Credit Approval Project

Dataset Source --->> [https://archive.ics.uci.edu/ml/datasets/Credit+Approval (https://archive.ics.uci.edu/ml/datasets/Credit+Approval)](https://archive.ics.uci.edu/ml/datasets/Credit+Approval)

Project Problem Statement

In this particular project we need to find out based on the given features whether a Credit card Application will get an approval or not.Hence we can also say that it is a binary classification task which we need to do here

Importing required libraries

In [345]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

Reading data

In [346]:

```
credit_data = pd.read_csv(r'credit_data_final.csv.txt')
```

In [347]:

```
credit_data.head(10)
```

Out[347]:

	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	3.040	t	t	6	f	g	00043	560	+
1	a	24.50	0.500	u	g	q	h	1.500	t	f	0	f	g	00280	824	+
2	b	27.83	1.540	u	g	w	v	3.750	t	t	5	t	g	00100	3	+
3	b	20.17	5.625	u	g	w	v	1.710	t	f	0	f	s	00120	0	+
4	b	32.08	4.000	u	g	m	v	2.500	t	f	0	t	g	00360	0	+
5	b	33.17	1.040	u	g	r	h	6.500	t	f	0	t	g	00164	31285	+
6	a	22.92	11.585	u	g	cc	v	0.040	t	f	0	f	g	00080	1349	+
7	b	54.42	0.500	y	p	k	h	3.960	t	f	0	f	g	00180	314	+
8	b	42.50	4.915	y	p	w	v	3.165	t	f	0	t	g	00052	1442	+
9	b	22.08	0.830	u	g	c	h	2.165	f	f	0	t	g	00128	0	+

Information about the dataset

1. Title: Credit Approval
2. Sources: (confidential) Submitted by quinlan@cs.su.oz.au
3. Past Usage:

   See Quinlan,

   - "Simplifying decision trees", Int J Man-Machine Studies 27, Dec 1987, pp. 221-234.
   - "C4.5: Programs for Machine Learning", Morgan Kaufmann, Oct 1992
4. Relevant Information:

   This file concerns credit card applications. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data.

   This dataset is interesting because there is a good mix of attributes -- continuous, nominal with small numbers of values, and nominal with larger numbers of values. There are also a few missing values.
5. Number of Instances: 690
6. Number of Attributes: 15 + class attribute
7. Attribute Information:

   A1: b, a. A2: continuous. A3: continuous. A4: u, y, l, t. A5: g, p, gg. A6: c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff. A7: v, h, bb, j, n, z, dd, ff, o. A8: continuous. A9: t, f. A10: t, f. A11: continuous. A12: t, f. A13: g, p, s. A14: continuous. A15: continuous. A16: +,- (class attribute)
8. Missing Attribute Values: 37 cases (5%) have one or more missing values. The missing values from particular attributes are:

   A1: 12 A2: 12 A4: 6 A5: 6 A6: 9 A7: 9 A14: 13
9. Class Distribution

   +: 307 (44.5%) -: 383 (55.5%)

In [348]:

```
credit_data.head(2)
```

Out[348]:

	b	30.83	0	u	g	w	v	1.25	t	t.1	01	f	g.1	00202	0.1	+
0	a	58.67	4.46	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
1	a	24.50	0.50	u	g	q	h	1.50	t	f	0	f	g	00280	824	+

In [349]:

```
credit_data.shape
```

Out[349]:

```
(689, 16)
```

In [350]:

```
# replacing the approved values to 1 and unapproved to 0
credit_data.replace(to_replace ="+", value = 1 , inplace = True)
credit_data.replace(to_replace ="-", value = 0 , inplace = True)
```

In [351]:

```
credit_data.head(10)
```

Out[351]:

	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	3.040	t	t	6	f	g	00043	560	1
1	a	24.50	0.500	u	g	q	h	1.500	t	f	0	f	g	00280	824	1
2	b	27.83	1.540	u	g	w	v	3.750	t	t	5	t	g	00100	3	1
3	b	20.17	5.625	u	g	w	v	1.710	t	f	0	f	s	00120	0	1
4	b	32.08	4.000	u	g	m	v	2.500	t	f	0	t	g	00360	0	1
5	b	33.17	1.040	u	g	r	h	6.500	t	f	0	t	g	00164	31285	1
6	a	22.92	11.585	u	g	cc	v	0.040	t	f	0	f	g	00080	1349	1
7	b	54.42	0.500	y	p	k	h	3.960	t	f	0	f	g	00180	314	1
8	b	42.50	4.915	y	p	w	v	3.165	t	f	0	t	g	00052	1442	1
9	b	22.08	0.830	u	g	c	h	2.165	f	f	0	t	g	00128	0	1

In [352]:

```
# changing the abrupt column names to simple column number names
credit_data.columns =list( i for i in range(16))
```

In [353]:

```
credit_data.head(10)
```

Out[353]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	a	58.67	4.460	u	g	q	h	3.040	t	t	6	f	g	00043	560	1
1	a	24.50	0.500	u	g	q	h	1.500	t	f	0	f	g	00280	824	1
2	b	27.83	1.540	u	g	w	v	3.750	t	t	5	t	g	00100	3	1
3	b	20.17	5.625	u	g	w	v	1.710	t	f	0	f	s	00120	0	1
4	b	32.08	4.000	u	g	m	v	2.500	t	f	0	t	g	00360	0	1
5	b	33.17	1.040	u	g	r	h	6.500	t	f	0	t	g	00164	31285	1
6	a	22.92	11.585	u	g	cc	v	0.040	t	f	0	f	g	00080	1349	1
7	b	54.42	0.500	y	p	k	h	3.960	t	f	0	f	g	00180	314	1
8	b	42.50	4.915	y	p	w	v	3.165	t	f	0	t	g	00052	1442	1
9	b	22.08	0.830	u	g	c	h	2.165	f	f	0	t	g	00128	0	1

In [354]:

```
# changing the column name of 15 to approval
credit_data.rename(columns = {15: "Approval"}, inplace = True)
```

In [355]:

```
credit_data.head(10)
```

Out[355]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Approval
0	a	58.67	4.460	u	g	q	h	3.040	t	t	6	f	g	00043	560	1
1	a	24.50	0.500	u	g	q	h	1.500	t	f	0	f	g	00280	824	1
2	b	27.83	1.540	u	g	w	v	3.750	t	t	5	t	g	00100	3	1
3	b	20.17	5.625	u	g	w	v	1.710	t	f	0	f	s	00120	0	1
4	b	32.08	4.000	u	g	m	v	2.500	t	f	0	t	g	00360	0	1
5	b	33.17	1.040	u	g	r	h	6.500	t	f	0	t	g	00164	31285	1
6	a	22.92	11.585	u	g	cc	v	0.040	t	f	0	f	g	00080	1349	1
7	b	54.42	0.500	y	p	k	h	3.960	t	f	0	f	g	00180	314	1
8	b	42.50	4.915	y	p	w	v	3.165	t	f	0	t	g	00052	1442	1
9	b	22.08	0.830	u	g	c	h	2.165	f	f	0	t	g	00128	0	1

In [356]:

```
credit_data[3].value_counts()
```

Out[356]:

```
u    518
y    163
?      6
l      2
Name: 3, dtype: int64
```

In [357]:

```
credit_data[4].value_counts()
```

Out[357]:

```
g     518
p     163
?       6
gg      2
Name: 4, dtype: int64
```

In [358]:

```
credit_data[5].value_counts()
```

Out[358]:

```
c     137
q      78
w      63
i      59
aa     54
ff     53
k      51
cc     41
m      38
x      38
d      30
e      25
j      10
?       9
r       3
Name: 5, dtype: int64
```

In [359]:

```
credit_data[6].value_counts()
```

Out[359]:

```
v     398
h     138
bb     59
ff     57
?       9
j       8
z       8
dd      6
n       4
o       2
Name: 6, dtype: int64
```

Replacing the absurd values in our dataset with NAN and replacing the missing values with the mean or meadian values of that particular column.

