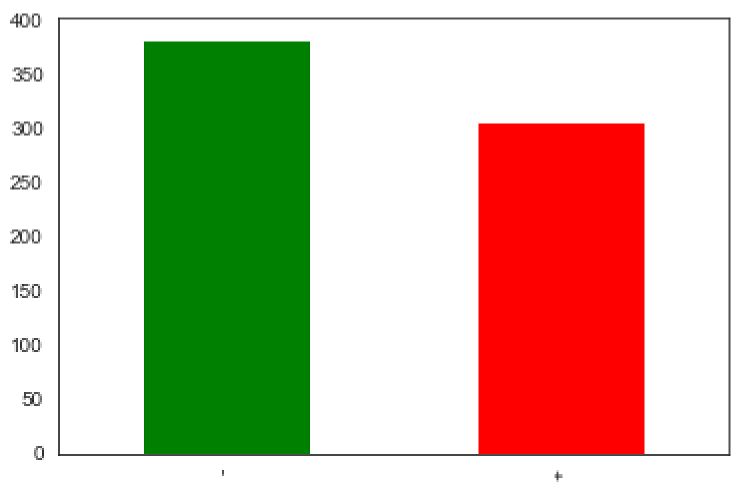
In [225]:
```
data.sex.value_counts().plot(kind="bar", color=["red", "green",'yellow'])
```

Out[225]: <matplotlib.axes._subplots.AxesSubplot at 0x2cdb2fb9248>



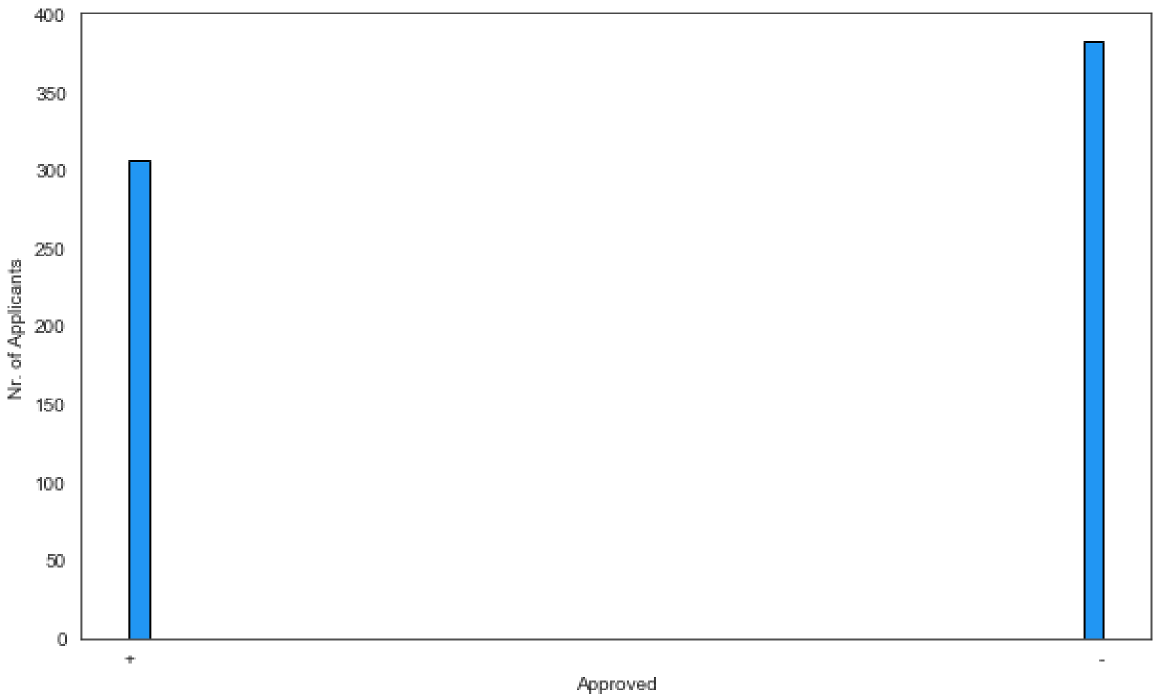
In [226]:
```
data.Approved.value_counts().plot(kind="bar", color=["green","red"])
```

Out[226]: <matplotlib.axes._subplots.AxesSubplot at 0x2cdb33c4448>



In [227]:

```
plt.figure(figsize=(10, 6))
plt.hist(data['Approved'], bins=50, ec='black', color='#2196f3')
plt.xlabel('Approved')
plt.ylabel('Nr. of Applicants ')
plt.show()
```



In [228]:
```
data.info()
```

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

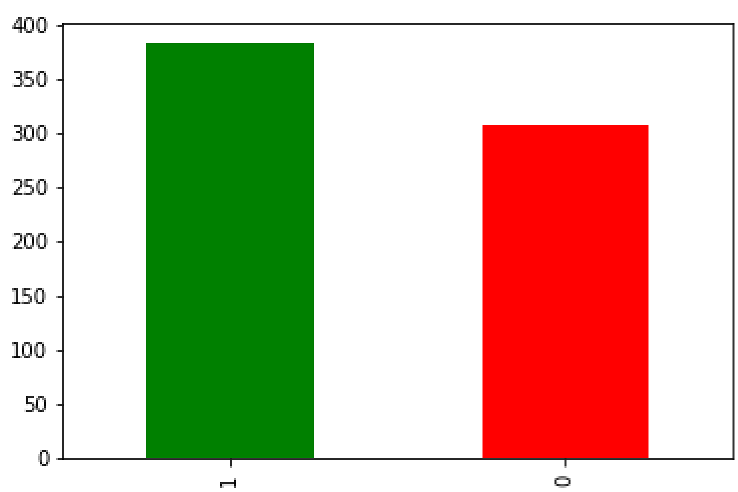
In [229]:
```
# Import LabelEncoder
from sklearn.preprocessing import LabelEncoder
# Instantiate LabelEncoder
le=LabelEncoder()

# 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])
```

In [230]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   sex             690 non-null    int32
 1   Age             690 non-null    int32
 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(2), int32(12), int64(2)
memory usage: 54.0 KB
```

In [130]:
```
data.Approved.value_counts().plot(kind="bar", color=["green","red"])
```

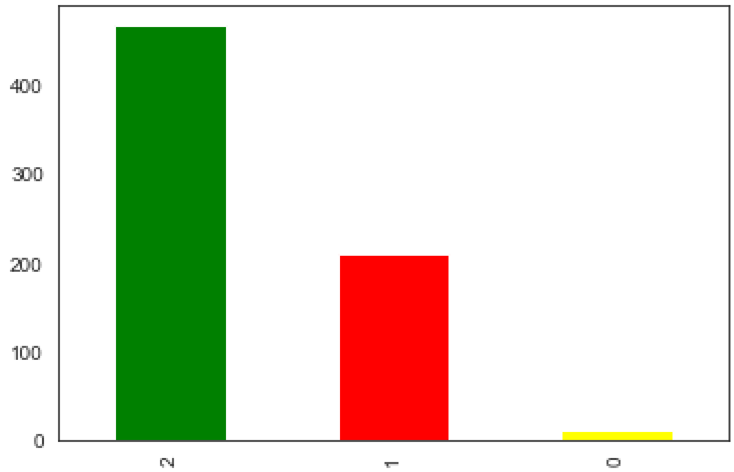
Out[130]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2cdaee82c88>
```



In [231]:

```
data.sex.value_counts().plot(kind="bar", color=["green","red","yellow"])
```

Out[231]: <matplotlib.axes._subplots.AxesSubplot at 0x2cdb0bf0e88>



In [232]:
```
categorical_col = []
continous_col = []
for column in data.columns:
    print('==============================')
    print(f"{column} : {data[column].unique()}")
    if len(data[column].unique()) <= 10:
        categorical_col.append(column)
    else:
        continous_col.append(column)
```

```
==============================
sex : [2 1 0]
==============================
Age : [156 328  89 125  43 168 179  74 310 255  64 145 220 282 270 211 1
29  78
  61  34  94 280 121 244   3 274 318 322 252 138 251 291 212  70 119  75
 124 311 188 136 142 234 316 309 243 166 247  84 103 106 215 196 189  79
  77 265 198 261 319 163  80  46 113 203 100  36 171 195 223 264 266  48
 187  59  57 292 123 235 349 213 102 186 289 334 161 303 135 133  69 132
 216 199  29 312 242  40 137  90 184  99 218 305 324  49 236  91 263  81
 279  72 190 131 340 278 206 173 287 182  31 116 159 317 258  67 175 239
 152 306 273 325 214  76 174 341 126 262  95 167 332 241  35 245 315 293
  71 117 107  26  54 321  68 288 237 240 134 178  55 249 191 283 122  85
  92 207 200 344 233 231  87 331 314 308  32 294 339  17  22  39  96 181
 326 108 257 219  47 284 130  30  27 267 238 246  19  23  41 302 296  12
  25 330  21 217 154  28   8  53  18  10 140  60  24 202   6 342 176   9
  65 323   4 164  33  16 183 148 180  97 194 277 228 256 226 335 170 114
 336 155   5  37 172 149 115  11  88 232  82  56 177 110 109 128  50 260
 320   1 144 157 301 343  38  66 222 112 197 139 290 300 327 307 118 105
```

