Retention of Credit card customers

In [1]:
```
#importing libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline
import sys
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import f1_score,precision_score,roc_auc_score
```

In [2]:
```
import numpy as np
```

In [3]:
```
#reading data
df = pd.read_csv('customer retention.csv')
```

In [4]:
```
df.shape
```

Out[4]: (10127, 23)

In [5]:
```
df.head(10)
```

Out[5]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Ca
0	768805383	Existing Customer	45	M	3	High School	Married	60K−80K	
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	
2	713982108	Existing Customer	51	M	3	Graduate	Married	80K−120K	
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	
4	709106358	Existing Customer	40	M	3	Uneducated	Married	60K−80K	
5	713061558	Existing Customer	44	M	2	Graduate	Married	40K−60K	
6	810347208	Existing Customer	51	M	4	Unknown	Married	$120K +	
7	818906208	Existing Customer	32	M	0	High School	Unknown	60K−80K	
8	710930508	Existing Customer	37	M	3	Uneducated	Single	60K−80K	
9	719661558	Existing Customer	48	M	2	Graduate	Single	80K−120K	

10 rows × 23 columns

In [6]:
```
#dropping unecessary columns
df = df.drop('Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Educati
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['CLIENTNUM', 'Attrition_Flag', 'Customer_Age', 'Gender',
       'Dependent_count', 'Education_Level', 'Marital_Status',
       'Income_Category', 'Card_Category', 'Months_on_book',
       'Total_Relationship_Count', 'Months_Inactive_12_mon',
       'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal',
       'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt',
       'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio',
       'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Lev
el_Months_Inactive_12_mon_2'],
      dtype='object')
```

In [8]:
```
df = df.drop('Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Educati
```

In [9]:
```
df.columns
```

Out[9]:
```
Index(['CLIENTNUM', 'Attrition_Flag', 'Customer_Age', 'Gender',
       'Dependent_count', 'Education_Level', 'Marital_Status',
       'Income_Category', 'Card_Category', 'Months_on_book',
       'Total_Relationship_Count', 'Months_Inactive_12_mon',
       'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal',
       'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt',
       'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio'],
      dtype='object')
```

In [10]:
```
df.shape
```

Out[10]: (10127, 21)

In [11]:
```
for col in df.select_dtypes('category').columns.to_list():
    print(col + ': '+ str(X[col].cat.categories.to_list()))
```

In [12]:
```
#checking for missing values
df.isna().sum()
```

Out[12]:
```
CLIENTNUM                   0
Attrition_Flag              0
Customer_Age                0
Gender                      0
Dependent_count             0
Education_Level             0
Marital_Status              0
Income_Category             0
Card_Category               0
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
dtype: int64
```

In [13]:
```
df2_corr = df.corr()
df2_corr
```

Out[13]:

	CLIENTNUM	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_
CLIENTNUM	1.000000	0.007613	0.006772	0.134588	0.006907	0
Customer_Age	0.007613	1.000000	-0.122254	0.788912	-0.010931	0
Dependent_count	0.006772	-0.122254	1.000000	-0.103062	-0.039076	-0
Months_on_book	0.134588	0.788912	-0.103062	1.000000	-0.009203	0
Total_Relationship_Count	0.006907	-0.010931	-0.039076	-0.009203	1.000000	-0
Months_Inactive_12_mon	0.005729	0.054361	-0.010768	0.074164	-0.003675	1
Contacts_Count_12_mon	0.005694	-0.018452	-0.040505	-0.010774	0.055203	0
Credit_Limit	0.005708	0.002476	0.068065	0.007507	-0.071386	-0
Total_Revolving_Bal	0.000825	0.014780	-0.002688	0.008623	0.013726	-0

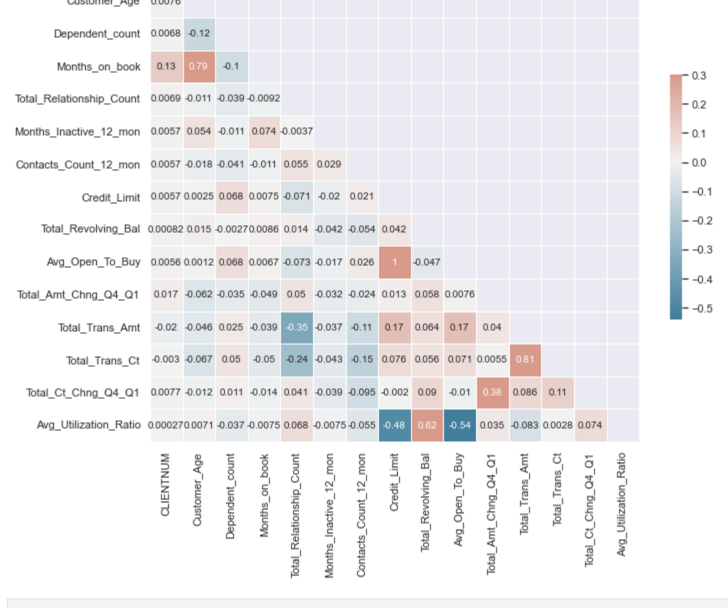
14	0.005708	0.002476	0.068065	0.007507	-0.071386	-0.020394	0.020817	1.000000	0.042493	0.995981	0.012813	0.1
15	0.000825	0.014780	-0.002688	0.008623	0.013726	-0.042210	-0.053913	0.042493	1.000000	-0.047167	0.058174	0.06
16	0.005633	0.001151	0.068291	0.006732	-0.072601	-0.016605	0.025646	0.995981	-0.047167	1.000000	0.007595	0.16
17	0.017369	-0.062042	-0.035439	-0.048959	0.050119	-0.032247	-0.024445	0.012813	0.058174	0.007595	1.000000	0.03
18	-0.019692	-0.046446	0.025046	-0.038591	-0.347229	-0.036982	-0.112774	0.171730	0.064370	0.165923	0.039678	1.00
19	-0.002961	-0.067097	0.049912	-0.049819	-0.241891	-0.042787	-0.152213	0.075927	0.056060	0.070885	0.005469	0.8
20	0.007696	-0.012143	0.011087	-0.014072	0.040831	-0.038989	-0.094997	-0.002020	0.089861	-0.010076	0.384189	0.08
21	0.000266	0.007114	-0.037135	-0.007541	0.067663	-0.007503	-0.055471	-0.482965	0.624022	-0.538808	0.035235	-0.08

In [22]:
```
df.columns
```

Out[22]:
```
Index(['CLIENTNUM', 'Attrition_Flag', 'Customer_Age', 'Gender',
       'Dependent_count', 'Education_Level', 'Marital_Status',
       'Income_Category', 'Card_Category', 'Months_on_book',
       'Total_Relationship_Count', 'Months_Inactive_12_mon',
       'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal',
       'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt',
       'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio'],
      dtype='object')
```

In [23]:
```
#plotting heatmap for correlation matrix
df_corr = df.corr()
mask = np.triu(np.ones_like(df_corr, dtype=bool))
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(df_corr,annot=True, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Out[23]: <AxesSubplot:>



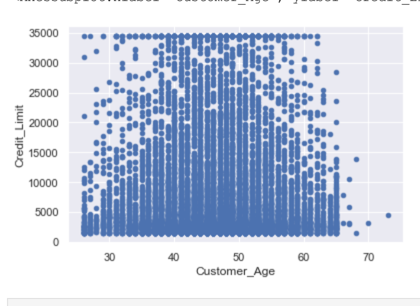
In [24]:
```
df['Education_Level'].value_counts()
```

Out[24]:
```
Graduate         3128
High School      2013
Unknown          1519
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: Education_Level, dtype: int64
```

In [25]:
```
df.plot.scatter('Customer_Age', 'Credit_Limit')
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```

Out[25]: <AxesSubplot:xlabel='Customer_Age', ylabel='Credit_Limit'>



In [26]:
```
df1 = pd.DataFrame(columns=['Income_Category','Attrition_Flag'], data = df)
```

In [27]:
```
df1.head()
```

Out[27]:

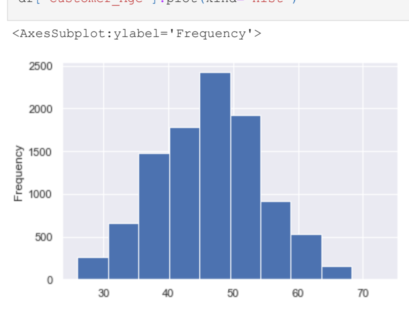
	Income_Category	Attrition_Flag
0	60K−80K	Existing Customer
1	Less than $40K	Existing Customer
2	80K−120K	Existing Customer
3	Less than $40K	Existing Customer
4	60K−80K	Existing Customer

In [28]:
```
df['Income_Category'].value_counts()
```

Out[28]:
```
Less than $40K    3561
$40K - $60K       1790
$80K - $120K      1535
$60K - $80K       1402
Unknown           1112
$120K +            727
Name: Income_Category, dtype: int64
```

In [29]:
```
df['Customer_Age'].plot(kind='hist')
```

Out[29]: <AxesSubplot:ylabel='Frequency'>

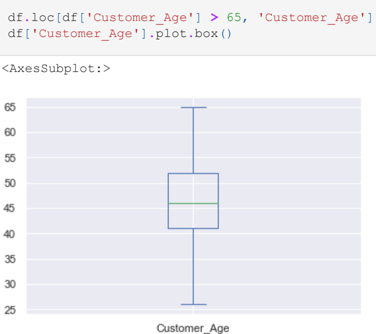


In [30]:
```
pd.cut(df['Customer_Age'],bins = 4).value_counts()
```

Out[30]:
```
(37.75, 49.5]      5197
(49.5, 61.25]      3218
(25.953, 37.75]    1400
(61.25, 73.0]       312
Name: Customer_Age, dtype: int64
```

In [31]:
```
df.loc[df['Customer_Age'] > 65, 'Customer_Age'] = np.mean(df['Customer_Age'])
df['Customer_Age'].plot.box()
```

Out[31]: <AxesSubplot:>



In [32]:
```
bins = [0,35,45,55,70]
cut_labels = ['0_35', '35_45', '45_55', '55_70']
df['Customer_Age'] = pd.cut(df['Customer_Age'], bins=bins, labels=cut_labels)
```

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

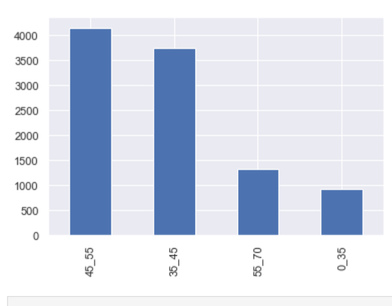
Out[33]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Ca
0	768805383	Existing Customer	35_45	M	3	High School	Married	60K−80K	
1	818770008	Existing Customer	45_55	F	5	Graduate	Single	Less than $40K	
2	713982108	Existing Customer	45_55	M	3	Graduate	Married	80K−120K	
3	769911858	Existing Customer	35_45	F	4	High School	Unknown	Less than $40K	
4	709106358	Existing Customer	35_45	M	3	Uneducated	Married	60K−80K	

5 rows × 21 columns

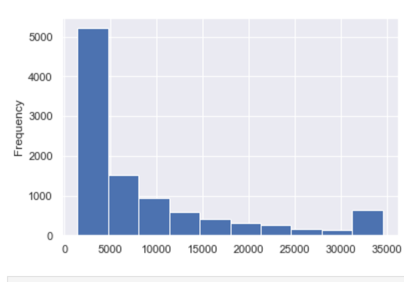
In [34]:
```
df['Customer_Age'].value_counts().plot(kind = 'bar')
```

Out[34]: <AxesSubplot:>

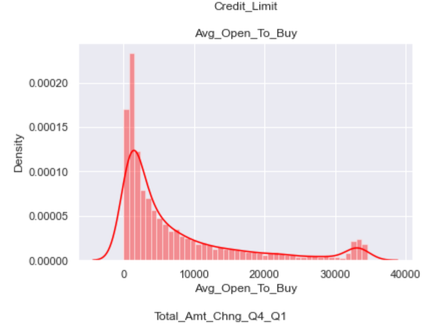


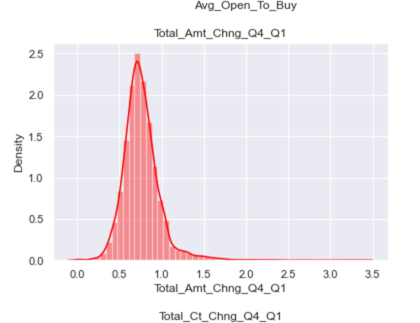
In [35]:
```
df['Credit_Limit'].plot(kind = 'hist')
```

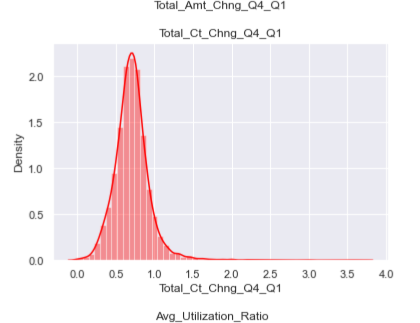
Out[35]: <AxesSubplot:ylabel='Frequency'>

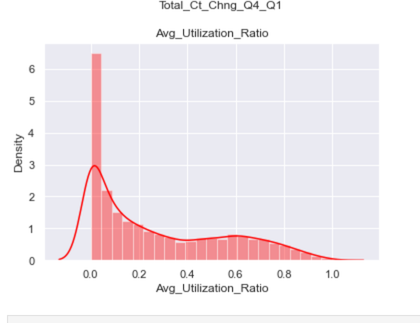


In [36]:
```
cont_attributes = df.select_dtypes(['float64']).columns.to_list()

for cont_attribute in cont_attributes:
    plt.figure()
    plt.title(cont_attribute)
    ax = sns.distplot(df[cont_attribute], color = 'red')
```

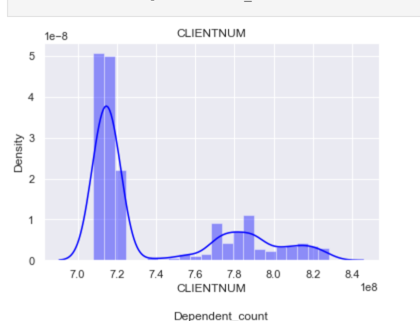


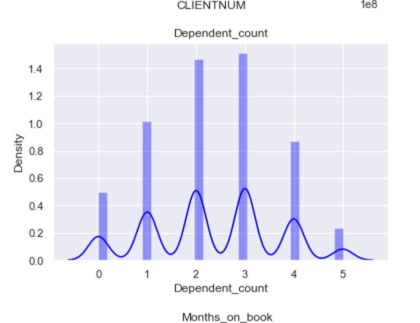


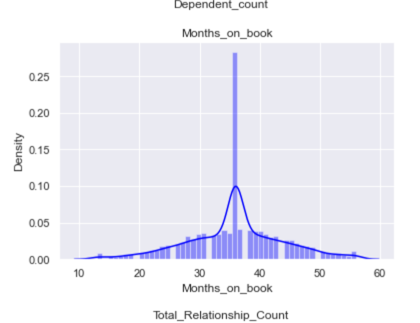


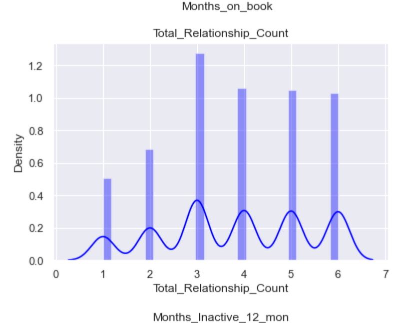


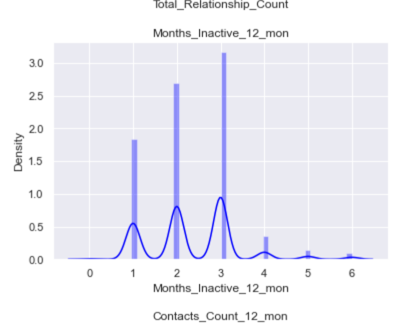
In [37]:
```
cont_attributes = df.select_dtypes(['int64']).columns.to_list()

for cont_attribute in cont_attributes:
    plt.figure()
    plt.title(cont_attribute)
    ax = sns.distplot(df[cont_attribute], color = 'blue')
```