In [360]:

```
# we have lot of "?" values in most of the columns of our dataset hence trying to repla
ce them
credit_data = credit_data.replace(['?'],np.NaN)
```

In [361]:

```
pd.isna(credit_data).count()
```

Out[361]:

```
0           689
1           689
2           689
3           689
4           689
5           689
6           689
7           689
8           689
9           689
10          689
11          689
12          689
13          689
14          689
Approval    689
dtype: int64
```

In [362]:

```
credit_data.shape
```

Out[362]:

```
(689, 16)
```

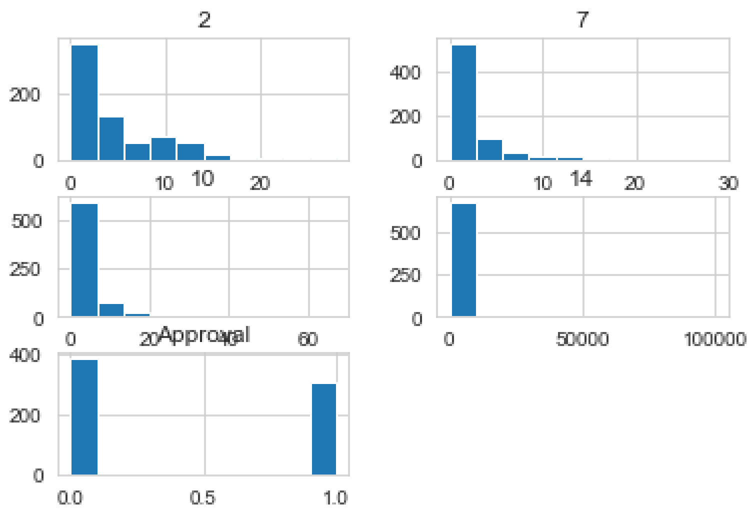
It means that 383 applications for the credit card were not approved while 306 were approved.

