In [1]:

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

In [2]:

```
df = pd.read_csv('insurance.csv')
df.head()
```

Out[2]:

	age	sex	bmi	children	smoker	region	charges	insuranceclaim
0	19	0	27.900	0	1	3	16884.92400	1
1	18	1	33.770	1	0	2	1725.55230	1
2	28	1	33.000	3	0	2	4449.46200	0
3	33	1	22.705	0	0	1	21984.47061	0
4	32	1	28.880	0	0	1	3866.85520	1

In [3]:

```
# checking first five rows

df.head()
```

Out[3]:

	age	sex	bmi	children	smoker	region	charges	insuranceclaim
0	19	0	27.900	0	1	3	16884.92400	1
1	18	1	33.770	1	0	2	1725.55230	1
2	28	1	33.000	3	0	2	4449.46200	0
3	33	1	22.705	0	0	1	21984.47061	0
4	32	1	28.880	0	0	1	3866.85520	1

In [4]:

```
# checking last five rows

df.tail()
```

Out[4]:

	age	sex	bmi	children	smoker	region	charges	insuranceclaim
1333	50	1	30.97	3	0	1	10600.5483	0
1334	18	0	31.92	0	0	0	2205.9808	1
1335	18	0	36.85	0	0	2	1629.8335	1
1336	21	0	25.80	0	0	3	2007.9450	0
1337	61	0	29.07	0	1	1	29141.3603	1

In [5]:

```
# shape of dataset

print('number of rows :',df.shape[0])
print('number of columns :',df.shape[1])
```

```
number of rows : 1338
number of columns : 8
```

In [6]:

```
# total number of datapoints in our dataset

print('size of our dataset :',df.size)
```

```
size of our dataset : 10704
```

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
age               0
sex               0
bmi               0
children          0
smoker            0
region            0
charges           0
insuranceclaim    0
dtype: int64
```

In [8]:

```
# checking for information of our dataset
# number of columns
# type of columns
# their datatypes

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             1338 non-null   int64  
 1   sex             1338 non-null   int64  
 2   bmi             1338 non-null   float64
 3   children        1338 non-null   int64  
 4   smoker          1338 non-null   int64  
 5   region          1338 non-null   int64  
 6   charges         1338 non-null   float64
 7   insuranceclaim  1338 non-null   int64  
dtypes: float64(2), int64(6)
memory usage: 83.8 KB
```

Insight :

1. there are total 8 columns

2. there are no null values
3. 6 columns are of int64 datatype
4. 2 columns are of float64 datatype
5. there are no columns with object or boolean datatype

In [9]:

```
df.describe()
```

Out[9]:

	age	sex	bmi	children	smoker	region	charges
count	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	0.505232	30.663397	1.094918	0.204783	1.515695	13270.422265
std	14.049960	0.500160	6.098187	1.205493	0.403694	1.104885	12110.011237
min	18.000000	0.000000	15.960000	0.000000	0.000000	0.000000	1121.873900
25%	27.000000	0.000000	26.296250	0.000000	0.000000	1.000000	4740.287150
50%	39.000000	1.000000	30.400000	1.000000	0.000000	2.000000	9382.033000
75%	51.000000	1.000000	34.693750	2.000000	0.000000	2.000000	16639.912515
max	64.000000	1.000000	53.130000	5.000000	1.000000	3.000000	63770.428010

insights :

1. mix age is 18 and max age is 64
2. maximum customers are of mid age between 35-50
3. min claim settlement is 1121
4. max claim settlement is 63770
5. almost equal number of male and female customers
6. if the customer has higher bmi and is smoker then theere is high problality of claim
7. many of the customers has 1-2 children

In [ ]:

```

```

creating a base model without any changes

Splitting the dataset between independent variables and dependent variable

In [10]:

```
X_1 = df.drop(['insuranceclaim'],axis = 1)
y_1 = df['insuranceclaim']
```

In [11]:

```
# splitting data into train and test

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X_1,y_1,test_size = 0.20,random_state = 42)
```

In [12]:

```
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
```

In [13]:

```
# Ogistic regression

lr1 = LogisticRegression()

# fit on data

lr1.fit(X_train,y_train)
```

Out[13]:

```
LogisticRegression()
```

In [14]:

```
# predict

ypr1 = lr1.predict(X_test)
```

In [15]:

```
# accuracy

ac1 = accuracy_score(y_test,ypr1)
ac1
```

Out[15]:

```
0.7947761194029851
```

svm

In [16]:

```
# instance

svm1 = svm.SVC()