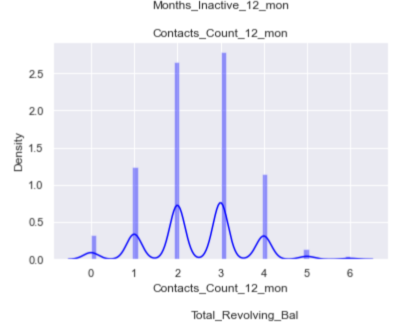


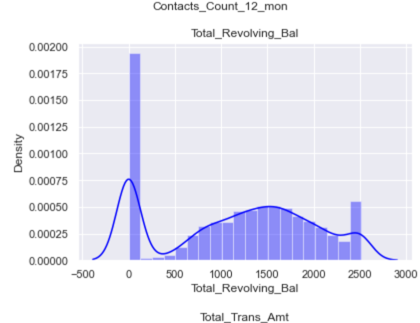


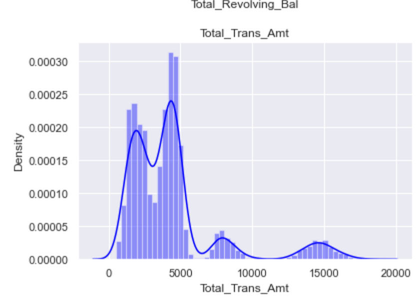


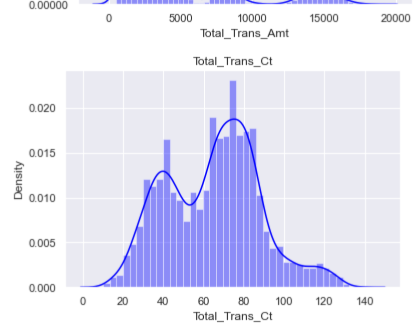




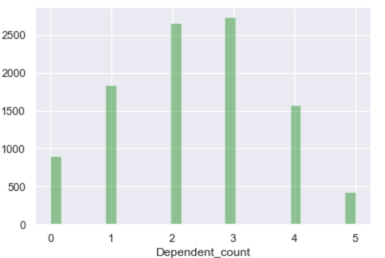


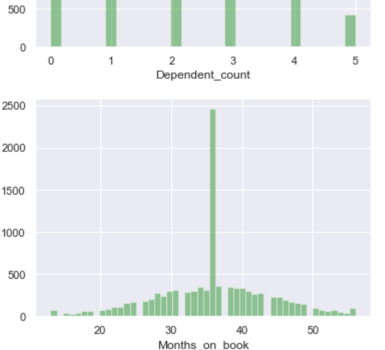


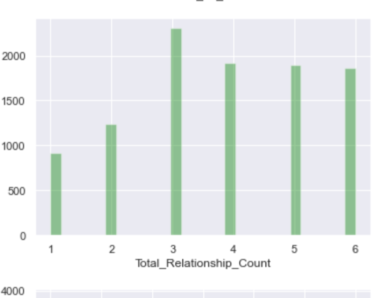


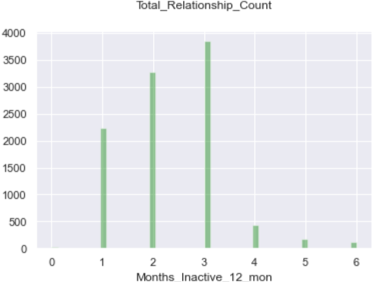


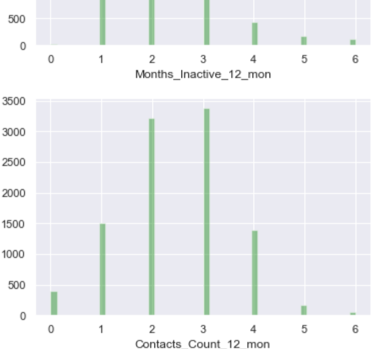
In [38]:
```
discrete_attributes = df.select_dtypes(['int64']).columns.to_list()

for disc_attribute in discrete_attributes:
    if not(disc_attribute == 'CLIENTNUM'):
        plt.figure()
        ax = sns.distplot(df[disc_attribute],kde = False, color = 'green')
```

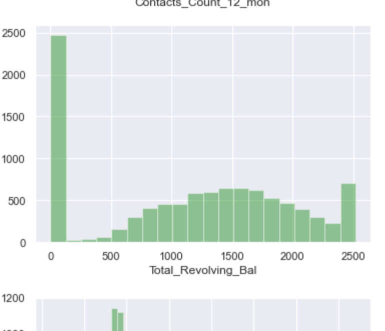












In [39]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   CLIENTNUM                 10127 non-null  int64
 1   Attrition_Flag            10127 non-null  object
 2   Customer_Age              10127 non-null  category
 3   Gender                    10127 non-null  object
 4   Dependent_count           10127 non-null  int64
 5   Education_Level           10127 non-null  object
 6   Marital_Status            10127 non-null  object
 7   Income_Category           10127 non-null  object
 8   Card_Category             10127 non-null  object
 9   Months_on_book            10127 non-null  int64
 10  Total_Relationship_Count  10127 non-null  int64
 11  Months_Inactive_12_mon    10127 non-null  int64
 12  Contacts_Count_12_mon     10127 non-null  int64
 13  Credit_Limit              10127 non-null  float64
 14  Total_Revolving_Bal       10127 non-null  int64
 15  Avg_Open_To_Buy           10127 non-null  float64
 16  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 17  Total_Trans_Amt           10127 non-null  int64
 18  Total_Trans_Ct            10127 non-null  int64
 19  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 20  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: category(1), float64(5), int64(9), object(6)
memory usage: 1.6+ MB
```

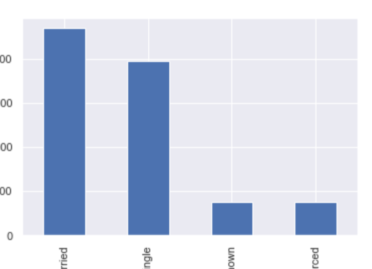
In [40]:
```
df['Attrition_Flag'].value_counts()
```

Out[40]:
```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
```

In [41]:
```
#labels = {'Existing Customer':'0','Attrited Customer':'1'}
#df['Attrition_Flag'] = df['Attrition_Flag'].map(labels)
#df.head()
```

In [42]:
```
df['Marital_Status'].value_counts().plot(kind = 'bar')
```

Out[42]:
```
<AxesSubplot:>
```



In [43]:
```
#dropping duplicate records
df.drop_duplicates().head()
```

Out[43]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Ca
0	768805383	Existing Customer	35_45	M	3	High School	Married	60K−80K	
1	818770008	Existing Customer	45_55	F	5	Graduate	Single	Less than $40K	
2	713982108	Existing Customer	45_55	M	3	Graduate	Married	80K−120K	
3	769911858	Existing Customer	35_45	F	4	High School	Unknown	Less than $40K	
4	709106358	Existing Customer	35_45	M	3	Uneducated	Married	60K−80K	

5 rows × 21 columns

In [44]:
```
df['Marital_Status'].replace(to_replace = 'Unknown', value = 'Single', inplace = True)
```

In [45]:
```
df['Marital_Status'].value_counts()
```

Out[45]:
```
Single      4692
Married     4687
Divorced     748
Name: Marital_Status, dtype: int64
```

In [46]:
```
cat_types = ['bool','object','category']
data = df.copy()
data[data.select_dtypes(cat_types).columns] = data.select_dtypes(cat_types).apply(lambda x: x.astype('category'
```

In [47]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   CLIENTNUM                 10127 non-null  int64
 1   Attrition_Flag            10127 non-null  category
 2   Customer_Age              10127 non-null  category
 3   Gender                    10127 non-null  category
 4   Dependent_count           10127 non-null  int64
 5   Education_Level           10127 non-null  category
 6   Marital_Status            10127 non-null  category
 7   Income_Category           10127 non-null  category
 8   Card_Category             10127 non-null  category
 9   Months_on_book            10127 non-null  int64
 10  Total_Relationship_Count  10127 non-null  int64
 11  Months_Inactive_12_mon    10127 non-null  int64
 12  Contacts_Count_12_mon     10127 non-null  int64
 13  Credit_Limit              10127 non-null  float64
 14  Total_Revolving_Bal       10127 non-null  int64
 15  Avg_Open_To_Buy           10127 non-null  float64
 16  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 17  Total_Trans_Amt           10127 non-null  int64
 18  Total_Trans_Ct            10127 non-null  int64
 19  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 20  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: category(7), float64(5), int64(9)
memory usage: 1.2 MB
```