In [233]:
```
categorical_col
```

Out[233]:
```
['sex',
 'Married',
 'BankCustomer',
 'Ethnicity',
 'PriorDefault',
 'Employed',
 'DriversLicense',
 'Citizen',
 'Approved']
```

In [234]:
```
continous_col
```

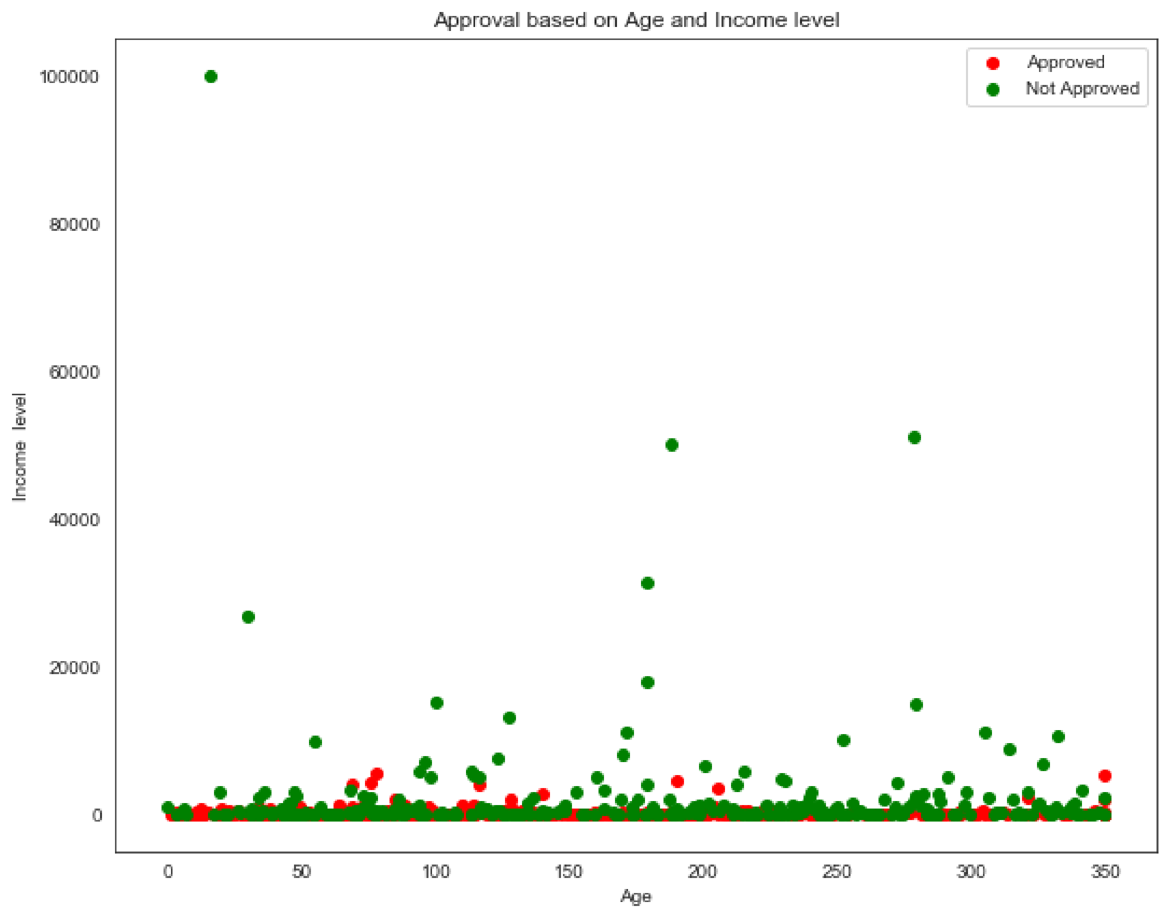
Out[234]:
```
['Age',
 'Debt',
 'EducationLevel',
 'YearsEmployed',
 'CreditScore',
 'ZipCode',
 'Income']
```

In [235]:

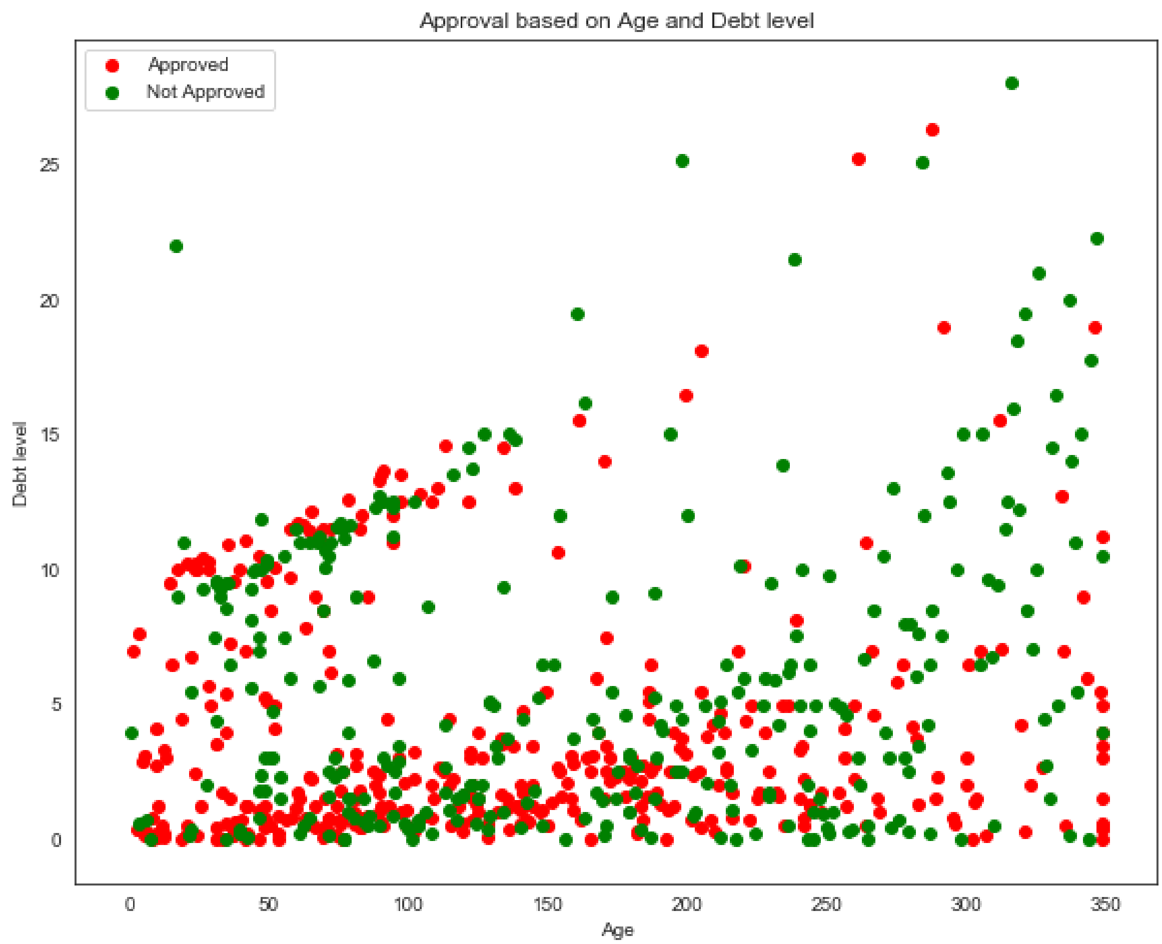
```
plt.figure(figsize=(15, 15))

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

In [236]:

```
# 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 [237]:
```
# 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 [238]:
```
data.corr() # Pearson Correlation Coefficients
```

Out[238]:

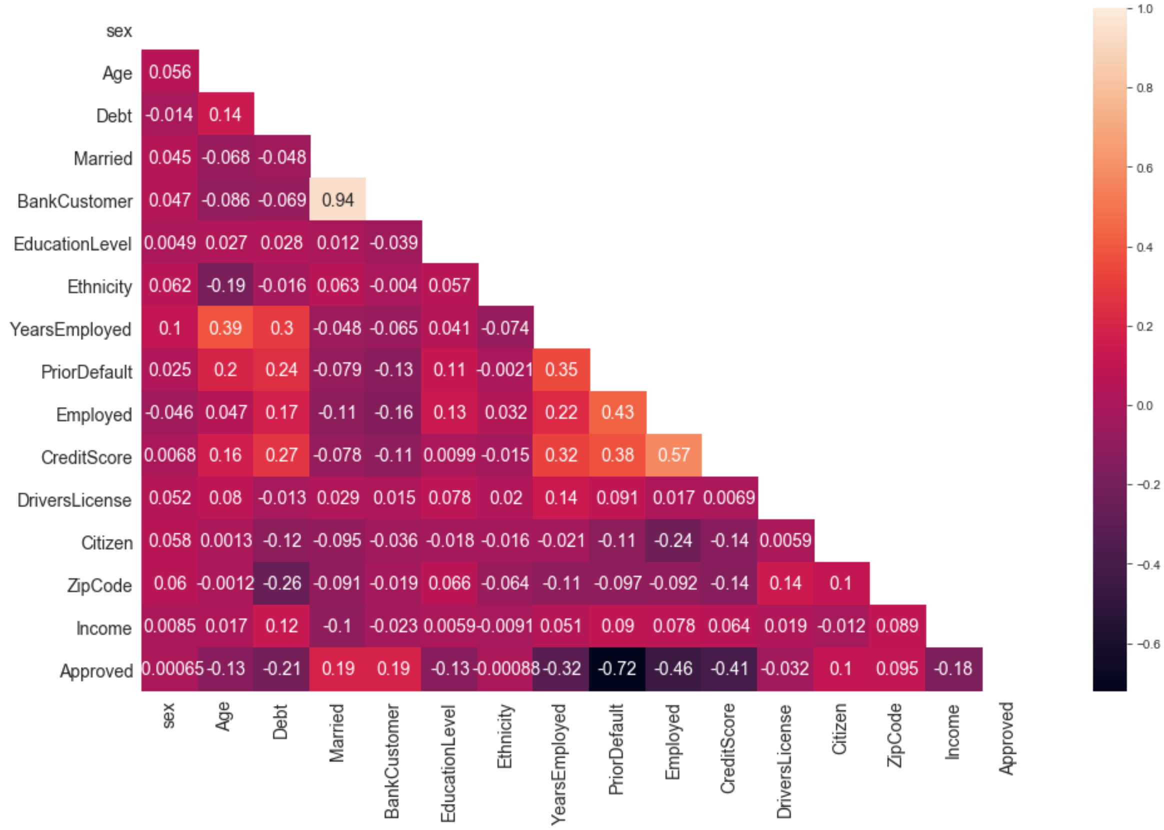
	sex	Age	Debt	Married	BankCustomer	EducationLevel	Et
sex	1.000000	0.055999	-0.013967	0.044990	0.047057	0.004854	0.
Age	0.055999	1.000000	0.135058	-0.068214	-0.085631	0.026752	-0.
Debt	-0.013967	0.135058	1.000000	-0.047608	-0.068773	0.027622	-0.
Married	0.044990	-0.068214	-0.047608	1.000000	0.942463	0.011832	0.
BankCustomer	0.047057	-0.085631	-0.068773	0.942463	1.000000	-0.038876	-0.
EducationLevel	0.004854	0.026752	0.027622	0.011832	-0.038876	1.000000	0.
Ethnicity	0.062229	-0.190155	-0.016451	0.063158	-0.003989	0.057192	1.
YearsEmployed	0.099863	0.386076	0.298902	-0.048423	-0.065497	0.040598	-0.
PriorDefault	0.025241	0.197493	0.244317	-0.078851	-0.129863	0.113752	-0.
Employed	-0.045808	0.047300	0.174846	-0.114926	-0.162464	0.132744	0.
CreditScore	0.006799	0.160599	0.271207	-0.077948	-0.106457	0.009907	-0.
DriversLicense	0.052396	0.079829	-0.013023	0.029057	0.015342	0.077824	0.
Citizen	0.058113	0.001284	-0.122233	-0.094585	-0.036095	-0.018090	-0.
ZipCode	0.059978	-0.001211	-0.262772	-0.091238	-0.018734	0.066057	-0.
Income	0.008504	0.016829	0.123121	-0.101102	-0.022904	0.005907	-0.
Approved	-0.000648	-0.133304	-0.206294	0.191431	0.187520	-0.129398	-0.

In [239]:
```
mask = np.zeros_like(data.corr())
triangle_indices = np.triu_indices_from(mask)
mask[triangle_indices] = True
mask
```

Out[239]:
```
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 [240]:

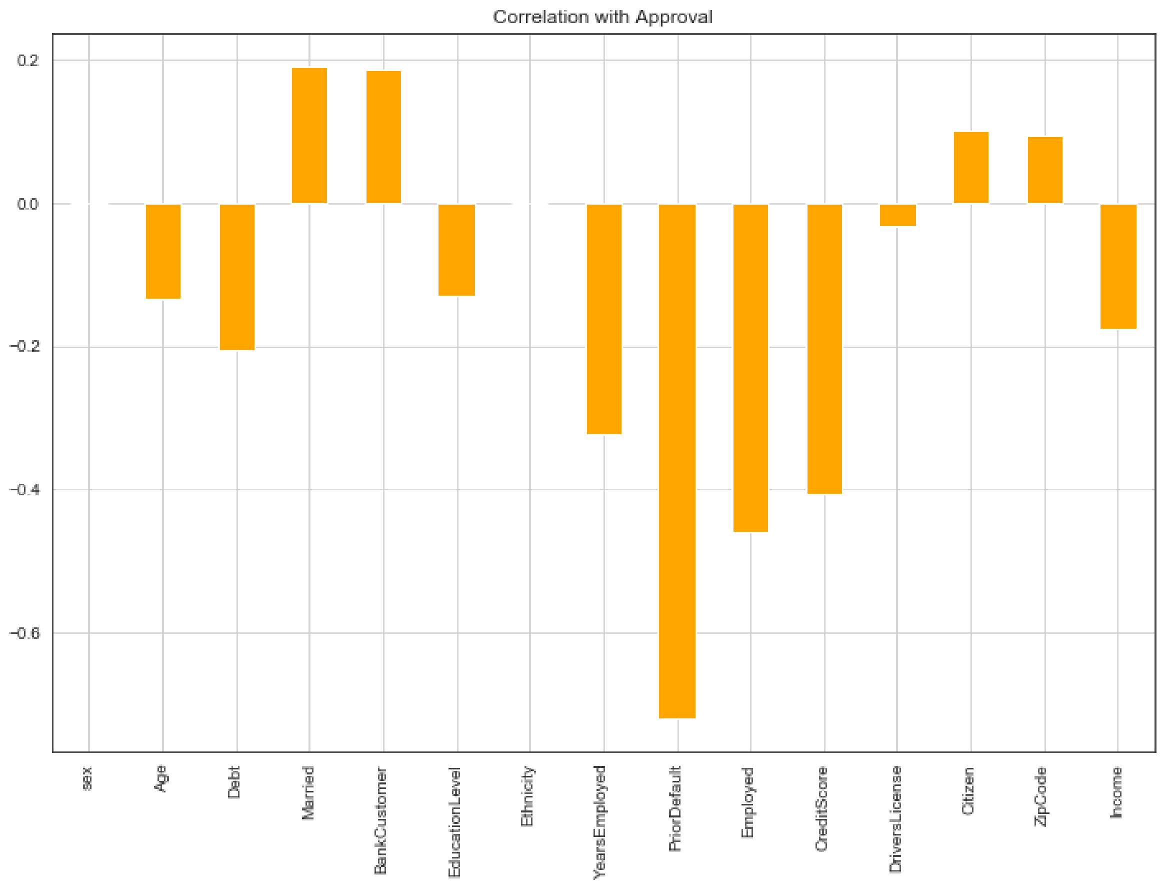
```
plt.figure(figsize=(16,10))
sns.heatmap(data.corr(), mask=mask, annot=True, annot_kws={"size": 14})
sns.set_style('white')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



In [241]:

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

Out[241]: <matplotlib.axes._subplots.AxesSubplot at 0x2cdb3020d08>



In [242]:

```
categorical_col.remove('Approved')
dataset = pd.get_dummies(data, columns = categorical_col)
```

In [243]:

```
dataset.head()
```

Out[243]:

	Age	Debt	EducationLevel	YearsEmployed	CreditScore	ZipCode	Income	Approved	sex
0	156	0.000	13	1.25	1	68	0	0	
1	328	4.460	11	3.04	6	11	560	0	
2	89	0.500	11	1.50	0	96	824	0	
3	125	1.540	13	3.75	5	31	3	0	
4	43	5.625	13	1.71	0	37	0	0	

5 rows × 38 columns

In [244]:

```
print(data.columns)
print(dataset.columns)
```

```
Index(['sex', 'Age', 'Debt', 'Married', 'BankCustomer', 'EducationLevel',
       'Ethnicity', 'YearsEmployed', 'PriorDefault', 'Employed', 'CreditSco
re',
       'DriversLicense', 'Citizen', 'ZipCode', 'Income', 'Approved'],
      dtype='object')
Index(['Age', 'Debt', 'EducationLevel', 'YearsEmployed', 'CreditScore',
       'ZipCode', 'Income', 'Approved', 'sex_0', 'sex_1', 'sex_2', '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 [245]:
```
dataset.describe()
```

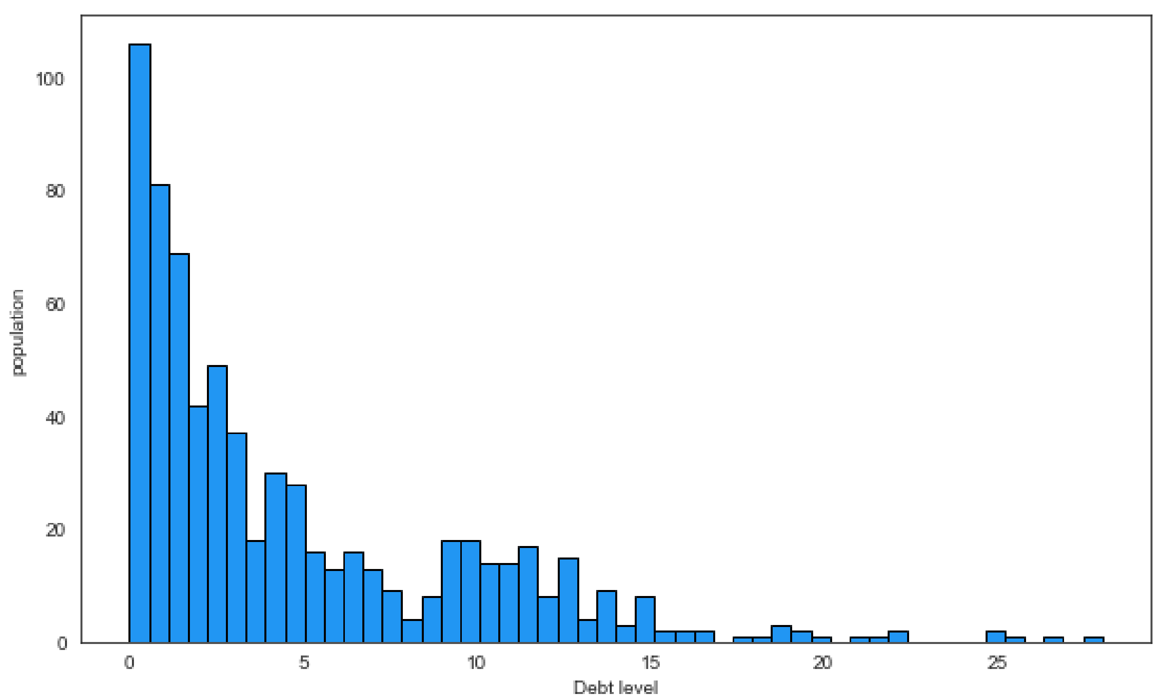
Out[245]:

	Age	Debt	EducationLevel	YearsEmployed	CreditScore	ZipCode	
count	690.000000	690.000000	690.000000	690.000000	690.00000	690.000000	6
mean	150.528986	4.758725	6.672464	2.223406	2.40000	59.392754	10
std	96.188946	4.978163	4.320266	3.346513	4.86294	48.231670	52
min	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	
25%	71.000000	1.000000	2.000000	0.165000	0.00000	23.000000	
50%	133.500000	2.750000	6.000000	1.000000	0.00000	52.000000	
75%	226.000000	7.207500	11.000000	2.625000	3.00000	96.000000	3
max	349.000000	28.000000	14.000000	28.500000	67.00000	170.000000	1000

8 rows × 38 columns

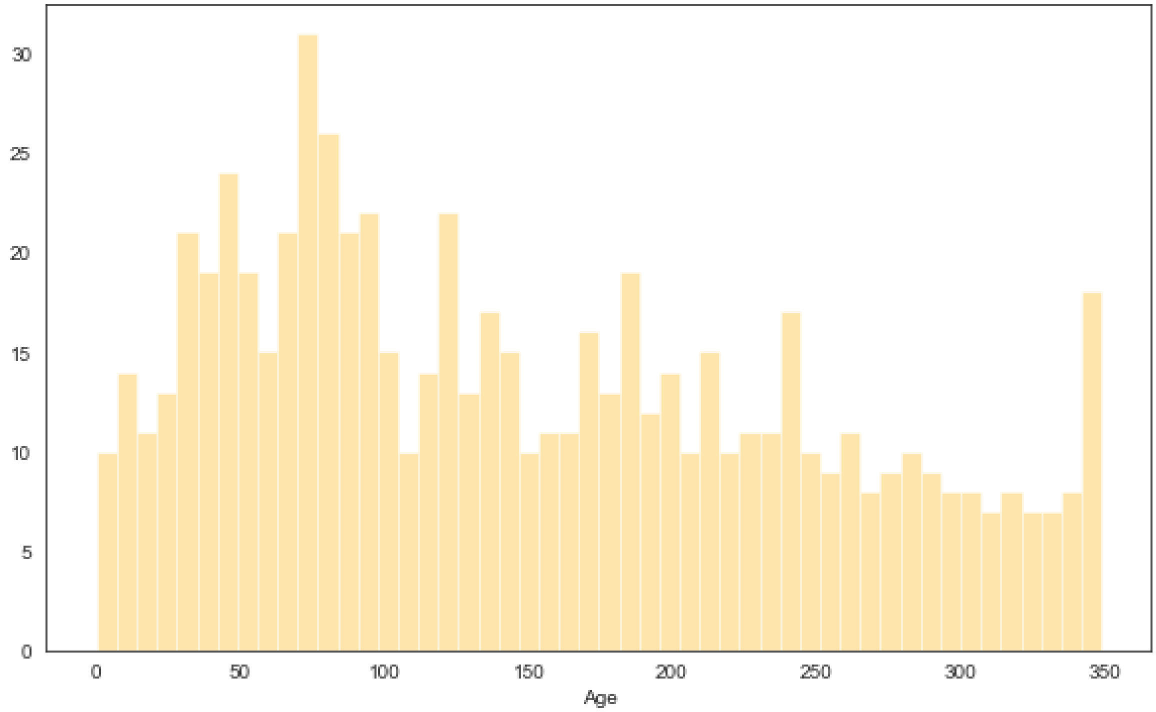
Visualising Data - Histograms, Distributions and Bar Charts

In [246]:
```
plt.figure(figsize=(10, 6))
plt.hist(dataset['Debt'], bins=50, ec='black', color='#2196f3')
plt.xlabel('Debt level')
plt.ylabel('population')
plt.show()
```



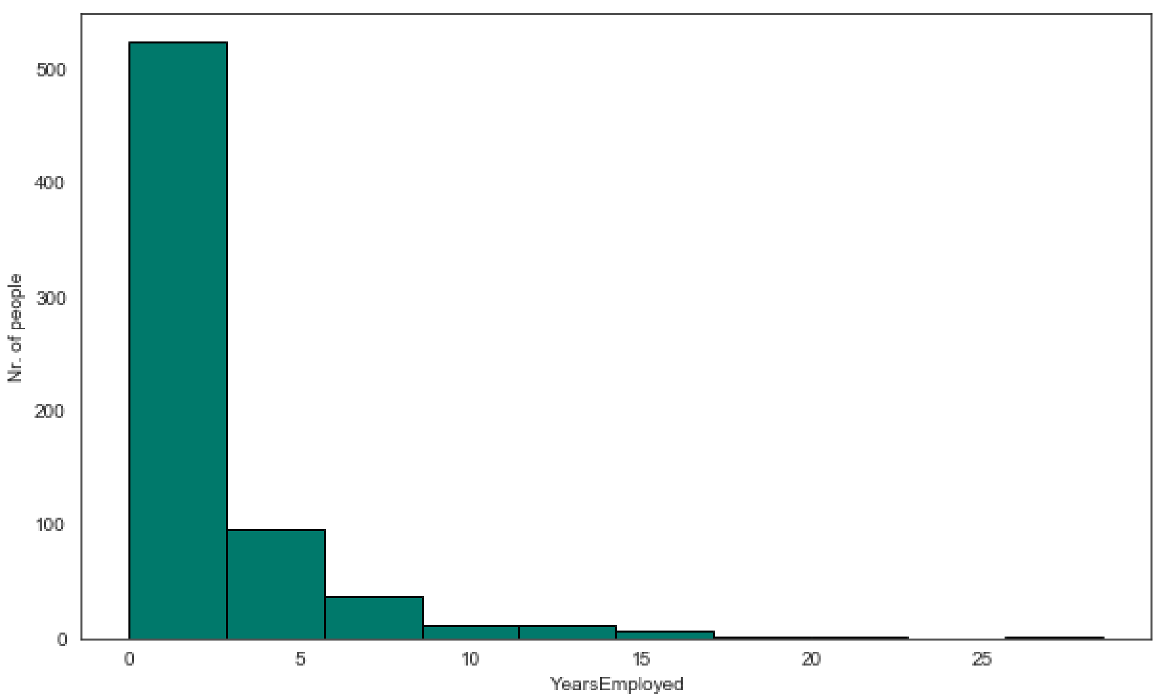
In [247]:

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



In [248]:

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



In [249]:

```
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 [250]:

```
dataset.head()
```

Out[250]:

	Age	Debt	EducationLevel	YearsEmployed	CreditScore	ZipCode	Income	Ap
0	0.056919	-0.956613	1.465680	-0.291083	-0.288101	0.178586	-0.195413	
1	1.846363	-0.060051	1.002409	0.244190	0.740830	-1.004068	-0.087852	
2	-0.640132	-0.856102	1.002409	-0.216324	-0.493887	0.759538	-0.037144	
3	-0.265597	-0.647038	1.465680	0.456505	0.535044	-0.589102	-0.194837	
4	-1.118704	0.174141	1.465680	-0.153526	-0.493887	-0.464612	-0.195413	

5 rows × 38 columns

In [251]:

```
dataset.describe()
```

Out[251]:

	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	3.226083e-17	1.605801e-16	8.688702e-18	1.673380e-16	1.153667e-15	3.700743e
std	1.000725e+00	1.000725e+00	1.000725e+00	1.000725e+00	1.000725e+00	1.000725e
min	-1.566065e+00	-9.566132e-01	-1.545577e+00	-6.648767e-01	-4.938866e-01	-1.232299e
25%	-8.273994e-01	-7.555902e-01	-1.082307e+00	-6.155359e-01	-4.938866e-01	-7.550880e
50%	-1.771653e-01	-4.037999e-01	-1.557662e-01	-3.658414e-01	-4.938866e-01	-1.533871e
75%	7.851813e-01	4.922602e-01	1.002409e+00	1.200908e-01	1.234717e-01	7.595383e
max	2.064842e+00	4.672031e+00	1.697315e+00	7.857628e+00	1.329378e+01	2.294913e

8 rows × 38 columns

In [346]:
```
dataset.corr() # Pearson Correlation Coefficients
```

Out[346]:

	Age	Debt	EducationLevel	YearsEmployed	CreditScore	ZipCode
Age	1.000000	0.135058	0.026752	0.386076	0.160599	-0.001211
Debt	0.135058	1.000000	0.027622	0.298902	0.271207	-0.262772
EducationLevel	0.026752	0.027622	1.000000	0.040598	0.009907	0.066057
YearsEmployed	0.386076	0.298902	0.040598	1.000000	0.322330	-0.106919
CreditScore	0.160599	0.271207	0.009907	0.322330	1.000000	-0.139028
ZipCode	-0.001211	-0.262772	0.066057	-0.106919	-0.139028	1.000000
Income	0.016829	0.123121	0.005907	0.051345	0.063692	0.088546
Approved	-0.133304	-0.206294	-0.129398	-0.322475	-0.406410	0.094851
sex_0	0.000652	-0.046288	-0.041267	-0.042040	-0.056580	0.055042
sex_1	-0.062296	0.041746	0.018081	-0.086544	0.024630	-0.097600
sex_2	0.061179	-0.028166	-0.006261	0.097009	-0.008427	0.080732
Married_0	0.011989	-0.089595	-0.144757	-0.062271	-0.046257	0.214938
Married_1	-0.073779	0.069678	-0.027132	0.044767	-0.026629	0.068359
Married_2	0.085961	0.093017	0.087078	0.082493	0.122543	-0.054940
Married_3	-0.080657	-0.083781	-0.053435	-0.075905	-0.111077	0.000211
BankCustomer_0	0.011989	-0.089595	-0.144757	-0.062271	-0.046257	0.214938
BankCustomer_1	0.085961	0.093017	0.087078	0.082493	0.122543	-0.054940
BankCustomer_2	-0.073779	0.069678	-0.027132	0.044767	-0.026629	0.068359
BankCustomer_3	-0.080657	-0.083781	-0.053435	-0.075905	-0.111077	0.000211
Ethnicity_0	0.044550	-0.084398	-0.177680	-0.074047	-0.056777	0.147478
Ethnicity_1	0.161263	-0.003667	-0.063237	0.074188	0.032423	0.077190
Ethnicity_2	-0.071484	0.010693	-0.036283	-0.046331	-0.023771	-0.018253
Ethnicity_3	0.134071	0.037302	-0.038206	-0.073321	-0.033368	-0.116373
Ethnicity_4	0.015040	0.061269	0.121021	0.178414	0.065464	0.035167
Ethnicity_5	-0.015103	-0.033528	0.014489	-0.062901	-0.020059	-0.041332
Ethnicity_6	-0.061194	-0.002532	0.045591	-0.009126	0.005500	-0.030329
Ethnicity_7	-0.000577	0.074880	-0.002154	-0.033147	-0.026629	0.012425
Ethnicity_8	-0.198155	-0.095540	0.005905	-0.143370	-0.049884	0.000200
Ethnicity_9	0.165464	0.203351	-0.041958	0.194173	0.096952	-0.118578
PriorDefault_0	-0.197493	-0.244317	-0.113752	-0.345689	-0.379532	0.096796
PriorDefault_1	0.197493	0.244317	0.113752	0.345689	0.379532	-0.096796
Employed_0	-0.047300	-0.174846	-0.132744	-0.222982	-0.571498	0.091529
Employed_1	0.047300	0.174846	0.132744	0.222982	0.571498	-0.091529
DriversLicense_0	-0.079829	0.013023	-0.077824	-0.138139	-0.006944	-0.137117

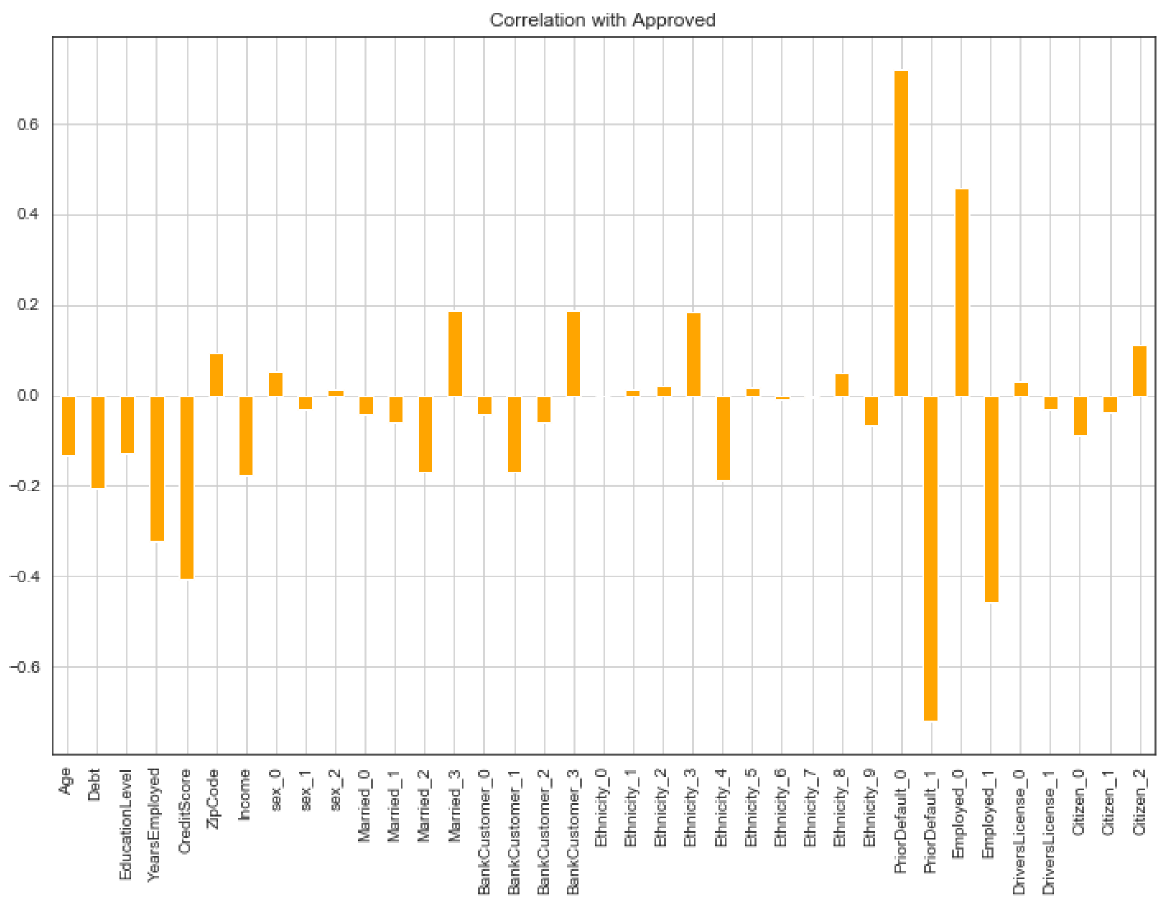
	Age	Debt	EducationLevel	YearsEmployed	CreditScore	ZipCode
DriversLicense_1	0.079829	-0.013023	0.077824	0.138139	0.006944	0.137117
Citizen_0	-0.000496	0.123569	0.043338	0.031670	0.142938	-0.133048
Citizen_1	-0.003976	-0.037842	-0.142308	-0.065938	-0.053491	0.203048
Citizen_2	0.002073	-0.116404	0.009353	-0.007965	-0.130871	0.062219

38 rows × 38 columns

In [347]:

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

Out[347]: <matplotlib.axes._subplots.AxesSubplot at 0x2cdb4dd4d88>



In [348]:

```
dataset['Approved'].corr(dataset['Age'])
```

Out[348]: -0.13330359179485982

In [349]:
```
dataset['Approved'].corr(dataset['Debt'])
```
Out[349]: -0.20629373864503894

In [350]:
```
dataset['Approved'].corr(dataset['EducationLevel'])
```
Out[350]: -0.12939760820234666

In [351]:
```
dataset['Approved'].corr(dataset['YearsEmployed'])
```
Out[351]: -0.3224753582553844

In [352]:
```
dataset['Approved'].corr(dataset['CreditScore'])
```
Out[352]: -0.4064100087639563

In [353]:
```
dataset['Approved'].corr(dataset['ZipCode'])
```
Out[353]: 0.09485112553643885

In [354]:
```
dataset['Approved'].corr(dataset['Income'])
```
Out[354]: -0.17565720099350488

Applying ML in one method

In [355]:
```
from sklearn.dummy import DummyRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
features =  list(dataset.drop(['Approved'],axis=1))
y = dataset.Approved
X = dataset[features]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_sta
dummy_median = DummyRegressor(strategy='mean')
dummy_regressor = dummy_median.fit(X_train,y_train)
dummy_predicts = dummy_regressor.predict(X_test)
print("Model Accuracy:", dummy_regressor.score(X_test,y_test)*100)
```

```
Model Accuracy: -0.06543803418803673
```

In [356]:
```
print(mean_absolute_error(y_test,dummy_predicts))
```

```
0.4931474480151227
```

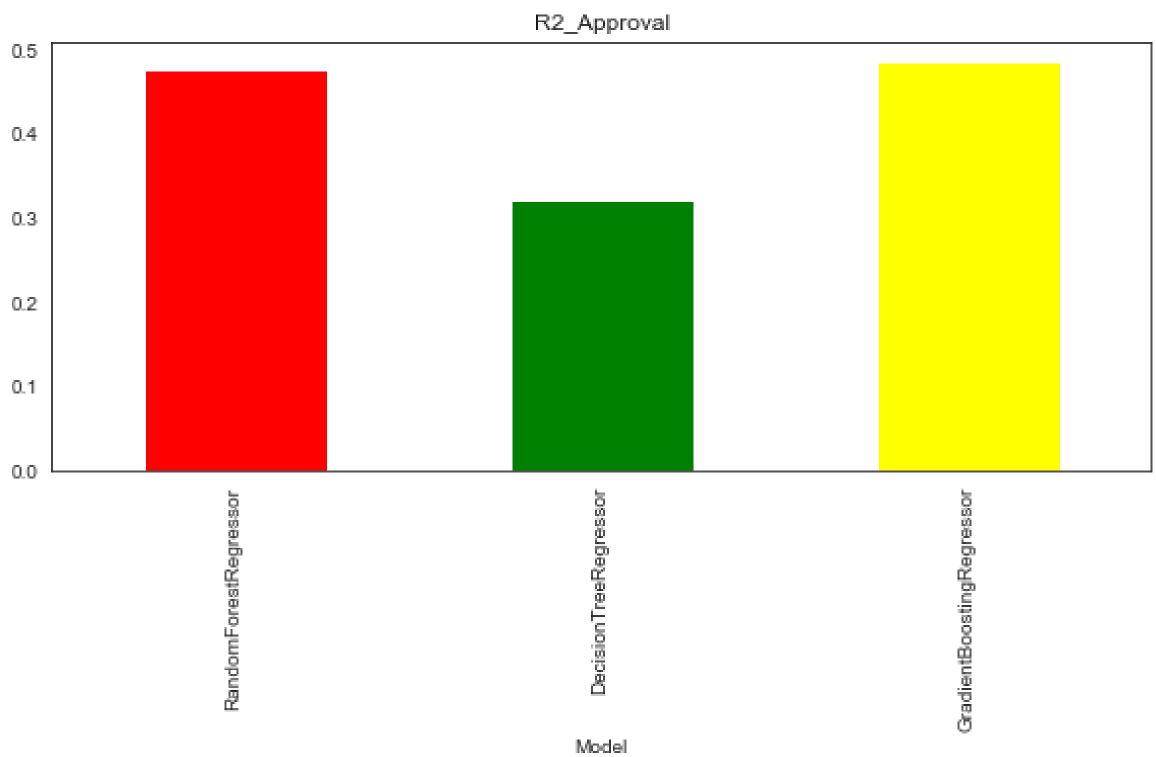
In [357]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
models = [RandomForestRegressor(n_estimators=200,criterion='mse',max_depth=20
learning_mods = pd.DataFrame()
temp = {}
```

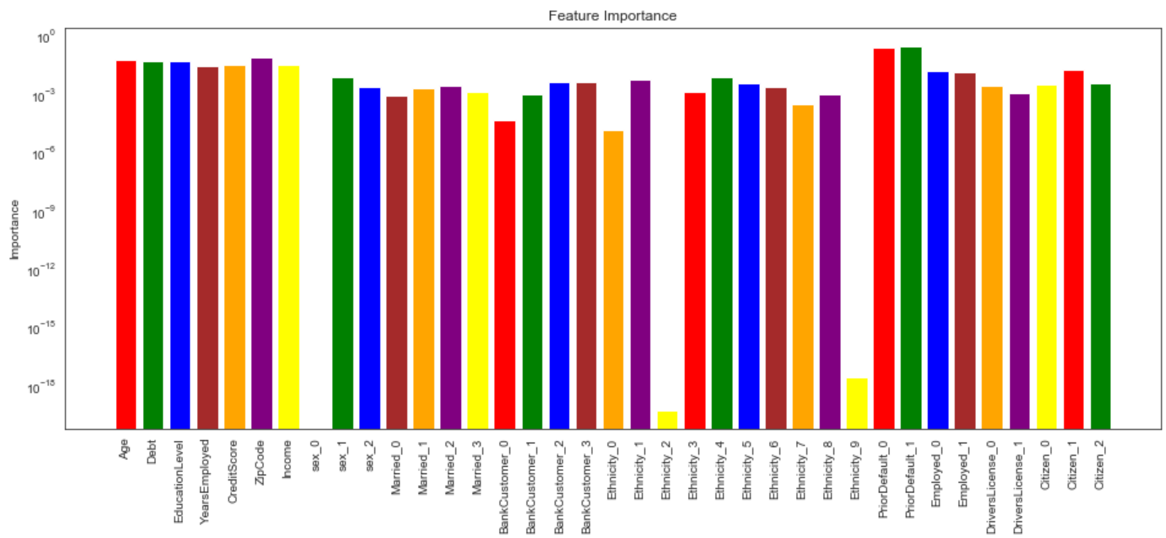
In [358]:

```
#run through models
for model in models:
    print(model)
    m = str(model)
    temp['Model'] = m[:m.index('(')]
    model.fit(X_train, y_train)
    temp['R2_Approval'] = r2_score(y_test, model.predict(X_test))
    print('score on training',model.score(X_train, y_train))
    print('R2_Approval',r2_score(y_test, model.predict(X_test)))
    learning_mods = learning_mods.append([temp])
learning_mods.set_index('Model', inplace=True)

fig, axes = plt.subplots(ncols=1, figsize=(10, 4))
learning_mods.R2_Approval.plot(ax=axes, kind='bar', title='R2_Approval',color
plt.show()
```

```
RandomForestRegressor(max_depth=20, n_estimators=200, random_state=100)
score on training 0.9472852273179797
R2_Approval 0.47645733974358984
DecisionTreeRegressor(max_depth=11, random_state=100)
score on training 0.9937194815538215
R2_Approval 0.32119309262166407
GradientBoostingRegressor(max_depth=12, n_estimators=200)
score on training 1.0
R2_Approval 0.4849700974748604
```