# fit on data

svm1.fit(X_train,y_train)
```

Out[16]:

```
SVC()
```

In [17]:

```
# predict

ypr2 = svm1.predict(X_test)

# accuracy

ac2 = accuracy_score(y_test,ypr2)
ac2
```

Out[17]:

```
0.6082089552238806
```

random forest classifier

In [18]:

```
# creeating instances

rf1 = RandomForestClassifier()

# fit on data

rf1.fit(X_train,y_train)
```

Out[18]:

```
RandomForestClassifier()
```

In [19]:

```
# predict

ypr3 = rf1.predict(X_test)

# accuracy

ac3 = accuracy_score(y_test,ypr3)
ac3
```

Out[19]:

```
0.914179104477612
```

Decision Tree Classifier

In [20]:

```
# creating instance

dt1 = DecisionTreeClassifier()

# fit on data

dt1.fit(X_train,y_train)
```

Out[20]:

```
DecisionTreeClassifier()
```

In [21]:

```
# predict

ypr4 = dt1.predict(X_test)

# accuracy score

ac4 = accuracy_score(y_test,ypr4)
ac4
```

Out[21]:

```
0.9776119402985075
```

Gradient boosting

In [22]:

```
# creating instance

gbc1 = GradientBoostingClassifier()

# fit on data

gbc1.fit(X_train,y_train)
```

Out[22]:

```
GradientBoostingClassifier()
```

In [23]:

```
# predict

ypr5 = gbc1.predict(X_test)

# accuracy

ac5 = accuracy_score(y_test,ypr5)
ac5
```

Out[23]:

```
0.9701492537313433
```

K-Nearest Neighbor

In [24]:

```
# creating instance

knn1 = KNeighborsClassifier()

# fit on data

knn1.fit(X_train,y_train)
```

Out[24]:

```
KNeighborsClassifier()
```

In [25]:

```
# predict

ypr6 = knn1.predict(X_test)

# accuracy score

ac6 = accuracy_score(y_test,ypr6)
ac6
```

```
C:\Users\WOLF\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:22
8: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the
default behavior of `mode` typically preserves the axis it acts along. In SciPy
1.11.0, this behavior will change: the default value of `keepdims` will become Fa
lse, the `axis` over which the statistic is taken will be eliminated, and the val
ue None will no longer be accepted. Set `keepdims` to True or False to avoid this
warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Out[25]:

```
0.6380597014925373
```

In [26]:

```
f_df = pd.DataFrame({'models':['lr1','svm1','rf1','dt1','gbc1','knn1'],
                    'accuracy':[ac1,ac2,ac3,ac4,ac5,ac6]})
```

In [27]:

```
f_df
```

Out[27]:

	models	accuracy
0	lr1	0.794776
1	svm1	0.608209
2	rf1	0.914179
3	dt1	0.977612
4	gbc1	0.970149
5	knn1	0.638060

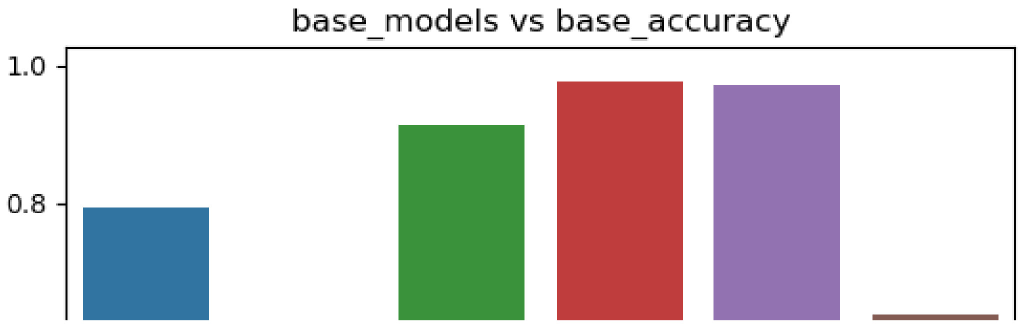
In [28]:

```
sns.barplot(f_df['models'],f_df['accuracy'])
plt.title('base_models vs base_accuracy')
```

```
C:\Users\WOLF\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWar
ning: Pass the following variables as keyword args: x, y. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[28]:

```
Text(0.5, 1.0, 'base_models vs base_accuracy')
```



Insight :

1. As we can see our model iis showing very high accuracy
2. But as this is just a base model without any alterations using eda and others
3. therefore it wont be a great idea to use it. So,that we will make another one

In [29]:

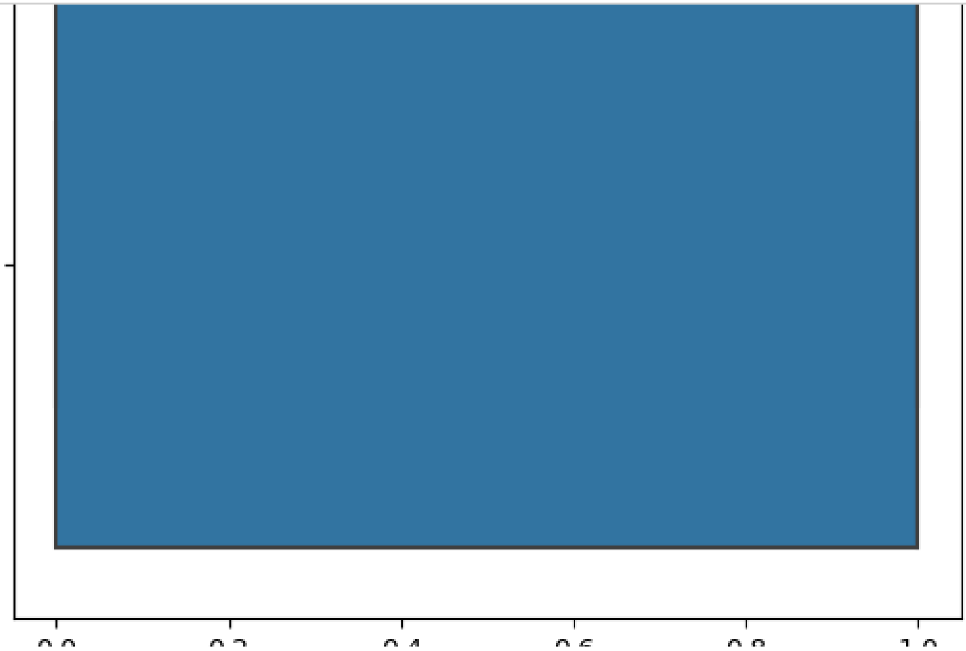
```
df.columns
```

Out[29]:

```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges',
       'insuranceclaim'],
      dtype='object')
```

In [30]:

```
columns = ['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges','insuranceclaim']
for i in columns:
    plt.figure()
    sns.boxplot(df[i])
```



In [31]:

```
np.where(df['charges']>=35000)
```

Out[31]:

```
(array([  14,   19,   23,   29,   30,   34,   38,   39,   49,   53,   55,
          82,   84,   86,   94,  109,  123,  146,  158,  161,  175,  185,
         203,  240,  242,  251,  252,  254,  256,  263,  265,  271,  281,
         288,  292,  298,  312,  322,  327,  328,  330,  338,  373,  377,
         381,  420,  421,  422,  441,  476,  488,  500,  524,  530,  543,
         549,  558,  569,  577,  587,  609,  615,  621,  629,  665,  667,
         668,  674,  677,  682,  697,  706,  725,  736,  738,  739,  742,
         759,  803,  819,  826,  828,  842,  845,  850,  852,  856,  860,
         883,  893,  901,  917,  947,  951,  953,  956,  958, 1012, 1021,
        1022, 1031, 1036, 1037, 1047, 1049, 1062, 1070, 1090, 1096, 1111,
        1117, 1118, 1122, 1124, 1139, 1146, 1152, 1156, 1186, 1206, 1207,
        1218, 1230, 1240, 1241, 1249, 1284, 1288, 1300, 1301, 1303, 1313,
        1323], dtype=int64),)
```

In [32]:

```
np.where(df['bmi']>=46)
```

Out[32]:

```
(array([ 116,  286,  401,  438,  454,  543,  547,  549,  660,  847,  860,
         930,  941, 1047, 1088, 1317], dtype=int64),)
```

Insight :

As we can see there are some outlliers in these features but as they are very low we wont consider doing any operations with it. i am not going to drop the outliers of charges as we already have as they are very low in numbers

In [ ]:

```

```

New Model

In [33]:

```
X = df.drop(['insuranceclaim'],axis = 1)
y = df['insuranceclaim']
```

In [34]:

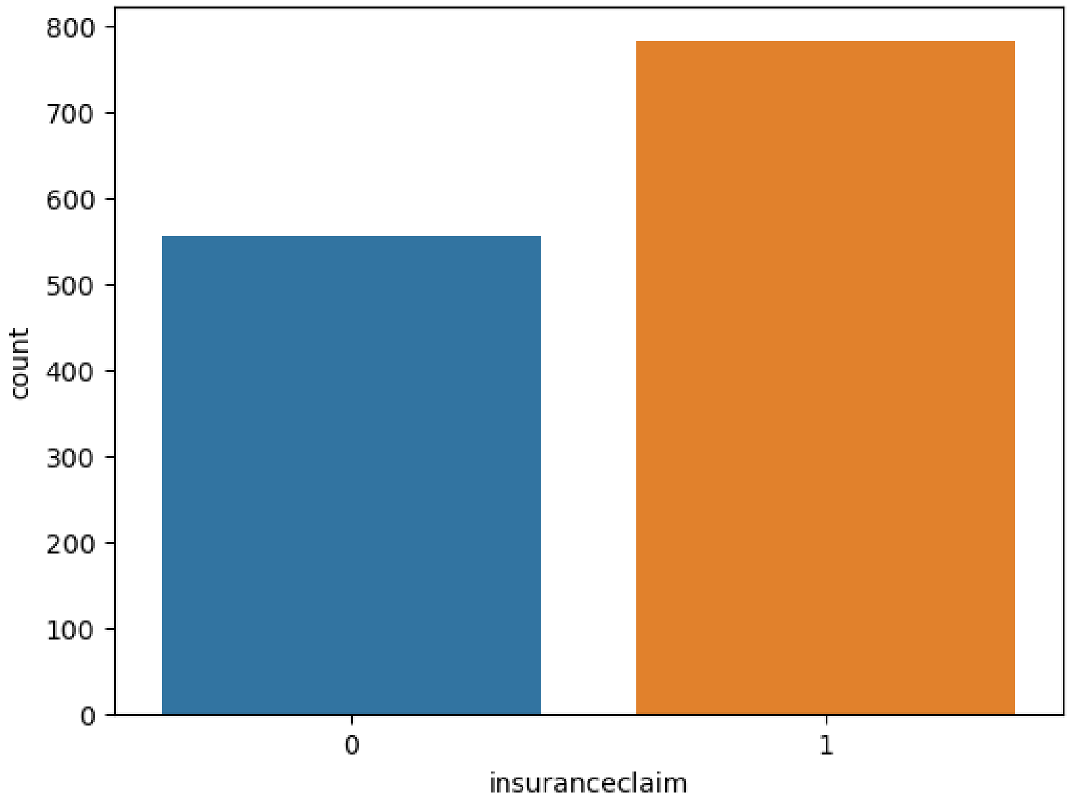
```
# chech for equal distribution

sns.countplot(y)
```

```
C:\Users\WOLF\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only v
alid positional argument will be `data`, and passing other arguments without an e
xplicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[34]:

```
<AxesSubplot:xlabel='insuranceclaim', ylabel='count'>
```



In [35]:

```
# handling imbalanced dataset with smote
# smote syntesizes new minority instances (it creates artificial datapoints that are slightli s
!pip install imblearn

from imblearn.over_sampling import SMOTE
```

```
Requirement already satisfied: imblearn in c:\users\wolf\anaconda3\lib\site-packa
ges (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\wolf\anaconda3\lib\si
te-packages (from imblearn) (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in c:\users\wolf\anaconda3\lib\site-
packages (from imbalanced-learn->imblearn) (1.21.5)
Requirement already satisfied: joblib>=1.1.1 in c:\users\wolf\anaconda3\lib\site-
packages (from imbalanced-learn->imblearn) (1.2.0)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\wolf\anaconda3\lib
\site-packages (from imbalanced-learn->imblearn) (1.0.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\wolf\anaconda3\li
b\site-packages (from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: scipy>=1.3.2 in c:\users\wolf\anaconda3\lib\site-p
ackages (from imbalanced-learn->imblearn) (1.9.1)
```

In [36]:

```
X_res,y_res = SMOTE().fit_resample(X,y)
```

In [37]:

```
y_res.value_counts()
```

Out[37]:

```
1    783
0    783
Name: insuranceclaim, dtype: int64
```

splitting between training and test data

In [38]:

```
# importing train_test_split for data splitting

from sklearn.model_selection import train_test_split
```

In [39]:

```
X_train,X_test,y_train,y_test = train_test_split(X_res,y_res,test_size = 0.20,random_state = 42
```

In [40]:

```
print('X_train shape :',X_train.shape)
print('X_test shape :',X_test.shape)
print('y_train shape :',y_train.shape)
print('y_test shape :',y_test.shape)
```

```
X_train shape : (1252, 7)
X_test shape : (314, 7)
y_train shape : (1252,)
y_test shape : (314,)
```