In [48]:
```
labels = {'Existing Customer':'0','Attrited Customer':'1'}
data['Attrition_Flag'] = data['Attrition_Flag'].map(labels)
data.head()
```

Out[48]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Ca
0	768805383	0	35_45	M	3	High School	Married	60K−80K	
1	818770008	0	45_55	F	5	Graduate	Single	Less than $40K	
2	713982108	0	45_55	M	3	Graduate	Married	80K−120K	
3	769911858	0	35_45	F	4	High School	Single	Less than $40K	
4	709106358	0	35_45	M	3	Uneducated	Married	60K−80K	

5 rows × 21 columns

In [49]:
```
data['Attrition_Flag'].value_counts()
```

Out[49]:
```
0    8500
1    1627
Name: Attrition_Flag, dtype: int64
```

In [50]:
```
#Output and input variables
Y = data['Attrition_Flag']
X = data.drop('Attrition_Flag',axis = 1)
```

In [51]:
```
X.shape,Y.shape
```

Out[51]:
```
((10127, 20), (10127,))
```

In [52]:
```
for col in X.select_dtypes('category').columns.to_list():
    print(col + ': '+ str(X[col].cat.categories.to_list()))
```

```
Customer_Age: ['0_35', '35_45', '45_55', '55_70']
Gender: ['F', 'M']
Education_Level: ['College', 'Doctorate', 'Graduate', 'High School', 'Post-Graduate', 'Uneducated', 'Unknown']
Marital_Status: ['Divorced', 'Married', 'Single']
Income_Category: ['$120K +', '$40K - $60K', '$60K - $80K', '$80K - $120K', 'Less than $40K', 'Unknown']
Card_Category: ['Blue', 'Gold', 'Platinum', 'Silver']
```

In [53]:
```
col_list = X.select_dtypes('category').columns.to_list()
col_list
```

Out[53]:
```
['Customer_Age',
 'Gender',
 'Education_Level',
 'Marital_Status',
 'Income_Category',
 'Card_Category']
```

Preparing Data

One-hot encoding

In [54]:
```
def encode_fun(dataframe,attributes):
    dummies = pd.get_dummies(dataframe[attributes])
    result = pd.concat([dataframe,dummies],axis =1)
    result = result.drop([attributes], axis=1)
    return(result)
```

In [55]:
```
col_list = X.select_dtypes('category').columns.to_list()
for col in col_list:
    X = encode_fun(X,col)
```

In [56]:
```
X.head()
```

Out[56]:

	CLIENTNUM	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon	Cre
0	768805383	3	39	5	1	3	
1	818770008	5	44	6	1	2	
2	713982108	3	36	4	1	0	
3	769911858	4	34	3	4	1	
4	709106358	3	21	5	1	0	

5 rows × 40 columns

In [57]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 40 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   CLIENTNUM                 10127 non-null  int64
 1   Dependent_count           10127 non-null  int64
 2   Months_on_book            10127 non-null  int64
 3   Total_Relationship_Count  10127 non-null  int64
 4   Months_Inactive_12_mon    10127 non-null  int64
 5   Contacts_Count_12_mon     10127 non-null  int64
 6   Credit_Limit              10127 non-null  float64
 7   Total_Revolving_Bal       10127 non-null  int64
 8   Avg_Open_To_Buy           10127 non-null  float64
 9   Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 10  Total_Trans_Amt           10127 non-null  int64
 11  Total_Trans_Ct            10127 non-null  int64
 12  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 13  Avg_Utilization_Ratio     10127 non-null  float64
 14  0_35                      10127 non-null  uint8
 15  35_45                     10127 non-null  uint8
 16  45_55                     10127 non-null  uint8
 17  55_70                     10127 non-null  uint8
 18  F                         10127 non-null  uint8
 19  M                         10127 non-null  uint8
 20  College                   10127 non-null  uint8
 21  Doctorate                 10127 non-null  uint8
 22  Graduate                  10127 non-null  uint8
 23  High School               10127 non-null  uint8
 24  Post-Graduate             10127 non-null  uint8
 25  Uneducated                10127 non-null  uint8
 26  Unknown                   10127 non-null  uint8
 27  Divorced                  10127 non-null  uint8
 28  Married                   10127 non-null  uint8
 29  Single                    10127 non-null  uint8
 30  $120K +                   10127 non-null  uint8
 31  $40K - $60K               10127 non-null  uint8
 32  $60K - $80K               10127 non-null  uint8
 33  $80K - $120K              10127 non-null  uint8
 34  Less than $40K            10127 non-null  uint8
 35  Unknown                   10127 non-null  uint8
 36  Blue                      10127 non-null  uint8
 37  Gold                      10127 non-null  uint8
 38  Platinum                  10127 non-null  uint8
 39  Silver                    10127 non-null  uint8
dtypes: float64(5), int64(9), uint8(26)
memory usage: 1.3 MB
```

Performing Stratified K-Fold split

In [58]:
```
def kfold_split(X,Y):
    skfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=0)
    for train_index, test_index in skfold.split(X,Y):
        skf_X_train, skf_X_test = X.iloc[train_index], X.iloc[test_index]
        skf_Y_train, skf_Y_test = Y.iloc[train_index], Y.iloc[test_index]
        skf_X_train, skf_X_test, skf_Y_train, skf_Y_test = skf_X_train.values, skf_X_test.values, skf_Y_train.v
        return skf_X_train, skf_X_test, skf_Y_train, skf_Y_test
```

In [59]:
```
X_train, X_test, Y_train, Y_test = kfold_split(X,Y)
```

Cross-validation function

In [60]:
```
from sklearn.model_selection import KFold
def cross_val_score_func(model,X,y):
    cv = KFold(n_splits=10, random_state=1, shuffle=True)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
    return('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Implementing models

Decision tree

In [61]:
```
model1 = DecisionTreeClassifier(random_state=0)
```

In [62]:
```
model1.fit(X_train,Y_train)
model1.score(X_test,Y_test)
```

Out[62]:
```
0.9259624876604146
```

In [63]:
```
model1.predict(X_test)
```

Out[63]:
```
array(['0', '0', '1', ..., '0', '0', '0'], dtype=object)
```

In [64]:
```
pred1=model1.predict(X_test)
pred1[:10], model1.score(X_test, Y_test)
```

Out[64]:
```
(array(['0', '0', '1', '1', '0', '0', '1', '0', '0', '1'], dtype=object),
 0.9259624876604146)
```

In [65]:
```
train_accuracy = []
test_accuracy = []
for d in range(1,10):
    model_test = DecisionTreeClassifier(max_depth=d, random_state=0)
    model_test.fit(X_train,Y_train)
    train_accuracy.append(model_test.score(X_train,Y_train))
    test_accuracy.append(model_test.score(X_test,Y_test))
```