In [363]:

```
# plotting histogram for each column



credit_data.hist()
```

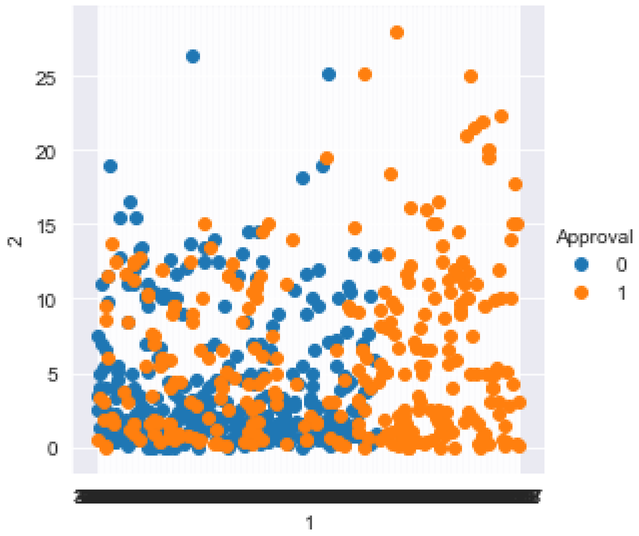
Out[363]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001DF07544CF
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF076ED2B
0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001DF08D8990
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF07572F2
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001DF070975C
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF05468C1
8>]],
      dtype=object)
```



In [364]:

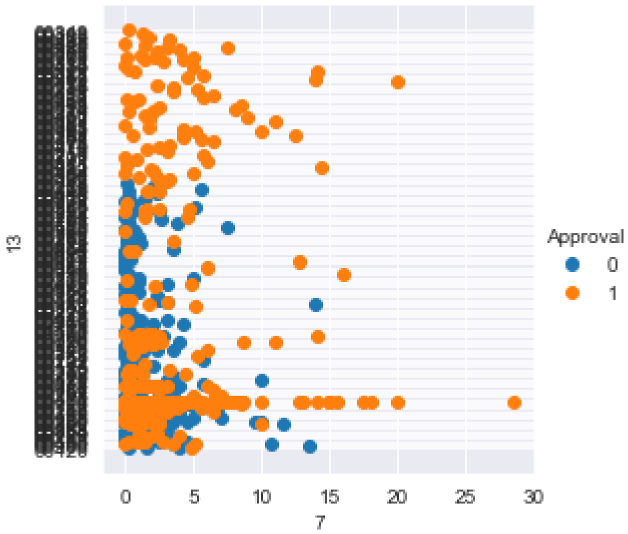
```
sns.set_style("darkgrid");
sns.FacetGrid(credit_data, hue="Approval", height=4).map(plt.scatter, 1, 2).add_legend
();
plt.show()
```



From above we can see that data points with higher values of column 1 are all accepted and having value of column 2 as low increases the chances of a project approval hence there are more chances of getting accepted if value in column 1 is higher and column 2 is lowe

In [365]:

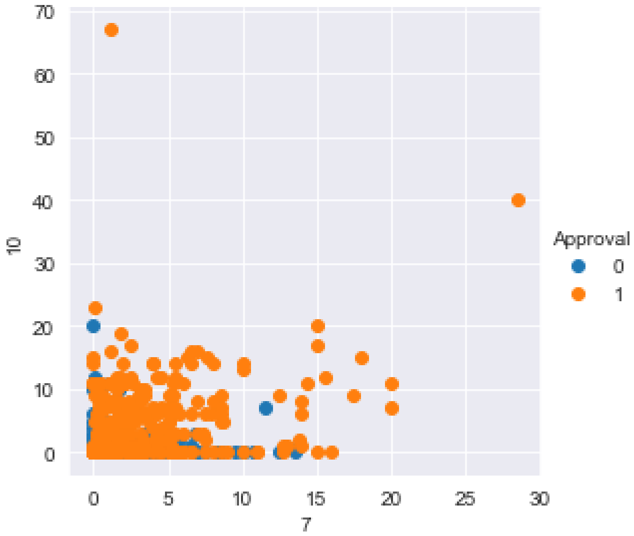
```
sns.set_style("darkgrid");
sns.FacetGrid(credit_data, hue="Approval", height=4).map(plt.scatter, 7, 13).add_legend
();
plt.show()
```



From above we can say that lower values of column 7 have highest chances of having the credit card approved and nothing special can be said about column 13.

In [366]:

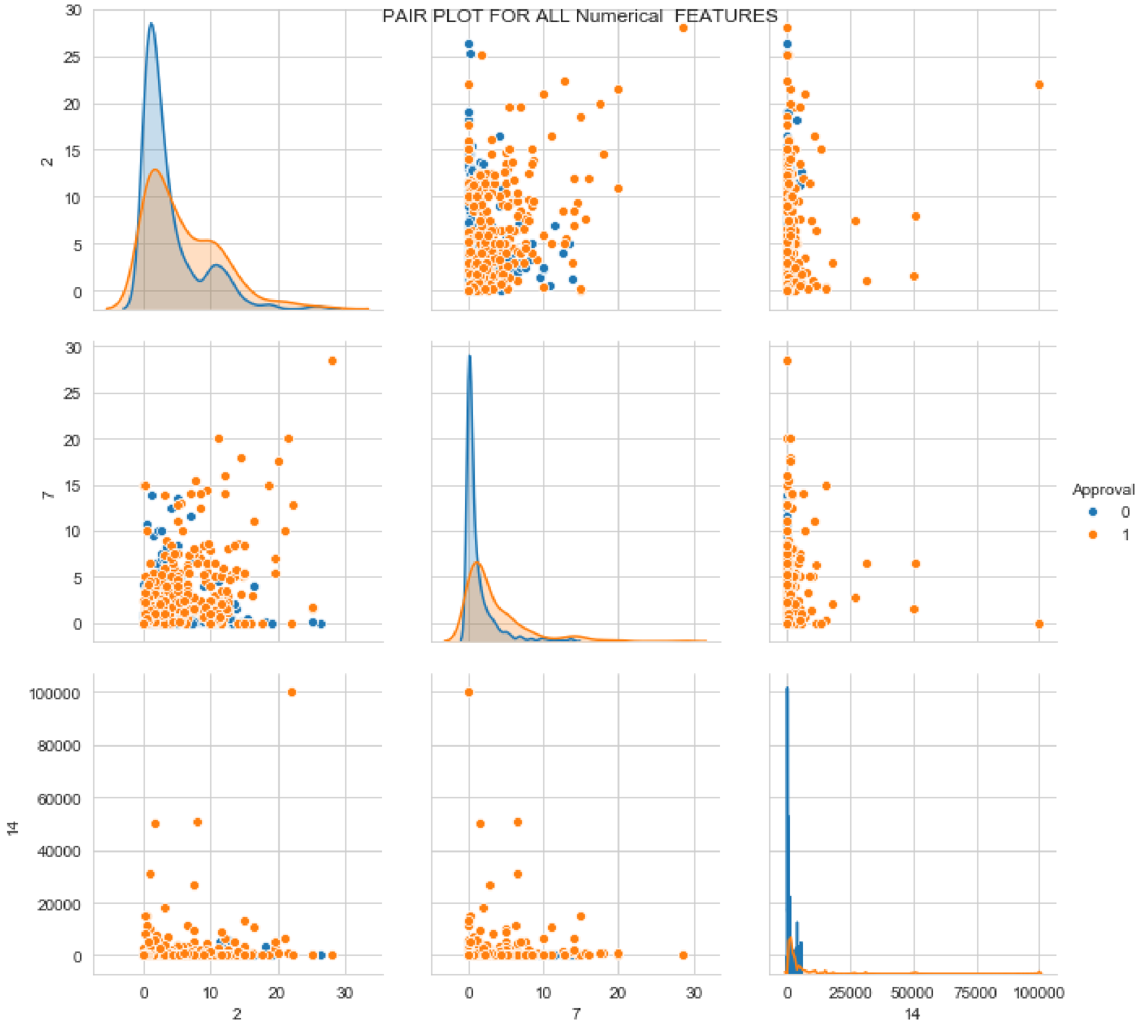
```
sns.set_style("darkgrid");
sns.FacetGrid(credit_data, hue="Approval", height=4).map(plt.scatter, 7, 10).add_legend
();
plt.show()
```



Let us try to draw the pair plots for numerical columns

In [367]:

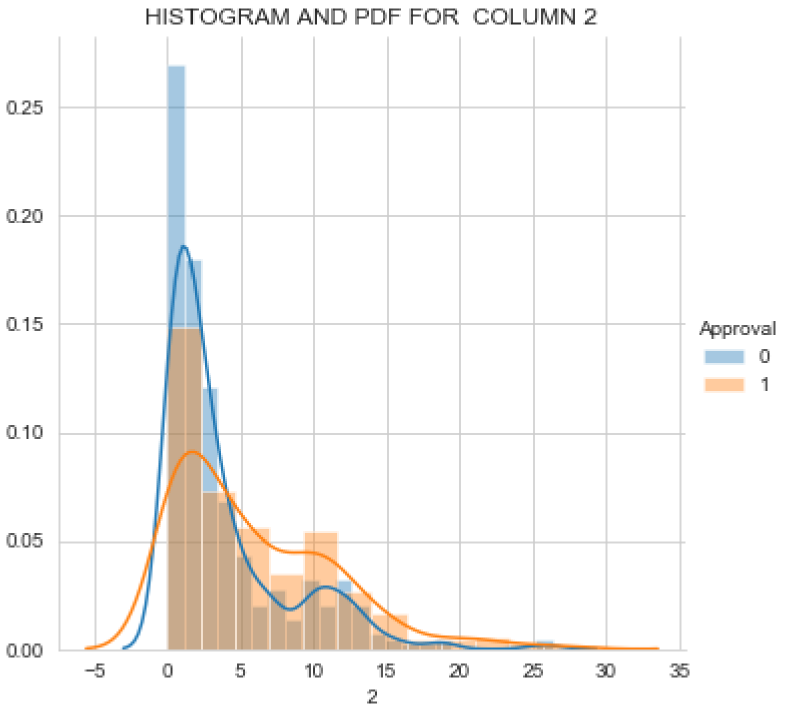
```
plt.close();
sns.set_style("whitegrid");
sns.pairplot(credit_data, hue='Approval', height=3 , vars = [2,7,14]);
plt.suptitle('PAIR PLOT FOR ALL Numerical  FEATURES')
plt.show()
```



Let us draw the histogram to get the idea about the data

In [368]:

```
sns.FacetGrid(credit_data, hue="Approval", height=5).map(sns.distplot, 2).add_legend();
plt.title("HISTOGRAM AND PDF FOR  COLUMN 2")

plt.show();
```

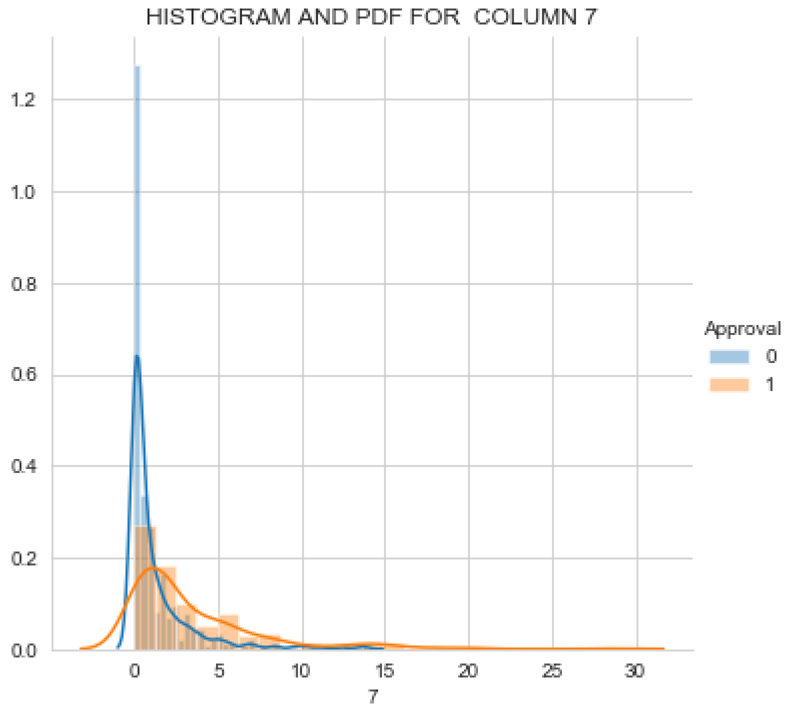


From above it can be seen that most of the non approvals are for those datasets which have column 2 values -1 to 5

While the number of approvals are higher for people having column 2 value between 5 to 20 hence column 2 might stand for the salary etc.

In [369]:

```
sns.FacetGrid(credit_data, hue="Approval", height=5).map(sns.distplot, 7).add_legend();
plt.title("HISTOGRAM AND PDF FOR  COLUMN 7")

plt.show();
```

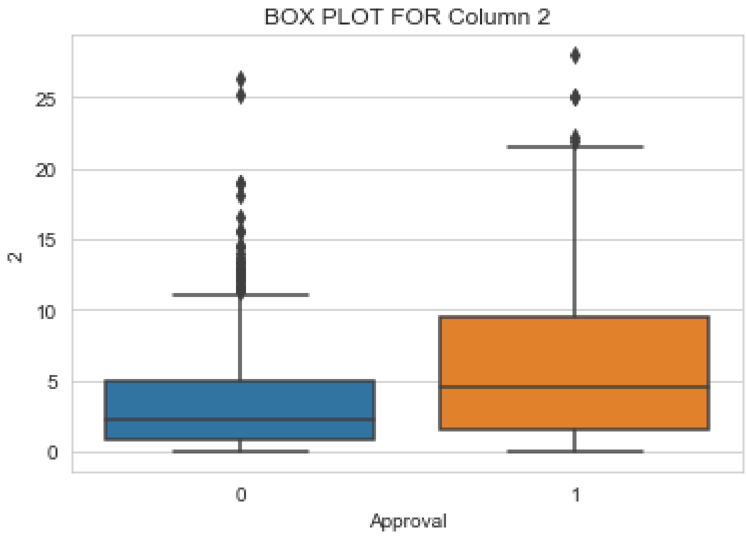


Column 7 is also behaving similar to column 2 having highest approvals between 2-10.

Box plot for above data

In [370]:

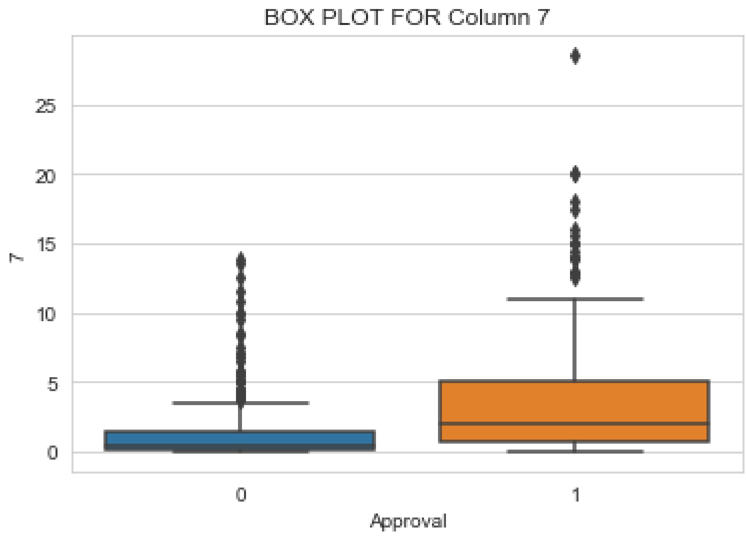
```
sns.boxplot(x='Approval',y=2, data=credit_data)
plt.title("BOX PLOT FOR Column 2")
plt.show()
```



There are lot of outliers values for the data points which are non approved.

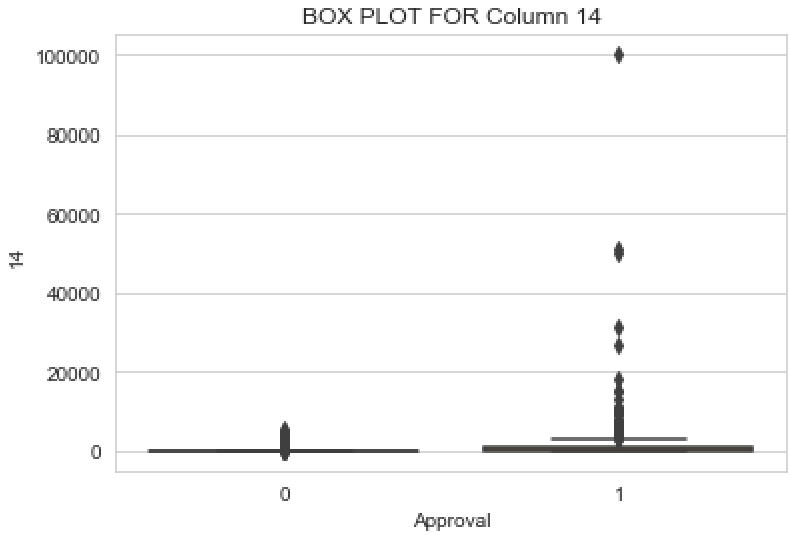
In [371]:

```
sns.boxplot(x='Approval',y=7, data=credit_data)
plt.title("BOX PLOT FOR Column 7")
plt.show()
```



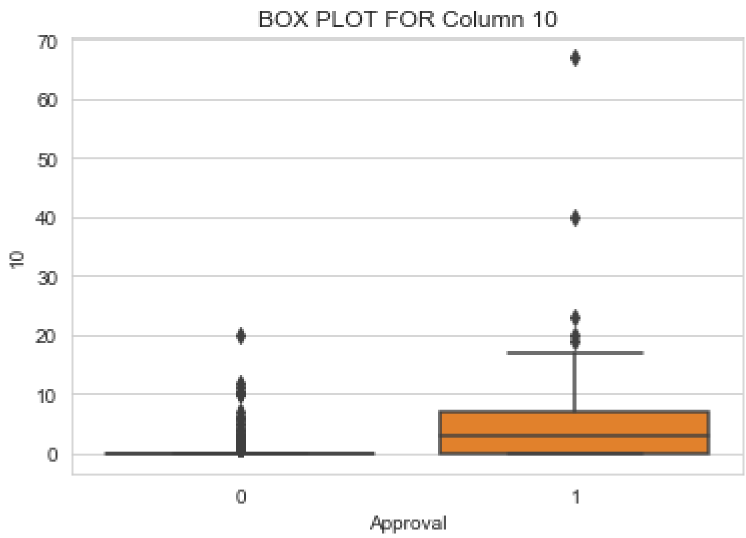
In [372]:

```
sns.boxplot(x='Approval',y=14, data=credit_data)
plt.title("BOX PLOT FOR Column 14")
plt.show()
```



In [373]:

```
sns.boxplot(x='Approval',y=10, data=credit_data)
plt.title("BOX PLOT FOR Column 10")
plt.show()
```



In [374]:

```
credit_data[2].describe()
```

Out[374]:

```
count    689.000000
mean       4.765631
std        4.978470
min        0.000000
25%        1.000000
50%        2.750000
75%        7.250000
max       28.000000
Name: 2, dtype: float64
```

In [375]:

```
credit_data[7].describe()
```

Out[375]:

```
count    689.000000
mean       2.224819
std        3.348739
min        0.000000
25%        0.165000
50%        1.000000
75%        2.625000
max       28.500000
Name: 7, dtype: float64
```

In [376]:

```
credit_data[14].describe()
```

Out[376]:

```
count       689.000000
mean       1018.862119
std        5213.743149
min           0.000000
25%           0.000000
50%           5.000000
75%         396.000000
max      100000.000000
Name: 14, dtype: float64
```

In [377]:

```
credit_data.dtypes
```

Out[377]:

```
0             object
1             object
2            float64
3             object
4             object
5             object
6             object
7            float64
8             object
9             object
10             int64
11            object
12            object
13            object
14             int64
Approval       int64
dtype: object
```

Now most of our columns have an object data type hence we need to convert it into numeric type oyherwise no ML algorithm will work on it until then.

In [378]:

```
credit_data.shape
```

Out[378]:

```
(689, 16)
```

In [379]:

```
# we will use the label encoder for encoding into numerical type
# reference --- > https://scikit-learn.org/stable/modules/generated/sklearn.preprocessi
ng.LabelEncoder.html
from sklearn import preprocessing

label_en = preprocessing.LabelEncoder()
```

In [380]:

```
# we will try to find all the columns which have an ibject as data type and use the enc
oding over them

for i in credit_data:
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Approval
```

In [381]:

```
# from above we can see that in this manner we can get the column names one by one

#for column_name in credit_data:
#    if credit_data[column_name].dtype == "object":
#        credit_data[column_name] = label_en.fit_transform(credit_data[column_name])





# running the above code gave me the error -- >> '<' not supported between instances of
'str' and 'float'

# reference --> https://discuss.analyticsvidhya.com/t/getting-typeerror-not-supported-b
etween-instances-of-str-and-float/18535

# hence  now dealing with the missing values
```

In [382]:

```
# replacing the missing values with mean

credit_data.fillna(credit_data.mean(), inplace = True)
```

In [383]:

```
credit_data.shape
```

Out[383]:

```
(689, 16)
```

In [385]:

```
# filling all the non numeric values now
# https://stackoverflow.com/questions/27905295/how-to-replace-nans-by-preceding-values-in-pandas-dataframe

credit_data=credit_data.fillna(method='bfill')
```

In [386]:

```
credit_data.shape
```

Out[386]:

```
(689, 16)
```

In [387]:

```
# again trying the same function which was giving error before

for column_name in credit_data:
    if credit_data[column_name].dtype == "object":
        credit_data[column_name] = label_en.fit_transform(credit_data[column_name])
```

In [388]:

```
credit_data.dtypes
```

Out[388]:

```
0             int32
1             int32
2           float64
3             int32
4             int32
5             int32
6             int32
7           float64
8             int32
9             int32
10            int64
11            int32
12            int32
13            int32
14            int64
Approval      int64
dtype: object
```

In [389]:

```
type(credit_data)
```

Out[389]:

```
pandas.core.frame.DataFrame
```

In [390]:

```
credit_data.shape
```

Out[390]:

```
(689, 16)
```

In [391]:

```
credit_data.head(40)
```

Out[391]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Approval
0	0	327	4.460	1	0	10	3	3.040	1	1	6	0	0	11	560	1
1	0	89	0.500	1	0	10	3	1.500	1	0	0	0	0	95	824	1
2	1	125	1.540	1	0	12	7	3.750	1	1	5	1	0	31	3	1
3	1	43	5.625	1	0	12	7	1.710	1	0	0	0	2	37	0	1
4	1	167	4.000	1	0	9	7	2.500	1	0	0	1	0	114	0	1
5	1	178	1.040	1	0	11	3	6.500	1	0	0	1	0	54	31285	1
6	0	74	11.585	1	0	2	7	0.040	1	0	0	0	0	23	1349	1
7	1	309	0.500	2	2	8	3	3.960	1	0	0	0	0	62	314	1
8	1	254	4.915	2	2	12	7	3.165	1	0	0	1	0	15	1442	1
9	1	64	0.830	1	0	1	3	2.165	0	0	0	1	0	39	0	1
10	1	145	1.835	1	0	1	3	4.335	1	0	0	0	0	89	200	1
11	0	219	6.000	1	0	8	7	1.000	1	0	0	1	0	0	0	1
12	1	281	6.040	1	0	8	7	0.040	0	0	0	0	0	0	2690	1
13	0	269	10.500	1	0	10	7	5.000	1	1	7	1	0	0	0	1
14	1	210	4.415	2	2	8	7	0.250	1	1	10	1	0	104	0	1
15	1	129	0.875	1	0	9	7	0.960	1	1	3	1	0	126	0	1
16	0	78	5.875	1	0	10	7	3.170	1	1	10	0	0	37	245	1
17	1	61	0.250	1	0	3	3	0.665	1	0	0	1	0	0	0	1
18	0	34	8.585	1	0	2	3	0.750	1	1	7	0	0	29	0	1
19	1	94	11.250	1	0	1	7	2.500	1	1	17	0	0	67	1208	1
20	1	78	1.000	1	0	1	7	0.835	1	0	0	0	2	99	0	1
21	0	279	8.000	1	0	1	7	7.875	1	1	6	1	0	0	1260	1
22	0	121	14.500	1	0	13	3	3.085	1	1	1	0	0	37	11	1
23	0	243	6.500	1	0	10	7	0.500	1	1	3	1	0	47	0	1
24	0	3	0.585	1	0	1	3	1.500	1	1	2	0	0	31	0	1
25	0	273	13.000	1	0	6	0	5.165	1	1	9	1	0	0	0	1
26	1	317	18.500	1	0	3	0	15.000	1	1	17	1	0	0	0	1
27	1	321	8.500	1	0	4	3	7.000	1	1	3	0	0	0	0	1
28	1	251	1.040	1	0	12	7	5.000	1	1	6	1	0	149	10000	1
29	1	138	14.790	1	0	0	7	5.040	1	1	5	1	0	56	0	1
30	1	250	9.790	1	0	13	3	7.960	1	1	8	0	0	0	0	1
31	1	290	7.585	1	0	6	0	7.585	1	1	15	1	0	0	5000	1
32	0	211	5.125	1	0	4	7	5.000	1	0	0	1	0	0	4000	1
33	0	70	10.750	1	0	10	7	0.415	1	1	5	1	0	0	560	1
34	1	125	1.500	1	0	12	7	2.000	1	1	11	1	0	137	35	1
35	1	119	1.585	1	0	2	3	1.835	1	1	12	1	0	157	713	1
36	0	75	11.750	1	0	13	3	0.500	1	1	2	1	0	99	551	1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Approval
37	1	124	0.585	2	2	2	7	0.250	1	1	2	0	0	89	500	1
38	1	310	9.415	1	0	5	2	14.415	1	1	11	1	0	8	300	1
39	1	187	9.170	1	0	1	7	4.500	1	1	12	1	0	0	221	1

in our column 14 it can be seen that the values are very much deflecting and doesnot give a proper information hence lets try to drop this column

In [392]:

```
credit_data = credit_data.drop([14], axis=1)
```

In [393]:

```
credit_data.head()
```

Out[393]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	Approval
0	0	327	4.460	1	0	10	3	3.04	1	1	6	0	0	11	1
1	0	89	0.500	1	0	10	3	1.50	1	0	0	0	0	95	1
2	1	125	1.540	1	0	12	7	3.75	1	1	5	1	0	31	1
3	1	43	5.625	1	0	12	7	1.71	1	0	0	0	2	37	1
4	1	167	4.000	1	0	9	7	2.50	1	0	0	1	0	114	1

In [394]:

```
credit_data.shape
```

Out[394]:

```
(689, 15)
```

Let us standardise the data and bring everything to same scale

In [395]:

```
p = credit_data["Approval"]
```

In [396]:

```
credit_data.drop(['Approval'], axis=1, inplace=True)
```

In [397]:

```
from sklearn import preprocessing

x = credit_data.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
credit_data = pd.DataFrame(x_scaled)
```

In [398]:

```
credit_data.head(20)
```

Out[398]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.0	0.942363	0.159286	0.5	0.0	0.769231	0.375	0.106667	1.0	1.0	0.089552	0.0	0.0
1	0.0	0.256484	0.017857	0.5	0.0	0.769231	0.375	0.052632	1.0	0.0	0.000000	0.0	0.0
2	1.0	0.360231	0.055000	0.5	0.0	0.923077	0.875	0.131579	1.0	1.0	0.074627	1.0	0.0
3	1.0	0.123919	0.200893	0.5	0.0	0.923077	0.875	0.060000	1.0	0.0	0.000000	0.0	1.0
4	1.0	0.481268	0.142857	0.5	0.0	0.692308	0.875	0.087719	1.0	0.0	0.000000	1.0	0.0
5	1.0	0.512968	0.037143	0.5	0.0	0.846154	0.375	0.228070	1.0	0.0	0.000000	1.0	0.0
6	0.0	0.213256	0.413750	0.5	0.0	0.153846	0.875	0.001404	1.0	0.0	0.000000	0.0	0.0
7	1.0	0.890490	0.017857	1.0	1.0	0.615385	0.375	0.138947	1.0	0.0	0.000000	0.0	0.0
8	1.0	0.731988	0.175536	1.0	1.0	0.923077	0.875	0.111053	1.0	0.0	0.000000	1.0	0.0
9	1.0	0.184438	0.029643	0.5	0.0	0.076923	0.375	0.075965	0.0	0.0	0.000000	1.0	0.0
10	1.0	0.417867	0.065536	0.5	0.0	0.076923	0.375	0.152105	1.0	0.0	0.000000	0.0	0.0
11	0.0	0.631124	0.214286	0.5	0.0	0.615385	0.875	0.035088	1.0	0.0	0.000000	1.0	0.0
12	1.0	0.809798	0.215714	0.5	0.0	0.615385	0.875	0.001404	0.0	0.0	0.000000	0.0	0.0
13	0.0	0.775216	0.375000	0.5	0.0	0.769231	0.875	0.175439	1.0	1.0	0.104478	1.0	0.0
14	1.0	0.605187	0.157679	1.0	1.0	0.615385	0.875	0.008772	1.0	1.0	0.149254	1.0	0.0
15	1.0	0.371758	0.031250	0.5	0.0	0.692308	0.875	0.033684	1.0	1.0	0.044776	1.0	0.0
16	0.0	0.224784	0.209821	0.5	0.0	0.769231	0.875	0.111228	1.0	1.0	0.149254	0.0	0.0
17	1.0	0.175793	0.008929	0.5	0.0	0.230769	0.375	0.023333	1.0	0.0	0.000000	1.0	0.0
18	0.0	0.097983	0.306607	0.5	0.0	0.153846	0.375	0.026316	1.0	1.0	0.104478	0.0	0.0
19	1.0	0.270893	0.401786	0.5	0.0	0.076923	0.875	0.087719	1.0	1.0	0.253731	0.0	0.0

In [399]:

```
credit_data.shape
```

Out[399]:

```
(689, 14)
```

In [400]:

```
credit_data["Approval"]=p
```

In [401]:

```
credit_data.shape
```

Out[401]:

```
(689, 15)
```

In [402]:

```
credit_data.head()
```

Out[402]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.0	0.942363	0.159286	0.5	0.0	0.769231	0.375	0.106667	1.0	1.0	0.089552	0.0	0.0
1	0.0	0.256484	0.017857	0.5	0.0	0.769231	0.375	0.052632	1.0	0.0	0.000000	0.0	0.0
2	1.0	0.360231	0.055000	0.5	0.0	0.923077	0.875	0.131579	1.0	1.0	0.074627	1.0	0.0
3	1.0	0.123919	0.200893	0.5	0.0	0.923077	0.875	0.060000	1.0	0.0	0.000000	0.0	1.0
4	1.0	0.481268	0.142857	0.5	0.0	0.692308	0.875	0.087719	1.0	0.0	0.000000	1.0	0.0

Train Test Split

In [403]:

```
# note that here This stratify parameter makes a split so that the proportion of values
in the sample produced will be the same as the proportion of values provided to paramet
er stratify.
#For example, if variable y is a binary categorical variable with values 0 and 1 and th
ere are 25% of zeros and 75% of ones, stratify=y will make sure that your random split
 has 25% of 0's and 75% of 1's.