In [41]:

```
# Feature scaling


from sklearn.preprocessing import StandardScaler

std = StandardScaler()

X_train = std.fit_transform(X_train)
X_test = std.transform(X_test)
```

In [42]:

```
# we have to check to confirm scaling is done

X_train
```

Out[42]:

```
array([[-0.5639978 , -0.92902919, -0.18429644, ..., -0.45916416,
         1.43046451, -0.76494058],
       [ 1.31553296,  1.07639244,  2.25251124, ..., -0.45916416,
         1.43046451, -0.08861016],
       [-1.28689425,  1.07639244,  1.11088353, ..., -0.45916416,
        -0.44094778, -0.92452615],
       ...,
       [-0.13025994, -0.92902919,  2.90343932, ...,  2.17787035,
         1.43046451,  2.90040653],
       [-1.50376319,  1.07639244, -1.15130979, ..., -0.45916416,
        -1.37665393, -0.94290046],
       [ 1.17095367,  1.07639244, -0.05077273, ..., -0.45916416,
         1.43046451, -0.20643984]])
```

As we can see that our target variable has two classes therefore this is a classification problem

Now we will import different classification algorithms for model prediction

In [43]:

```
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

Logistic Regression

In [44]:

```
# creating instance

lr = LogisticRegression()

# fit on data

lr.fit(X_train,y_train)
```

Out[44]:

```
LogisticRegression()
```

In [45]:

```
# prediction
y_pred1 = lr.predict(X_test)
```

In [46]:

```
# checking model prdiction accuracy

from sklearn.metrics import accuracy_score
```

In [47]:

```
acc1 = accuracy_score(y_test,y_pred1)
acc1
```

Out[47]:

```
0.8789808917197452
```

In [48]:

```
# as our data set is imbalanced it is very fdangeroyus to use sccuracy score
# therefore we will use precision,recall and f1 score

from sklearn.metrics import precision_score,recall_score,f1_score
```

In [49]:

```
# as we know precision is  P = TP/(FP + TP)

ps1 = precision_score(y_test,y_pred1)
ps1
```

Out[49]:

```
0.9060402684563759
```

In [50]:

```
# as we know recalll is R = TP/(TP+FN)

rs1 = recall_score(y_test,y_pred1)
rs1
```

Out[50]:

```
0.8490566037735849
```

In [51]:

```
# f1 score

f1_s1 = f1_score(y_test,y_pred1)
f1_s1
```

Out[51]:

```
0.8766233766233767
```

Support vector machine

In [52]:

```
# creating instance

svm = svm.SVC()

# fit on data

svm.fit(X_train,y_train)
```

Out[52]:

```
SVC()
```

In [53]:

```
# prediction

y_pred2 = svm.predict(X_test)
```

In [54]:

```
# checking accuracy

acc2 = accuracy_score(y_test,y_pred2)
acc2
```

Out[54]:

```
0.8949044585987261
```

In [55]:

```
# precision

ps2 = precision_score(y_test,y_pred2)
ps2
```

Out[55]:

```
0.9256756756756757
```

In [56]:

```
# recall

rs2 = recall_score(y_test,y_pred2)
rs2
```

Out[56]:

```
0.8616352201257862
```

In [57]:

```
# f1 score

f1_s2 = f1_score(y_test,y_pred2)
f1_s2
```

Out[57]:

```
0.8925081433224755
```

Random Forest Classifier

In [58]:

```
# creating instance

rfc = RandomForestClassifier()

# fit on data

rfc.fit(X_train,y_train)
```

Out[58]:

```
RandomForestClassifier()
```

In [59]:

```
# prediction

y_pred3 = rfc.predict(X_test)
```

In [60]:

```
# accuracy

acc3 = accuracy_score(y_test,y_pred3)
acc3
```

Out[60]:

```
0.9426751592356688
```

In [61]:

```
# precision

ps3 = precision_score(y_test,y_pred3)
ps3
```

Out[61]:

```
0.9795918367346939
```

In [62]:

```
# recall

rs3 = recall_score(y_test,y_pred3)
rs3
```

Out[62]:

```
0.9056603773584906
```

In [63]:

```
# f1 score

f1_s3 = f1_score(y_test,y_pred3)
f1_s3
```

Out[63]:

```
0.9411764705882353
```

Decision Tree Classifier

In [64]:

```
# creating instance

dtc = DecisionTreeClassifier()

# fit on data

dtc.fit(X_train,y_train)
```

Out[64]:

```
DecisionTreeClassifier()
```

In [65]:

```
# prediction

y_pred4 = dtc.predict(X_test)
```

In [66]:

```
# accuracy

acc4 = accuracy_score(y_test,y_pred4)
acc4
```

Out[66]:

```
0.945859872611465
```

In [67]:

```
# precision

ps4 = precision_score(y_test,y_pred4)
ps4
```

Out[67]:

```
0.9551282051282052
```

In [68]:

```
# recall

rs4 = recall_score(y_test,y_pred4)
rs4
```

Out[68]:

```
0.9371069182389937
```

In [69]:

```
# f1 score

f1_s4 = f1_score(y_test,y_pred4)
f1_s4
```

Out[69]:

```
0.946031746031746
```

K-Nearest Classifier

In [70]:

```
# creating instance

knn = KNeighborsClassifier()

# fit on data

knn.fit(X_train,y_train)
```

Out[70]:

```
KNeighborsClassifier()
```

In [71]:

```
# predict

y_pred5 = knn.predict(X_test)
```

```
C:\Users\WOLF\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:22
8: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the
default behavior of `mode` typically preserves the axis it acts along. In SciPy
1.11.0, this behavior will change: the default value of `keepdims` will become Fa
lse, the `axis` over which the statistic is taken will be eliminated, and the val
ue None will no longer be accepted. Set `keepdims` to True or False to avoid this
warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [72]:

```
# accuracy

acc5 = accuracy_score(y_test,y_pred5)
acc5
```

Out[72]:

```
0.8949044585987261
```

As we have to assign values of k neighbors

In [73]:

```
score = []

for k in range(1,40):
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train,y_train)
    y_pred = knn.predict(X_test)
    score.append(accuracy_score(y_test,y_pred))
```

```
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\WOLF\anaconda3\lib\site-packages\sklearn\neighbors\_classification.p
y:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis
`), the default behavior of `mode` typically preserves the axis it acts along.
In SciPy 1.11.0, this behavior will change: the default value of `keepdims` wi
ll become False, the `axis` over which the statistic is taken will be eliminat
ed, and the value None will no longer be accepted. Set `keepdims` to True or F
alse to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\WOLF\anaconda3\lib\site-packages\sklearn\neighbors\_classification.p
y:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis
`), the default behavior of `mode` typically preserves the axis it acts along.
In SciPy 1.11.0, this behavior will change: the default value of `keepdims` wi
ll become False, the `axis` over which the statistic is taken will be eliminat
ed, and the value None will no longer be accepted. Set `keepdims` to True or F
alse to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\WOLF\anaconda3\lib\site-packages\sklearn\neighbors\_classification.p
y:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis
`), the default behavior of `mode` typically preserves the axis it acts along.
```

In [74]:

```
score
```

Out[74]:

```
[0.856687898089172,
 0.8630573248407644,
 0.8757961783439491,
 0.8821656050955414,
 0.8949044585987261,
 0.8885350318471338,
 0.8949044585987261,
 0.8853503184713376,
 0.8885350318471338,
 0.8885350318471338,
 0.8821656050955414,
 0.8789808917197452,
 0.8694267515923567,
 0.8630573248407644,
 0.8662420382165605,
 0.8789808917197452,
 0.8789808917197452,
 0.8757961783439491,
 0.8789808917197452,
 0.8789808917197452,
 0.8789808917197452,
 0.8757961783439491,
 0.8789808917197452,
 0.8694267515923567,
 0.8821656050955414,
 0.8821656050955414,
 0.8757961783439491,
 0.8598726114649682,
 0.8598726114649682,
 0.8630573248407644,
 0.8598726114649682,
 0.8598726114649682,
 0.8598726114649682,
 0.8535031847133758,
 0.8535031847133758,
 0.8535031847133758,
 0.8535031847133758,
 0.856687898089172,
 0.8535031847133758]
```

In [75]:

```
# as we are getting better accuracy for 5 neighbors as we can see in score
# therefore we will choose 5 neighbors

for k in range(1,40):
    knn = KNeighborsClassifier(n_neighbors = 5)
    knn.fit(X_train,y_train)
    y_pred = knn.predict(X_test)
    score.append(accuracy_score(y_test,y_pred))
```

```
C:\Users\WOLF\anaconda3\lib\site-packages\sklearn\neighbors\_classification.p
y:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis
`), the default behavior of `mode` typically preserves the axis it acts along.
In SciPy 1.11.0, this behavior will change: the default value of `keepdims` wi
ll become False, the `axis` over which the statistic is taken will be eliminat
ed, and the value None will no longer be accepted. Set `keepdims` to True or F
alse to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\WOLF\anaconda3\lib\site-packages\sklearn\neighbors\_classification.p
y:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis
`), the default behavior of `mode` typically preserves the axis it acts along.
In SciPy 1.11.0, this behavior will change: the default value of `keepdims` wi
ll become False, the `axis` over which the statistic is taken will be eliminat
ed, and the value None will no longer be accepted. Set `keepdims` to True or F
alse to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\WOLF\anaconda3\lib\site-packages\sklearn\neighbors\_classification.p
y:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis
`), the default behavior of `mode` typically preserves the axis it acts along.
```