In [66]:
```
test_res = pd.DataFrame({'max_depth':range(1,10),'train_accuracy':train_accuracy,'test_accuracy':test_accuracy
test_res
```

Out[66]:

	max_depth	train_accuracy	test_accuracy
0	1	0.839368	0.839092
1	2	0.891705	0.900296
2	3	0.918697	0.923001
3	4	0.927035	0.929911
4	5	0.944152	0.930898
5	6	0.955234	0.934847
6	7	0.965109	0.942744
7	8	0.974984	0.941757
8	9	0.980689	0.940770

In [67]:
```
plt.figure(figsize=(15,10))
plt.plot(test_res['max_depth'],test_res['train_accuracy'],marker = 'o')
plt.plot(test_res['max_depth'],test_res['test_accuracy'],marker = 'o',color = 'red')
plt.xlabel('max_depth')
plt.ylabel('score/accuracy')
plt.legend(['Train_accuracy','Test_accuracy'])
```

Out[67]:
```
<matplotlib.legend.Legend at 0x7f7fcfb27880>
```

Hyper-parameter tuning

In [68]:
```
train_accuracy = []
test_accuracy = []
for rs in range(0,100):
    model_test = DecisionTreeClassifier(max_depth=5, random_state=rs)
    model_test.fit(X_train,Y_train)
    train_accuracy.append(model_test.score(X_train,Y_train))
    test_accuracy.append(model_test.score(X_test,Y_test))
```

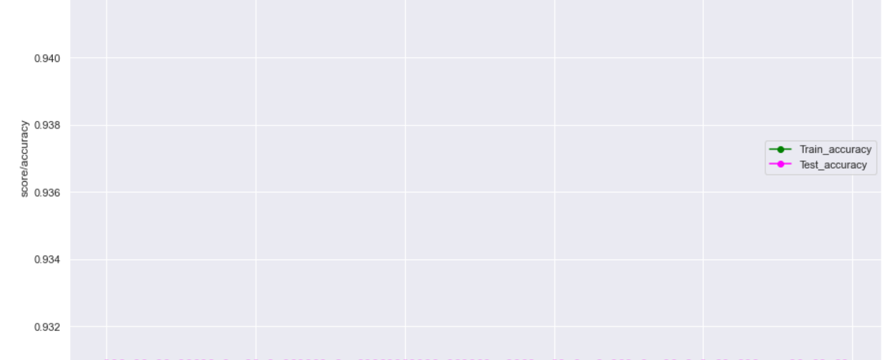
In [69]:
```
test_res2 = pd.DataFrame({'random_state':range(0,100),'train_accuracy':train_accuracy,'test_accuracy':test_accu
test_res2.head()
```

Out[69]:

	random_state	train_accuracy	test_accuracy
0	0	0.944152	0.930898
1	1	0.944152	0.930898
2	2	0.944152	0.930898
3	3	0.944152	0.929911
4	4	0.944152	0.930898

In [70]:
```
plt.figure(figsize=(15,10))
plt.plot(test_res2['random_state'],test_res2['train_accuracy'],marker = 'o',color = 'green')
plt.plot(test_res2['random_state'],test_res2['test_accuracy'],marker = 'o',color = 'magenta')
plt.xlabel('random_state')
plt.ylabel('score/accuracy')
plt.legend(['Train_accuracy','Test_accuracy'])
```

Out[70]:
```
<matplotlib.legend.Legend at 0x7f7fd12cea90>
```



In [71]:
```
train_accuracy = []
test_accuracy = []
for leaf_nodes in range(2,30):
    model_test = DecisionTreeClassifier(max_depth=5, random_state=0, max_leaf_nodes=leaf_nodes)
    model_test.fit(X_train,Y_train)
    train_accuracy.append(model_test.score(X_train,Y_train))
    test_accuracy.append(model_test.score(X_test,Y_test))
```

In [72]:
```
test_res3 = pd.DataFrame({'max_leaf_nodes':range(2,30),'train_accuracy':train_accuracy,'test_accuracy':test_acc
test_res3.head()
```

Out[72]:

	max_leaf_nodes	train_accuracy	test_accuracy
0	2	0.839368	0.839092
1	3	0.891705	0.900296
2	4	0.903665	0.909181
3	5	0.903665	0.909181
4	6	0.905859	0.910168

In [73]:
```
plt.figure(figsize=(15,10))
plt.plot(test_res3['max_leaf_nodes'],test_res3['train_accuracy'],marker = 'o',color = 'brown')
plt.plot(test_res3['max_leaf_nodes'],test_res3['test_accuracy'],marker = 'o',color = 'orange')
plt.xlabel('max_leaf_nodes')
plt.ylabel('score/accuracy')
plt.legend(['Train_accuracy','Test_accuracy'])
```

Out[73]:
```
<matplotlib.legend.Legend at 0x7f7fd0f44e20>
```

In [74]:
```
DT = DecisionTreeClassifier(max_depth=5,random_state=0,max_leaf_nodes=25)
DT.fit(X_train,Y_train)
DT_score = DT.score(X_test,Y_test)
```

In [75]:
```
DT_score
```

Out[75]:
```
0.930898321816387
```

In [76]:
```
def cross_val_func(X, Y, tree_depths, cv=5, scoring='accuracy'):
    cv_scores_list = []
    cv_scores_std = []
    cv_scores_mean = []
    accuracy_scores = []
    for depth in tree_depths:
        tree_model = DecisionTreeClassifier(max_depth=depth)
        cv_scores = cross_val_score(tree_model, X, Y, cv=cv, scoring=scoring)
        cv_scores_list.append(cv_scores)
        cv_scores_mean.append(cv_scores.mean())
        cv_scores_std.append(cv_scores.std())
        accuracy_scores.append(tree_model.fit(X, Y).score(X, Y))
    cv_scores_mean = np.array(cv_scores_mean)
    cv_scores_std = np.array(cv_scores_std)
    accuracy_scores = np.array(accuracy_scores)
    return cv_scores_mean, cv_scores_std, accuracy_scores
```

In [77]:
```
def plot_cross_val_tree(depths, cv_scores_mean, cv_scores_std, accuracy_scores, title):
    fig, ax = plt.subplots(1,1, figsize=(15,10))
    ax.plot(depths, cv_scores_mean, '-o', label='mean cross-validation accuracy', alpha=0.9)
    ax.fill_between(depths, cv_scores_mean-2*cv_scores_std, cv_scores_mean+2*cv_scores_std, alpha=0.2)
    ax.plot(depths, accuracy_scores, '-*', label='train accuracy', alpha=0.9)
    ax.set_title(title, fontsize=16)
    ax.set_xlabel('Tree depth', fontsize=14)
    ax.set_ylabel('Accuracy', fontsize=14)
    ax.set_ylim()
    ax.set_xticks(depths)
    ax.legend()
```

In [78]:
```
td = range(1,25)
sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores = cross_val_func(X_train, Y_train, td)
```

In [79]:
```
plot_cross_val_tree(td, sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores,
                    'Accuracy per decision tree depth on training data')
```

In [80]:
```
dtree = DecisionTreeClassifier(max_depth=5, random_state=0, max_leaf_nodes=25)
```

In [81]:
```
dtree.fit(X_train,Y_train)
```

Out[81]:
```
DecisionTreeClassifier(max_depth=5, max_leaf_nodes=25, random_state=0)
```

In [82]:
```
y_test_pred = dtree.predict(X_test)
```

In [83]:
```
y_train_pred = dtree.predict(X_train)
```

In [84]:
```
from sklearn.metrics import confusion_matrix, classification_report
cm_train = confusion_matrix(Y_train, y_train_pred,labels=dtree.classes_)
cm_test = confusion_matrix(Y_test, y_test_pred,labels=dtree.classes_)
```

In [85]:
```
cm_train
```

Out[85]:
```
array([[7395,  255],
       [ 258, 1206]])
```

In [86]:
```
cm_test
```

Out[86]:
```
array([[817,  33],
       [ 37, 126]])
```

In [87]:
```
dtree_f1score ={'0': 0.96,'1':0.77}
dtree_prec = {'0':0.96,'1':0.79}
```