In [362]:
```
regressionTree_imp = model.feature_importances_
plt.figure(figsize=(16,6))
plt.yscale('log',nonposy='clip')
plt.bar(range(len(regressionTree_imp)),regressionTree_imp,align='center',colo
'purple','yellow'])
plt.xticks(range(len(regressionTree_imp)),features,rotation='vertical')
plt.title('Feature Importance')
plt.ylabel('Importance')
plt.show()
```



machine learning algorithms

In [280]:
```
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 [281]:
```
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 [282]:
```
from sklearn.linear_model import LogisticRegression

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

Out[282]:
```
LogisticRegression(solver='liblinear')
```

In [283]:

```
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: 89.65%
_______________________________________________
Classification Report:  Precision Score: 93.73%
                        Recall Score: 87.55%
                        F1 score: 90.53%
_______________________________________________
Confusion Matrix:
 [[194  16]
 [ 34 239]]

Test Result:
================================================
Accuracy Score: 85.02%
_______________________________________________
Classification Report:  Precision Score: 89.11%
                        Recall Score: 81.82%
                        F1 score: 85.31%
_______________________________________________
Confusion Matrix:
 [[86 11]
 [20 90]]
```

In [284]:

```
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_sco
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df
```

Out[284]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	89.648033	85.024155

K-nearest neighbors

In [285]:
```
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: 90.27%
_______________________________________________
Classification Report:  Precision Score: 90.65%
                        Recall Score: 92.31%
                        F1 score: 91.47%
_______________________________________________
Confusion Matrix:
 [[184  26]
 [ 21 252]]

Test Result:
================================================
Accuracy Score: 83.57%
_______________________________________________
Classification Report:  Precision Score: 81.67%
                        Recall Score: 89.09%
                        F1 score: 85.22%
_______________________________________________
Confusion Matrix:
 [[75 22]
 [12 98]]
```

In [286]:
```
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[286]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	89.648033	85.024155
1	K-nearest neighbors	90.269151	83.574879

Support Vector machine

In [287]:

```
from sklearn.svm import SVC


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

Out[287]:

```
SVC(gamma=0.1)
```

In [288]:

```
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: 91.30%
_______________________________________________
Classification Report:  Precision Score: 95.65%
                        Recall Score: 88.64%
                        F1 score: 92.02%
_______________________________________________
Confusion Matrix:
 [[199  11]
 [ 31 242]]

Test Result:
================================================
Accuracy Score: 87.92%
_______________________________________________
Classification Report:  Precision Score: 88.99%
                        Recall Score: 88.18%
                        F1 score: 88.58%
_______________________________________________
Confusion Matrix:
 [[85 12]
 [13 97]]
```

In [289]:

```
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[289]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	89.648033	85.024155
1	K-nearest neighbors	90.269151	83.574879
2	Support Vector Machine	91.304348	87.922705

Decision Tree Classifier

In [290]:

```
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: 80.68%
_______________________________________________
Classification Report:  Precision Score: 78.69%
                        Recall Score: 87.27%
                        F1 score: 82.76%
_______________________________________________
Confusion Matrix:
 [[71 26]
 [14 96]]
```

In [291]:

```
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[291]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	89.648033	85.024155
1	K-nearest neighbors	90.269151	83.574879
2	Support Vector Machine	91.304348	87.922705
3	Decision Tree Classifier	100.000000	80.676329

Random Forest

In [292]:

```
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: 85.99%
_______________________________________________
Classification Report:   Precision Score: 87.85%
                         Recall Score: 85.45%
                         F1 score: 86.64%
_______________________________________________
Confusion Matrix:
 [[84 13]
 [16 94]]
```

In [293]:

```
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[293]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	89.648033	85.024155
1	K-nearest neighbors	90.269151	83.574879
2	Support Vector Machine	91.304348	87.922705
3	Decision Tree Classifier	100.000000	80.676329
4	Random Forest Classifier	100.000000	85.990338

XGBoost Classifer

In [294]:

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

In [295]:

```
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: 85.51%
_______________________________________________
Classification Report:  Precision Score: 85.71%
                        Recall Score: 87.27%
                        F1 score: 86.49%
_______________________________________________
Confusion Matrix:
 [[81 16]
 [14 96]]
```

In [296]:

```
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[296]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	89.648033	85.024155
1	K-nearest neighbors	90.269151	83.574879
2	Support Vector Machine	91.304348	87.922705
3	Decision Tree Classifier	100.000000	80.676329
4	Random Forest Classifier	100.000000	85.990338
5	XGBoost Classifier	100.000000	85.507246

Using Hyperparameter Tuning

Logistic Regression Hyperparameter Tuning

In [297]:

```
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 [298]:

```
# grid_search_cv.best_estimator_
```

In [299]:

```
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: 89.44%
_______________________________________________
Classification Report:  Precision Score: 93.02%
                        Recall Score: 87.91%
                        F1 score: 90.40%
_______________________________________________
Confusion Matrix:
 [[192  18]
 [ 33 240]]

Test Result:
================================================
Accuracy Score: 85.02%
_______________________________________________
Classification Report:  Precision Score: 89.11%
                        Recall Score: 81.82%
                        F1 score: 85.31%
_______________________________________________
Confusion Matrix:
 [[86 11]
 [20 90]]
```

In [300]:

```
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[300]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	89.440994	85.024155

K-nearest neighbors Hyperparameter Tuning

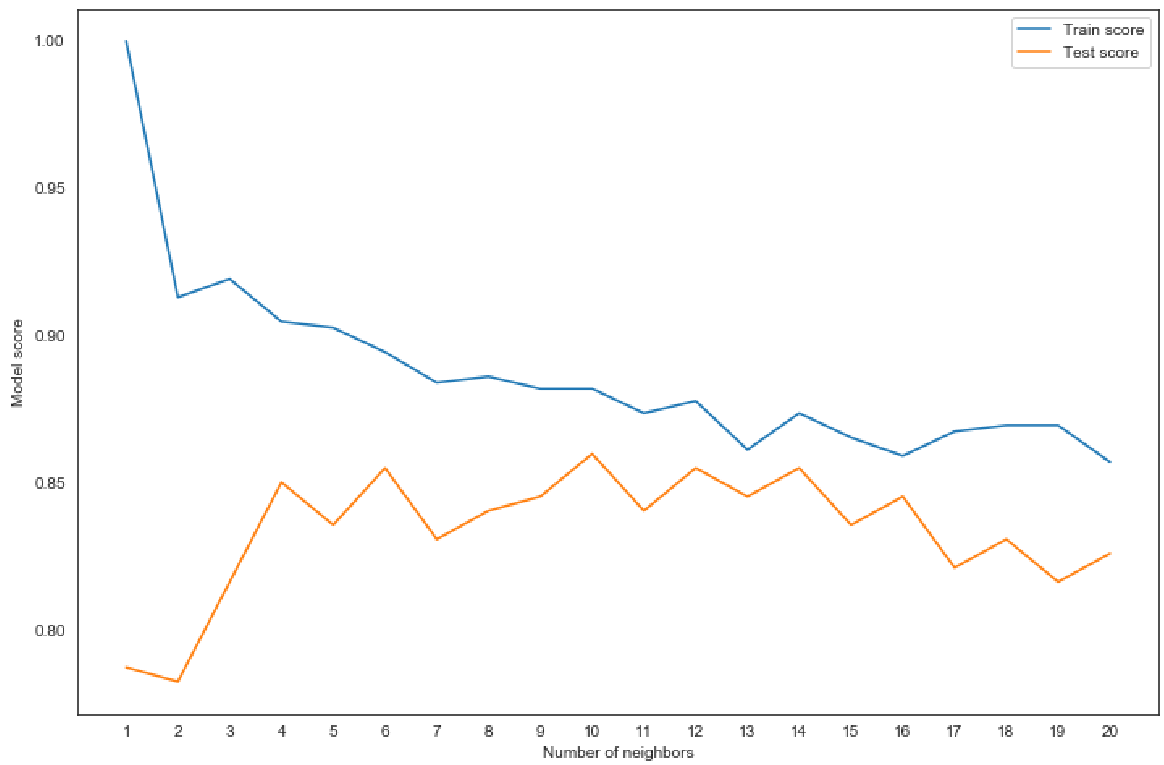
In [301]:

```
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 [302]:

```
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 [303]:

```
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: 86.96%
_______________________________________________
Classification Report:  Precision Score: 85.00%
                        Recall Score: 93.41%
                        F1 score: 89.01%
_______________________________________________
Confusion Matrix:
 [[165  45]
 [ 18 255]]

Test Result:
================================================
Accuracy Score: 81.64%
_______________________________________________
Classification Report:  Precision Score: 78.12%
                        Recall Score: 90.91%
                        F1 score: 84.03%
_______________________________________________
Confusion Matrix:
 [[ 69  28]
 [ 10 100]]
```

In [304]:

```
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[304]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	89.440994	85.024155
1	Tuned K-nearest neighbors	86.956522	81.642512

In [305]:

```
### Support Vector Machine Hyperparameter Tuning
```

In [306]:
```
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 [307]:
```
# svm_grid.best_estimator_
```

In [308]:
```
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: 95.63%
			Recall Score: 80.22%
			F1 score: 87.25%
_______________________________________________
Confusion Matrix:
 [[200  10]
 [ 54 219]]

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 [309]:
```
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[309]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	89.440994	85.024155
1	Tuned K-nearest neighbors	86.956522	81.642512
2	Tuned Support Vector Machine	86.749482	84.057971

Decision Tree Classifier Hyperparameter Tuning

In [310]:
```
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 [311]:
```
# grid_search_cv.best_estimator_
```

In [312]:

```
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: 87.58%
_______________________________________________
Classification Report:  Precision Score: 86.85%
                        Recall Score: 91.94%
                        F1 score: 89.32%
_______________________________________________
Confusion Matrix:
 [[172  38]
 [ 22 251]]

Test Result:
================================================
Accuracy Score: 82.61%
_______________________________________________
Classification Report:  Precision Score: 79.37%
                        Recall Score: 90.91%
                        F1 score: 84.75%
_______________________________________________
Confusion Matrix:
 [[ 71  26]
 [ 10 100]]
```

In [313]:

```
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[313]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	89.440994	85.024155
1	Tuned K-nearest neighbors	86.956522	81.642512
2	Tuned Support Vector Machine	86.749482	84.057971
3	Tuned Decision Tree Classifier	87.577640	82.608696

Random Forest Classifier Hyperparameter Tuning

In [314]:

```
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 [315]:

```
# rf_random.best_estimator_
```

In [316]:

```
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[316]:

```
RandomForestClassifier(max_depth=70, min_samples_leaf=4, min_samples_split=
10,
                       n_estimators=400)
```

In [317]:
```
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: 92.96%
_______________________________________________
Classification Report:	Precision Score: 94.76%
			Recall Score: 92.67%
			F1 score: 93.70%
_______________________________________________
Confusion Matrix:
 [[196  14]
 [ 20 253]]

Test Result:
================================================
Accuracy Score: 85.02%
_______________________________________________
Classification Report:	Precision Score: 87.62%
			Recall Score: 83.64%
			F1 score: 85.58%
_______________________________________________
Confusion Matrix:
 [[84 13]
 [18 92]]
```

In [318]:
```
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[318]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	89.440994	85.024155
1	Tuned K-nearest neighbors	86.956522	81.642512
2	Tuned Support Vector Machine	86.749482	84.057971
3	Tuned Decision Tree Classifier	87.577640	82.608696
4	Tuned Random Forest Classifier	92.960663	85.024155

XGBoost Classifier Hyperparameter Tuning

In [319]:

```
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 [320]:

```
# xgb_cv.best_estimator_
```

In [321]:

```
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[321]:

```
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 [322]:
```
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: 96.89%
_______________________________________________
Classification Report:  Precision Score: 96.74%
                        Recall Score: 97.80%
                        F1 score: 97.27%
_______________________________________________
Confusion Matrix:
 [[201   9]
 [  6 267]]

Test Result:
================================================
Accuracy Score: 85.99%
_______________________________________________
Classification Report:  Precision Score: 87.16%
                        Recall Score: 86.36%
                        F1 score: 86.76%
_______________________________________________
Confusion Matrix:
 [[83 14]
 [15 95]]
```

In [323]:
```
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[323]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	89.440994	85.024155
1	Tuned K-nearest neighbors	86.956522	81.642512
2	Tuned Support Vector Machine	86.749482	84.057971
3	Tuned Decision Tree Classifier	87.577640	82.608696
4	Tuned Random Forest Classifier	92.960663	85.024155
5	Tuned XGBoost Classifier	96.894410	85.990338

In [324]:
```
results_df
```

Out[324]:

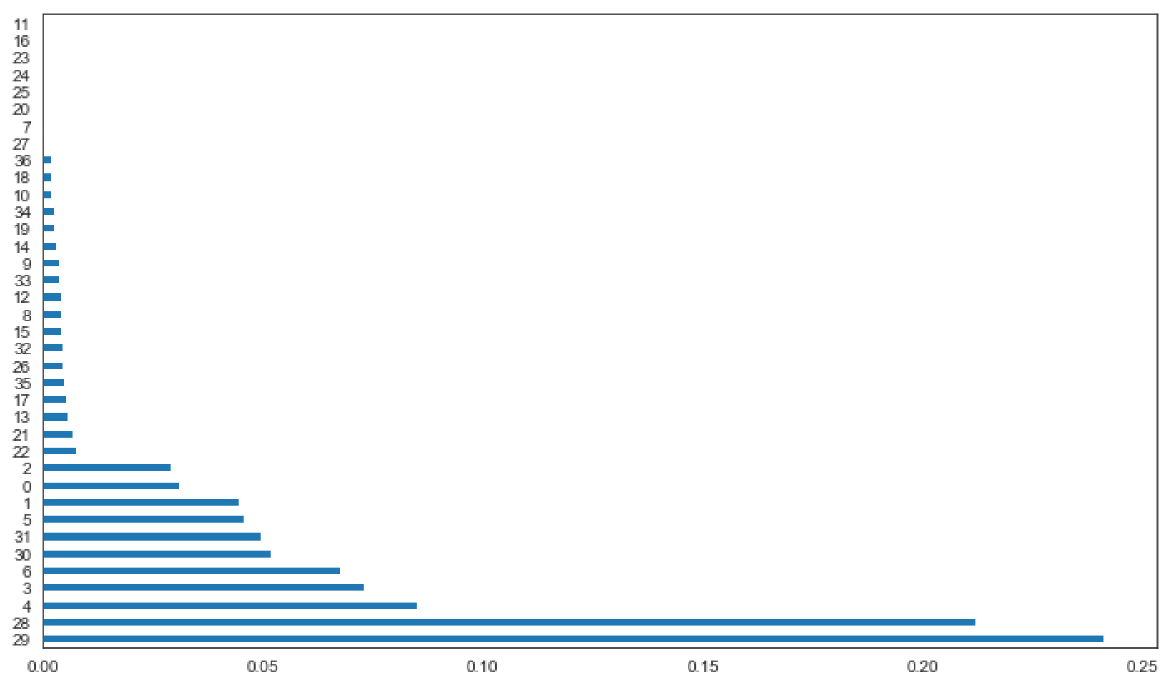
	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	89.648033	85.024155
1	K-nearest neighbors	90.269151	83.574879
2	Support Vector Machine	91.304348	87.922705
3	Decision Tree Classifier	100.000000	80.676329
4	Random Forest Classifier	100.000000	85.990338
5	XGBoost Classifier	100.000000	85.507246

Features Importance According to Random Forest and XGBoost

In [325]:
```
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 [326]:
```
feature_imp(X, rand_forest).plot(kind='barh', figsize=(12,7), legend=False)
```

Out[326]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2cdb3440308>
```



In [327]:
```
feature_imp(X, xgb_best).plot(kind='barh', figsize=(12,7), legend=False)
```

Out[327]: <matplotlib.axes._subplots.AxesSubplot at 0x2cdb15bbb48>