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(credit_data, credit_data['Approval'
], test_size=0.33, stratify = credit_data['Approval'])
```

In [404]:

```
# Now we will be removing the column "Approval" because that is the only one which our
 model needs to predict

X_train.drop(['Approval'], axis=1, inplace=True)
X_test.drop(['Approval'], axis=1, inplace=True)
```

In [405]:

```
X_train.shape
```

Out[405]:

```
(461, 14)
```

In [406]:

```
X_test.shape
```

Out[406]:

```
(228, 14)
```

In [407]:

```
y_train.shape
```

Out[407]:

```
(461,)
```

In [408]:

```
y_test.shape
```

Out[408]:

```
(228,)
```

Trying SVM model

Let us use the GridSearchCv for the purpose of SVM

Importing the required modules

In [409]:

```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_learning_lecture_2/Machine%20Learning%20Lecture%202.html

from sklearn.model_selection import learning_curve, GridSearchCV
from sklearn.linear_model import SGDClassifier
```

Giving a set of values of 'alpha' to get which works best.

In [414]:

```
tuned_parameters = {'alpha': [0.0075,0.015,0.03,0.06,0.15,0.3,0.75]}
```

Training our svm model using L2 Regularization as the penalty

In [415]:

```
# refer --- > https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html

# training the model

svm_ = SGDClassifier(loss = 'hinge' , penalty = 'l2',class_weight='balanced')

model_svm = GridSearchCV(svm_,tuned_parameters,cv=5,scoring = 'roc_auc')

model_svm.fit(X_train,y_train)
```

Out[415]:

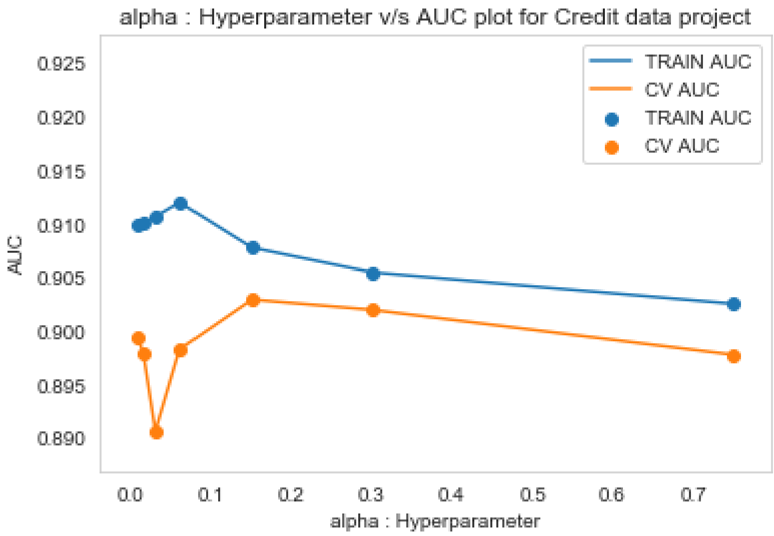
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight
='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=Non
e,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.0075, 0.015, 0.03, 0.06, 0.15, 0.3, 0.75]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [416]:

```
#https://stackoverflow.com/questions/44947574/what-is-the-meaning-of-mean-test-score-in
-cv-result

train_auc= model_svm.cv_results_['mean_train_score']
cv_auc = model_svm.cv_results_['mean_test_score']
```

In [417]:

```
plt.scatter(tuned_parameters['alpha'],train_auc,label = 'TRAIN AUC')
plt.scatter(tuned_parameters['alpha'],cv_auc,label = 'CV AUC')
plt.plot(tuned_parameters['alpha'],train_auc,label = 'TRAIN AUC')
plt.plot(tuned_parameters['alpha'],cv_auc,label = 'CV AUC')
plt.legend()
plt.xlabel("alpha : Hyperparameter")
plt.ylabel("AUC")
plt.title("alpha : Hyperparameter v/s AUC plot for Credit data project")
plt.grid()
plt.show()
```



From above we can see that we get closer cv_auc and train_auc at alpha = 0.3 and higher cv_auc value also hence we will consider alpha = 0.3 as the best alpha for our model

Training our svm model using L2 Regularization as the penalty

In [418]:

```
# refer --- > https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html

# training the model

svm_ = SGDClassifier(loss = 'hinge' , penalty = 'l1',class_weight='balanced')

model_svm = GridSearchCV(svm_,tuned_parameters,cv=5,scoring = 'roc_auc')

model_svm.fit(X_train,y_train)
```

Out[418]:

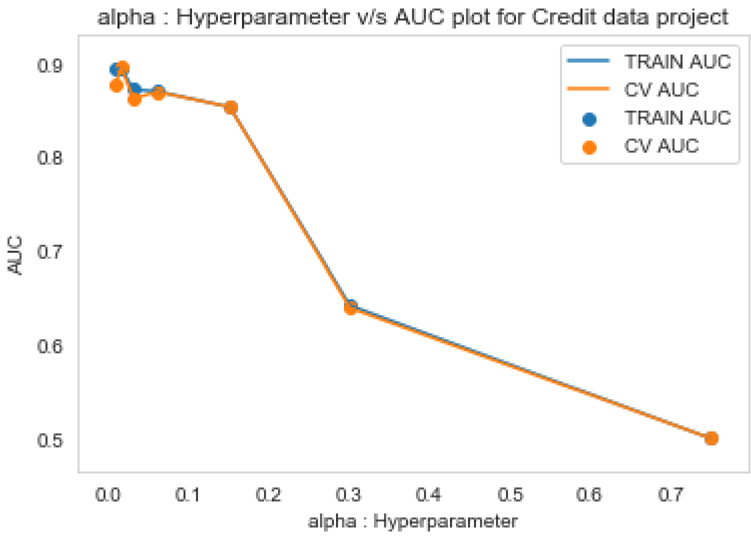
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight
='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=Non
e,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.0075, 0.015, 0.03, 0.06, 0.15, 0.3, 0.75]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [419]:

```
#https://stackoverflow.com/questions/44947574/what-is-the-meaning-of-mean-test-score-in
-cv-result

train_auc= model_svm.cv_results_['mean_train_score']
cv_auc = model_svm.cv_results_['mean_test_score']
```

In [420]:

```
plt.scatter(tuned_parameters['alpha'],train_auc,label = 'TRAIN AUC')
plt.scatter(tuned_parameters['alpha'],cv_auc,label = 'CV AUC')
plt.plot(tuned_parameters['alpha'],train_auc,label = 'TRAIN AUC')
plt.plot(tuned_parameters['alpha'],cv_auc,label = 'CV AUC')
plt.legend()
plt.xlabel("alpha : Hyperparameter")
plt.ylabel("AUC")
plt.title("alpha : Hyperparameter v/s AUC plot for Credit data project")
plt.grid()
plt.show()
```



we are getting lot of overlap by using L1 regularization as penalty hence we will go with L2 as penalty and alpha = 0.3 for our final model

Training our final model

In [422]:

```
svm_ = SGDClassifier(loss = 'hinge' , penalty = 'l2',alpha = 0.3,class_weight='balance
d')
svm_.fit(X_train,y_train)
```

Out[422]:

```
SGDClassifier(alpha=0.3, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=Non
e,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

Drawing the roc curve to see the performance

In [423]:

```
from sklearn.metrics import roc_auc_score
import math
```

In [424]:

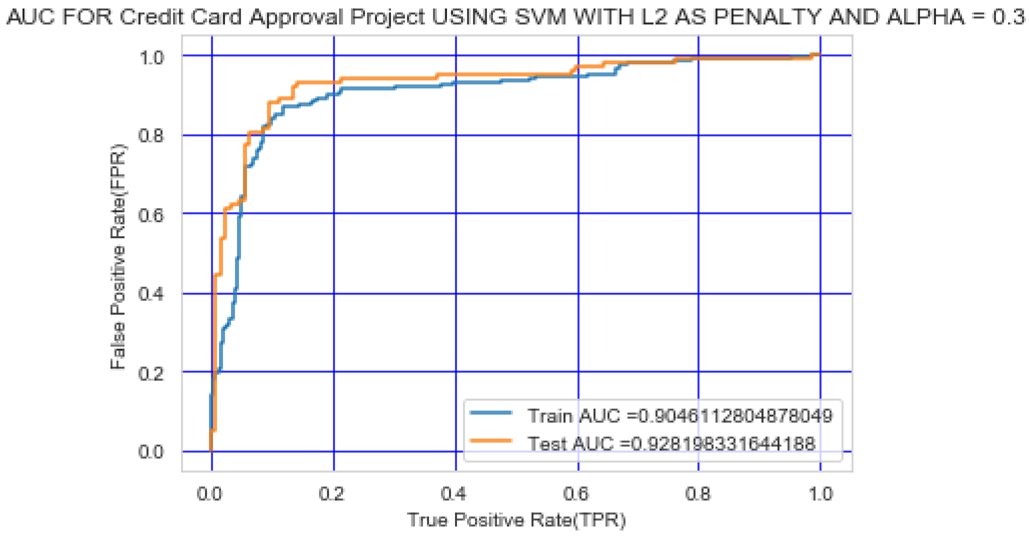
```
y_train_pred = svm_.decision_function(X_train)
y_test_pred = svm_.decision_function(X_test)
```

In [425]:

```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
```

In [426]:

```
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid(b=True, which='major', color='b', linestyle='-')
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC FOR Credit Card Approval Project USING SVM WITH L2 AS PENALTY AND ALPHA
 = 0.3")

plt.show()
```



We received the train accuracy of 0.90 and test accuracy of 0.93 which is very good actually.

Confusion matrix

In [427]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
nd(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Confusion matrix for test and train data.

In [428]:

```
print("////"*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_f
pr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr
)))
```

```
////////////////////////////////////////////////////////////////////////

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24945068359375 for threshold -0.624
[[134 122]
 [ 14 191]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2418004836009672 for threshold 0.417
[[120   7]
 [ 26  75]]
```

Visually seeing the confusion matrix for the training data

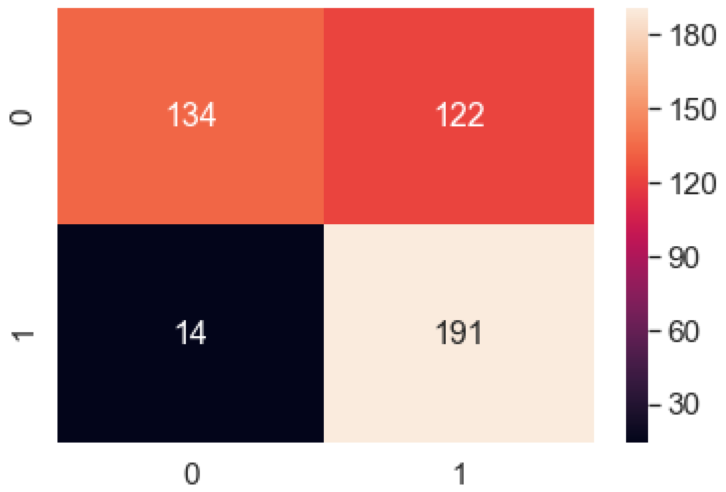
In [429]:

```
# Code for this segment from here -->> https://stackoverflow.com/questions/35572000/how
-can-i-plot-a-confusion-matrix

import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt


df_cm = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred, tr_thresholds, trai
n_fpr, train_fpr)), range(2),
                  range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')# font size
```

```
the maximum value of tpr*(1-fpr) 0.24945068359375 for threshold -0.624
```

Out[429]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1df0705b748>
```



Visually seeing the confusion matrix for the test data

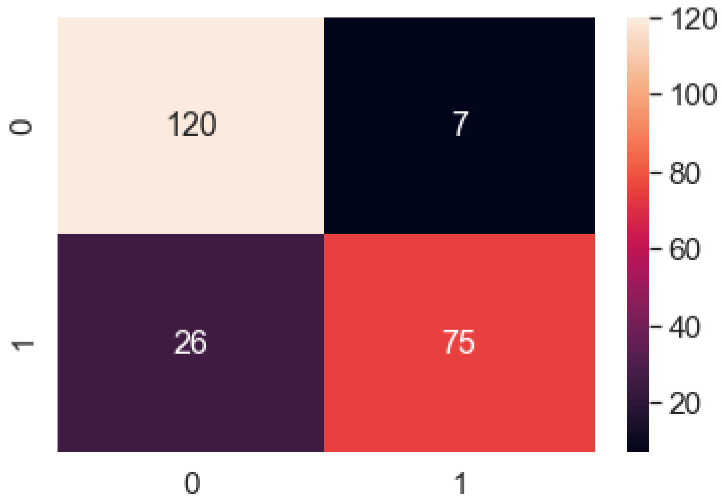
In [430]:

```
# Code for this segment from here -->> https://stackoverflow.com/questions/35572000/how
-can-i-plot-a-confusion-matrix

import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

df_cm_test = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred, tr_thresholds, t
est_fpr, test_fpr)), range(2),
                  range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm_test, annot=True,annot_kws={"size": 16},fmt='g')# font size
```

```
the maximum value of tpr*(1-fpr) 0.2418004836009672 for threshold 0.417
```

Out[430]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x1df06a21518&gt;
```



Summary

The project we have here was a supervised learning project where we had the data in labeled format but tha labels were given some random names due to confidentiality hence I renamed them based on the indices.

The best algorithm for the supervised learning whcih i know is support vector machines and hence to get he maximum accuracy I have used the support vector machines algorithm for the same

Real world application

Every bank receives 1000s of applications for credit card approval and that is not an easy task to go through every application by human beings hence using this model we can reject most of the applications which don't qualify for approval and hence decreasing the pressure on a human being and hence allowing the banks for faster process of the applications.

RESULT --- >>

TRAIN ACCURACY RECEIVED = 90%

TEST ACCURACY RECEIVED = 93%

THANKYOU

In [ ]:

```

```