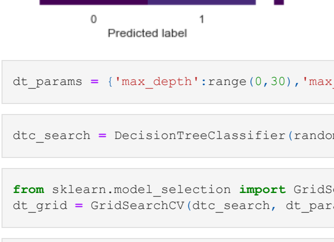
In [88]:
```
dtree_cval = cross_val_score(dtree,X_test,Y_test,scoring='accuracy')
dtree_cross_val = dtree_cval.mean()
dtree_cross_val
```

Out[88]:
```
0.8845095839633224
```

In [89]:
```
from sklearn.metrics import plot_confusion_matrix, ConfusionMatrixDisplay
```

In [90]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_test,display_labels=dtree.classes_)
```

In [91]:
```
disp.plot()
plt.grid(False)
```



In [92]:
```
dt_params = {'max_depth':range(0,30),'max_leaf_nodes':range(0,30),'min_samples_split': [2, 3, 4]}
```

In [93]:
```
dtc_search = DecisionTreeClassifier(random_state=0)
```

In [94]:
```
from sklearn.model_selection import GridSearchCV
dt_grid = GridSearchCV(dtc_search, dt_params, verbose=1, cv=3)
```

In [95]:
```
dt_grid.fit(X_train,Y_train)
```

Out[95]:
```
Fitting 3 folds for each of 2700 candidates, totalling 8100 fits
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': range(0, 30),
                         'max_leaf_nodes': range(0, 30),
                         'min_samples_split': [2, 3, 4]},
             verbose=1)
```

In [96]:
```
dt_grid.best_params_
```

Out[96]:
```
{'max_depth': 5, 'max_leaf_nodes': 16, 'min_samples_split': 2}
```

In [97]:
```
dt_grid.best_estimator_
```

Out[97]:
```
DecisionTreeClassifier(max_depth=5, max_leaf_nodes=16, random_state=0)
```

In [98]:
```
dt_grid.best_score_
```

Out[98]:
```
0.7026552556506473
```

```
In [99]:
print("Classification report: ")
rep = classification_report(Y_test,y_test_pred)
print(rep)
print('Accuracy: ')
print(dtree.score(X_test,Y_test))
```

```
Classification report: 
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       850
           1       0.79      0.77      0.78       163

    accuracy                           0.93      1013
   macro avg       0.87      0.87      0.87      1013
weighted avg       0.93      0.93      0.93      1013

Accuracy: 
0.930898321816387
```

Final score of Decision Tree is 93.83% on the test data

kNN

```
In [100]:
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
```

```
In [101]:
model2 = KNeighborsClassifier()
```

```
In [102]:
model2.fit(X_train,Y_train)
x_pred = model2.predict(X_test)
x_pred
```

```
Out[102]: array(['0', '0', '0', ..., '0', '0', '0'], dtype=object)
```

```
In [103]:
kNN_score = model2.score(X_test,Y_test)
```

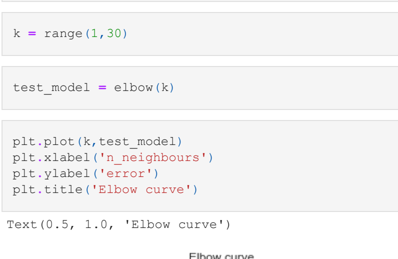
```
In [104]:
def elbow(k):
    test_error = []
    for i in k:
        test_model = KNeighborsClassifier(n_neighbors=i)
        test_model.fit(X_train,Y_train)
        pred_i = test_model.predict(X_test)
        test_error.append(np.mean(pred_i != Y_test))
    return test_error
```

```
In [105]:
k = range(1,30)
```

```
In [106]:
test_model = elbow(k)
```

```
In [107]:
plt.plot(k,test_model)
plt.xlabel('n_neighbours')
plt.ylabel('error')
plt.title('Elbow curve')
```

```
Out[107]: Text(0.5, 1.0, 'Elbow curve')
```




```
In [108]:
kNN_score
```

```
Out[108]: 0.8163869693978283
```

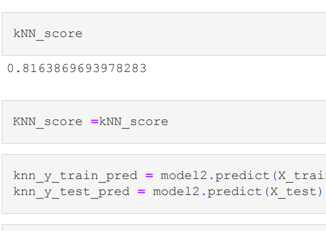
```
In [109]:
KNN_score =kNN_score
```

```
In [110]:
knn_y_train_pred = model2.predict(X_train)
knn_y_test_pred = model2.predict(X_test)
```

```
In [111]:
knn_cm_train = confusion_matrix(Y_train, knn_y_train_pred,labels=model2.classes_)
knn_cm_test = confusion_matrix(Y_test, knn_y_test_pred,labels=model2.classes_)
```

```
In [112]:
disp = ConfusionMatrixDisplay(confusion_matrix=knn_cm_test,display_labels=model2.classes_)
```

```
In [113]:
disp.plot()
plt.grid(False)
```




```
In [114]:
print("Classification report: ")
rep = classification_report(Y_test,knn_y_test_pred)
print(rep)
```

```
Classification report: 
              precision    recall  f1-score   support

           0       0.84      0.96      0.90       850
           1       0.23      0.06      0.10       163

    accuracy                           0.82      1013
   macro avg       0.54      0.51      0.50      1013
weighted avg       0.74      0.82      0.77      1013
```

```
In [115]:
knn_cval = cross_val_score(model2,X_test,Y_test,scoring='accuracy')
knn_cross_val = knn_cval.mean()
knn_cross_val
```

```
Out[115]: 0.8005950348729455
```

```
In [116]:
knn_f1score = {'0':0.90,'1':0.10}
knn_prec = {'0':0.84,'1':0.23}
```

```
In [117]:
knn_estimate = KNeighborsClassifier()
```

```
In [118]:
parameters_KNN = {'n_neighbors': [1,100, 1],'p': [1,2],'weights': ['uniform', 'distance'],'metric': ['minkowski
```

```
In [119]:
grid_search_KNN = GridSearchCV(
    estimator=knn_estimate,
    param_grid=parameters_KNN,
    scoring = 'accuracy',
    n_jobs = -1,
    cv = 5
)
```

```
In [120]:
grid_search_KNN.fit(X_train,Y_train)
```

```
Out[120]: GridSearchCV(cv=5, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'metric': ['minkowski', 'chebyshev'],
                         'n_neighbors': [1, 100, 1], 'p': [1, 2],
                         'weights': ['uniform', 'distance']},
             scoring='accuracy')
```

```
In [121]:
grid_search_KNN.best_params_
```

```
Out[121]: {'metric': 'minkowski', 'n_neighbors': 100, 'p': 1, 'weights': 'uniform'}
```

```
In [122]:
grid_search_KNN.best_estimator_
```

```
Out[122]: KNeighborsClassifier(n_neighbors=100, p=1)
```

```
In [123]:
grid_search_KNN.best_score_
```

```
Out[123]: 0.8393680456997519
```

FInal score of kNN is 81.49%

Random-Forest classifier

```
In [124]:
from sklearn.ensemble import RandomForestClassifier
```

```
In [125]:
RFC = RandomForestClassifier()
```

```
In [126]:
RFC.fit(X_train,Y_train)
```

```
Out[126]: RandomForestClassifier()
```

```
In [127]:
RFC.score(X_test,Y_test)
```

```
Out[127]: 0.9585389930898321
```

```
In [128]:
RFC_score = RFC.score(X_test,Y_test)
```

```
In [129]:
RFC.score(X_train,Y_train)
```

```
Out[129]: 1.0
```

```
In [130]:
RFC.feature_importances_
```

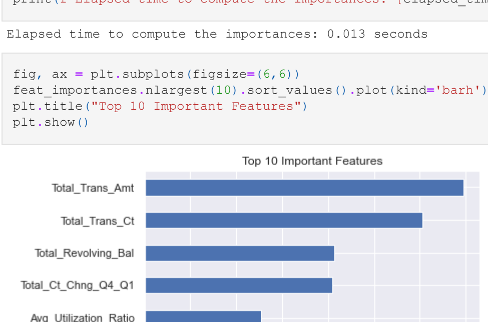
```
Out[130]: array([0.02798511, 0.01468376, 0.02600532, 0.06236768, 0.02679781,
       0.02683043, 0.03668295, 0.10325041, 0.0348332 , 0.06135704,
       0.17403662, 0.15169084, 0.10251076, 0.06366331, 0.00654652,
       0.00407256, 0.00513198, 0.00308278, 0.00792597, 0.00540371,
       0.00231961, 0.00224902, 0.0038614 , 0.0035403 , 0.00229217,
       0.00325971, 0.00319967, 0.00217038, 0.00587901, 0.00508476,
       0.0017417 , 0.00309023, 0.0026431 , 0.00294235, 0.00378429,
       0.00246572, 0.00168663, 0.00100223, 0.00032083, 0.00160809])
```

```
In [131]:
feat_importances = pd.Series(RFC.feature_importances_, index=X.columns)
```

```
In [132]:
import time
import numpy as np

start_time = time.time()
importances = RFC.feature_importances_
std = np.std([tree.feature_importances_ for tree in RFC.estimators_], axis=0)
elapsed_time = time.time() - start_time

print(f"Elapsed time to compute the importances: {elapsed_time:.3f} seconds")
```

```
Elapsed time to compute the importances: 0.013 seconds
```

```
In [133]:
fig, ax = plt.subplots(figsize=(6,6))
feat_importances.nlargest(10).sort_values().plot(kind='barh')
plt.title("Top 10 Important Features")
plt.show()
```



Final score of Random-Forest Classifier on test data is 95.11%

```
In [134]:
rf_y_test_pred = RFC.predict(X_test)
```

```
In [135]:
rf_y_train_pred = RFC.predict(X_train)
```

```
In [136]:
rf_cm_train = confusion_matrix(Y_train, rf_y_train_pred,labels=RFC.classes_)
rf_cm_test = confusion_matrix(Y_test, rf_y_test_pred,labels=RFC.classes_)
```

```
In [137]:
print('Training confusion matrix:')
print(rf_cm_train)
```

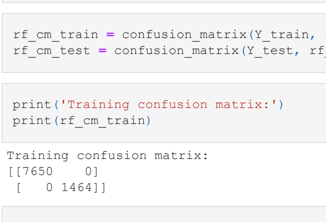
```
Training confusion matrix:
[[7650    0]
 [   0 1464]]
```

```
In [138]:
print('Test confusion matrix:')
print(rf_cm_test)
```

```
Test confusion matrix:
[[839  11]
 [ 31 132]]
```

```
In [139]:
disp = ConfusionMatrixDisplay(confusion_matrix=rf_cm_test,display_labels=RFC.classes_)
```

```
In [140]:
disp.plot()
plt.grid(False)
```




```
In [141]:
rfc_rep = classification_report(Y_test,rf_y_test_pred)
```

```
In [142]:
print("RFC performance report:")
print(rfc_rep)
```

```
RFC performance report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98       850
           1       0.92      0.81      0.86       163

    accuracy                           0.96      1013
   macro avg       0.94      0.90      0.92      1013
weighted avg       0.96      0.96      0.96      1013
```

```
In [143]:
rfc_f1score = {'0':0.97,'1':0.85}
rfc_prec = {'0':0.96,'1':0.91}
```

```
In [144]:
rfc_cval = cross_val_score(RFC,X_test,Y_test,scoring='accuracy')
rfc_cross_val = rfc_cval.mean()
rfc_cross_val
```

```
Out[144]: 0.9012437204311563
```

[illegible]

```
In [164]:
grid.fit(X_train,Y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.839 total time=   4.3s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.839 total time=   4.2s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.839 total time=   4.4s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.839 total time=   4.2s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.840 total time=   4.3s
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.839 total time=   4.8s
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.839 total time=   5.2s
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.839 total time=   4.9s
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.839 total time=   5.3s
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.840 total time=   4.6s
[CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.839 total time=   4.5s
[CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.839 total time=   4.5s
[CV 3/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.839 total time=   4.4s
[CV 4/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.839 total time=   4.9s
[CV 5/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.840 total time=   4.7s
[CV 1/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.839 total time=   4.5s
[CV 2/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.839 total time=   4.3s
[CV 3/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.839 total time=   4.4s
[CV 4/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.839 total time=   4.8s
[CV 5/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.840 total time=   4.3s
[CV 1/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.839 total time=   4.3s
[CV 2/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.839 total time=   4.2s
[CV 3/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.839 total time=   4.3s
[CV 4/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.839 total time=   4.2s
[CV 5/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.840 total time=   4.4s
[CV 1/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.839 total time=   5.2s
[CV 2/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.839 total time=   4.7s
[CV 3/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.839 total time=   4.5s
[CV 4/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.839 total time=   4.4s
[CV 5/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.840 total time=   4.4s
[CV 1/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.839 total time=   4.5s
[CV 2/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.839 total time=   4.5s
[CV 3/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.839 total time=   4.7s
[CV 4/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.839 total time=   4.4s
[CV 5/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.840 total time=   4.6s
[CV 1/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.839 total time=   4.5s
[CV 2/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.839 total time=   4.6s
[CV 3/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.839 total time=   4.5s
[CV 4/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.839 total time=   4.5s
[CV 5/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.840 total time=   4.4s
[CV 1/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.839 total time=   4.4s
[CV 2/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.839 total time=   4.7s
[CV 3/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.839 total time=   5.2s
[CV 4/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.839 total time=   4.6s
[CV 5/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.840 total time=   4.6s
[CV 1/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.839 total time=   4.5s
[CV 2/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.839 total time=   4.4s
[CV 3/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.839 total time=   4.6s
[CV 4/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.839 total time=   4.6s
[CV 5/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.840 total time=   4.4s
[CV 1/5] END .........C=10, gamma=1, kernel=rbf;, score=0.839 total time=   9.4s
[CV 2/5] END .........C=10, gamma=1, kernel=rbf;, score=0.839 total time=   8.7s
[CV 3/5] END .........C=10, gamma=1, kernel=rbf;, score=0.839 total time=   8.7s
[CV 4/5] END .........C=10, gamma=1, kernel=rbf;, score=0.839 total time=   8.3s
[CV 5/5] END .........C=10, gamma=1, kernel=rbf;, score=0.840 total time=   9.2s
[CV 1/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.839 total time=  11.1s
[CV 2/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.839 total time=  10.2s
[CV 3/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.839 total time=   8.5s
[CV 4/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.839 total time=   8.3s
[CV 5/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.840 total time=   8.4s
[CV 1/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.839 total time=   8.1s
[CV 2/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.839 total time=   9.1s
[CV 3/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.839 total time=   9.8s
[CV 4/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.839 total time=   8.3s
[CV 5/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.840 total time=   8.3s
[CV 1/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.839 total time=   8.2s
[CV 2/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.839 total time=   8.6s
[CV 3/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.839 total time=   8.6s
[CV 4/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.839 total time=   9.5s
[CV 5/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.840 total time=  10.0s
[CV 1/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.839 total time=   9.2s
[CV 2/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.839 total time=   8.4s
[CV 3/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.839 total time=   8.3s
[CV 4/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.839 total time=   8.6s
[CV 5/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.840 total time=   9.5s
[CV 1/5] END ........C=100, gamma=1, kernel=rbf;, score=0.839 total time=  11.7s
[CV 2/5] END ........C=100, gamma=1, kernel=rbf;, score=0.839 total time=   7.9s
[CV 3/5] END ........C=100, gamma=1, kernel=rbf;, score=0.839 total time=   7.8s
[CV 4/5] END ........C=100, gamma=1, kernel=rbf;, score=0.839 total time=   8.6s
[CV 5/5] END ........C=100, gamma=1, kernel=rbf;, score=0.840 total time=   9.2s
[CV 1/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.839 total time=   8.1s
[CV 2/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.839 total time=   8.3s
[CV 3/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.839 total time=   8.0s
[CV 4/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.839 total time=   9.7s
[CV 5/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.840 total time=  10.7s
[CV 1/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.839 total time=   8.1s
[CV 2/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.839 total time=   8.6s
[CV 3/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.839 total time=   8.5s
[CV 4/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.839 total time=   8.7s
[CV 5/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.840 total time=   8.6s
[CV 1/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.839 total time=   8.1s
[CV 2/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.839 total time=   8.3s
[CV 3/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.839 total time=   9.5s
[CV 4/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.839 total time=   9.5s
[CV 5/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.840 total time=   8.7s
[CV 1/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.839 total time=   8.2s
[CV 2/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.839 total time=   8.7s
[CV 3/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.839 total time=   8.2s
[CV 4/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.839 total time=   9.8s
[CV 5/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.840 total time=   9.1s
[CV 1/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.839 total time=   8.5s
[CV 2/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.839 total time=   8.5s
[CV 3/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.839 total time=   8.2s
[CV 4/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.839 total time=   8.7s
[CV 5/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.840 total time=  10.2s
[CV 1/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.839 total time=   8.8s
[CV 2/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.839 total time=   8.5s
[CV 3/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.839 total time=   8.3s
[CV 4/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.839 total time=   8.3s
[CV 5/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.840 total time=   8.7s
[CV 1/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.839 total time=   8.4s
[CV 2/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.839 total time=   8.2s
[CV 3/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.839 total time=   8.1s
[CV 4/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.839 total time=   8.4s
[CV 5/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.840 total time=   8.7s
[CV 1/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.839 total time=   8.3s
[CV 2/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.839 total time=   8.3s
[CV 3/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.839 total time=   8.2s
[CV 4/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.839 total time=   8.3s
[CV 5/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.840 total time=   8.6s
[CV 1/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.839 total time=   8.2s
[CV 2/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.839 total time=   9.1s
[CV 3/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.839 total time=   8.8s
[CV 4/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.839 total time=   8.7s
[CV 5/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.840 total time=   8.8s
```

```
Out[164]: GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf']},
             verbose=3)
```

```
In [165]:
print(grid.best_params_)
```

```
{'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
```

```
In [166]:
print(grid.best_estimator_)
```

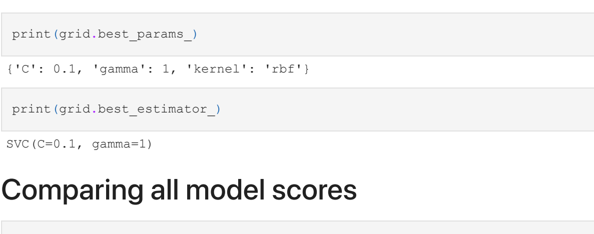
```
SVC(C=0.1, gamma=1)
```

Comparing all model scores

```
In [167]:
model_scores = {"Support_Vector_Machine":SVM_score,
                "Random_Forest_Classifier":RFC_score,
                "k_Nearest_Neighbours":KNN_score,
                "Decision_Tree":DT_score}
```

```
In [168]:
keys = model_scores.keys()
values = model_scores.values()
```

```
In [169]:
plt.figure(figsize=(10,4))
plt.bar(keys,values,width=0.4)
plt.xticks(rotation=90)
plt.show()
```



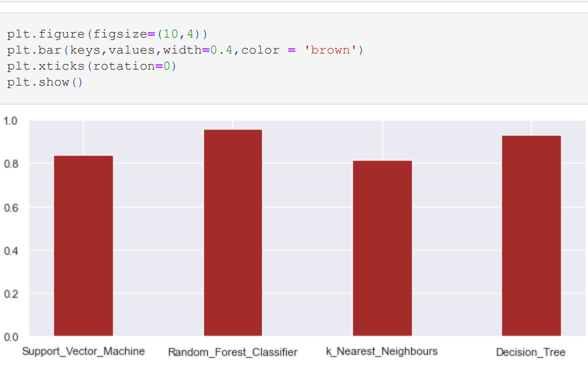

```
In [170]:
print('Model                 Scores')
for m,s in model_scores.items():
    print('{} {}'.format(m, s))
```

```
Model                 Scores
Support_Vector_Machine 0.8390918065153011
Random_Forest_Classifier 0.9585389930898321
k_Nearest_Neighbours 0.8163869693978283
Decision_Tree 0.930898321816387
```

```
In [171]:
model_cross_val_scores= {"Support_Vector_Machine":svm_cross_val,
                "Random_Forest_Classifier":rfc_cross_val,
                "k_Nearest_Neighbours":knn_cross_val,
                "Decision_Tree":dtree_cross_val}
```

```
In [172]:
keys = model_scores.keys()
values = model_scores.values()
```

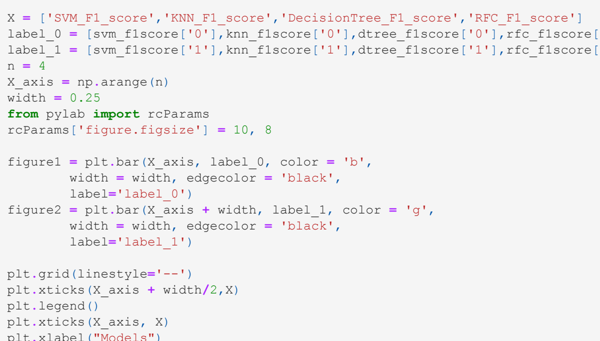
```
In [173]:
plt.figure(figsize=(10,4))
plt.bar(keys,values,width=0.4,color = 'brown')
plt.xticks(rotation=0)
plt.show()
```




```
In [174]:
print(svm_f1score,knn_f1score,dtree_f1score,rfc_f1score)
```

```
{'0': 0.91, '1': 0.0} {'0': 0.9, '1': 0.1} {'0': 0.96, '1': 0.77} {'0': 0.97, '1': 0.85}
```

```
In [175]:
X = ['SVM_F1_score','KNN_F1_score','DecisionTree_F1_score','RFC_F1_score']
label_0 = [svm_f1score['0'],knn_f1score['0'],dtree_f1score['0'],rfc_f1score['0']]
label_1 = [svm_f1score['1'],knn_f1score['1'],dtree_f1score['1'],rfc_f1score['1']]
n = 4
X_axis = np.arange(n)
width = 0.25
from pylab import rcParams
rcParams['figure.figsize'] = 10, 8

figure1 = plt.bar(X_axis, label_0, color = 'b',
        width = width, edgecolor = 'black',
        label='label_0')
figure2 = plt.bar(X_axis + width, label_1, color = 'g',
        width = width, edgecolor = 'black',
        label='label_1')

plt.grid(linestyle='--')
plt.xticks(X_axis + width/2,X)
plt.legend()
plt.xticks(X_axis, X)
plt.xlabel("Models")
plt.ylabel("F1_measures")
plt.title("F1 scores of models")
plt.legend()


plt.show()
```




```
In [176]:
X = ['SVM','KNN','DecisionTree','RFC']
label_0 = [svm_prec['0'],knn_prec['0'],dtree_prec['0'],rfc_prec['0']]
label_1 = [svm_prec['1'],knn_prec['1'],dtree_prec['1'],rfc_prec['1']]
n = 4
X_axis = np.arange(n)
width = 0.25
from pylab import rcParams
rcParams['figure.figsize'] = 10, 8

figure1 = plt.bar(X_axis, label_0, color = 'r',
        width = width, edgecolor = 'black',
        label='label_0')
figure2 = plt.bar(X_axis + width, label_1, color = 'y',
        width = width, edgecolor = 'black',
        label='label_1')

plt.grid(linestyle='--')
plt.xticks(X_axis + width/2,X)
plt.legend()
plt.xticks(X_axis, X)
plt.xlabel("Models")
plt.ylabel("Precision")
plt.title("Precision of models")
plt.legend()


plt.show()
```

RandomForest Classifier performs best comparitively with all other models

```
In [ ]:
```