In [76]:

```
acc5 = accuracy_score(y_test,y_pred5)
acc5
```

Out[76]:

```
0.8949044585987261
```

In [77]:

```

# precision

ps5 = precision_score(y_test,y_pred5)
ps5
```

Out[77]:

```
0.9090909090909091
```

In [78]:

```
# reacll

rs5 = recall_score(y_test,y_pred5)
rs5
```

Out[78]:

```
0.8805031446540881
```

In [79]:

```
# f1_score

f1_s5 = f1_score(y_test,y_pred5)
f1_s5
```

Out[79]:

```
0.8945686900958466
```

Gradient Boosting Classifier

In [80]:

```
# creating instance

gbc = GradientBoostingClassifier()

# fit on data

gbc.fit(X_train,y_train)
```

Out[80]:

```
GradientBoostingClassifier()
```

In [81]:

```
# prediction

y_pred6 = gbc.predict(X_test)

# acuracy

acc6 = accuracy_score(y_test,y_pred)
acc6
```

Out[81]:

```
0.8949044585987261
```

In [82]:

```
# precision

ps6 = precision_score(y_test,y_pred6)
ps6
```

Out[82]:

```
0.9655172413793104
```

In [83]:

```
# reacll

rs6 = recall_score(y_test,y_pred6)
rs6
```

Out[83]:

```
0.8805031446540881
```

In [84]:

```
# fi score

f1_s6 = f1_score(y_test,y_pred6)
f1_s6
```

Out[84]:

```
0.9210526315789475
```

In [85]:

```
final_df = pd.DataFrame({'Models':['LR','SVC','RF','DT','KNN','GBC'],
                        'ACC':[acc1,acc2,acc3,acc4,acc5,acc6],
                        'PS':[ps1,ps2,ps3,ps4,ps5,ps6],
                        'RS':[rs1,rs2,rs3,rs4,rs5,rs6],
                        'F1':[f1_s1,f1_s2,f1_s3,f1_s4,f1_s5,f1_s5]})
```

In [86]:

```
final_df
```

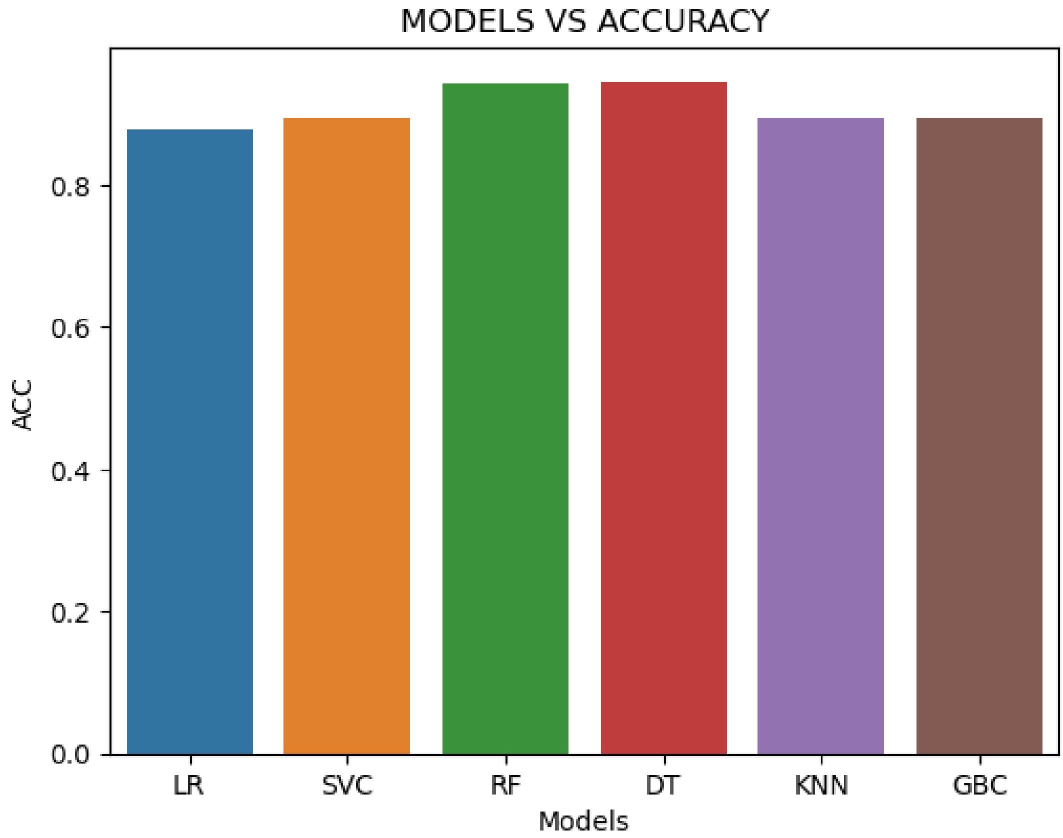
Out[86]:

	Models	ACC	PS	RS	F1
0	LR	0.878981	0.906040	0.849057	0.876623
1	SVC	0.894904	0.925676	0.861635	0.892508
2	RF	0.942675	0.979592	0.905660	0.941176
3	DT	0.945860	0.955128	0.937107	0.946032
4	KNN	0.894904	0.909091	0.880503	0.894569
5	GBC	0.894904	0.965517	0.880503	0.894569

In [87]:

```
sns.barplot(final_df['Models'],final_df['ACC'])
plt.title('MODELS VS ACCURACY')
plt.show()
```

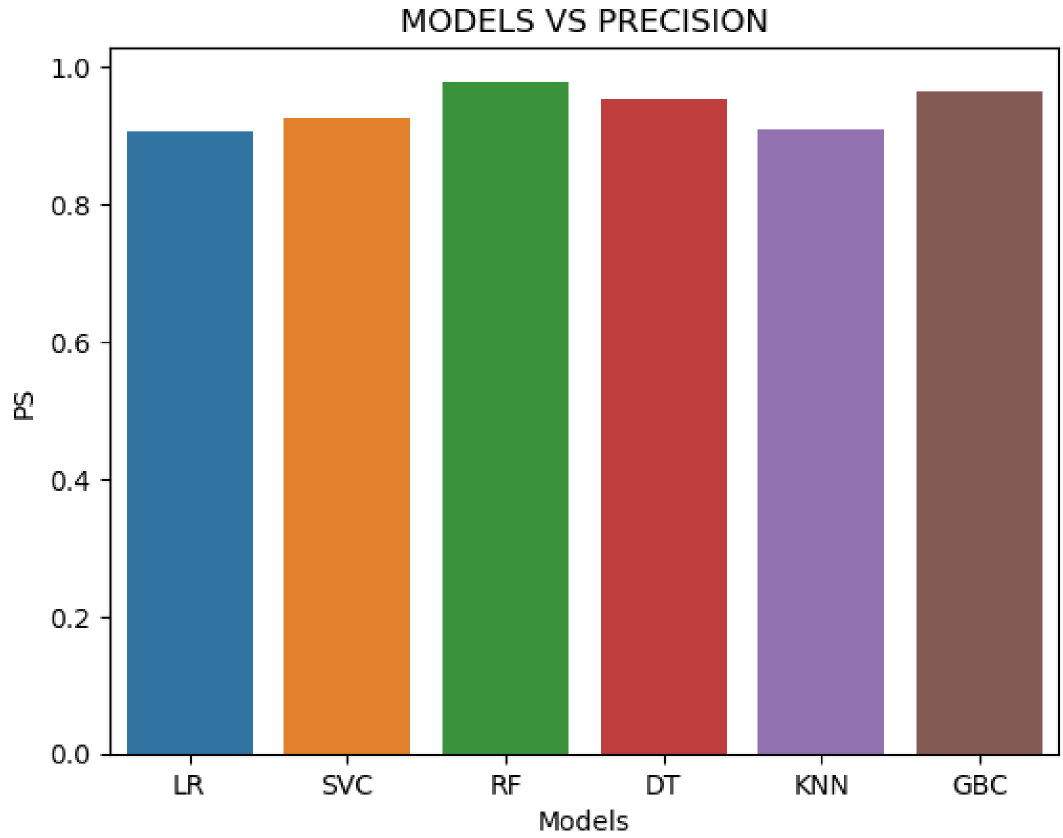
```
C:\Users\WOLF\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [88]:

```
sns.barplot(final_df['Models'],final_df['PS'])
plt.title('MODELS VS PRECISION')
plt.show()
```

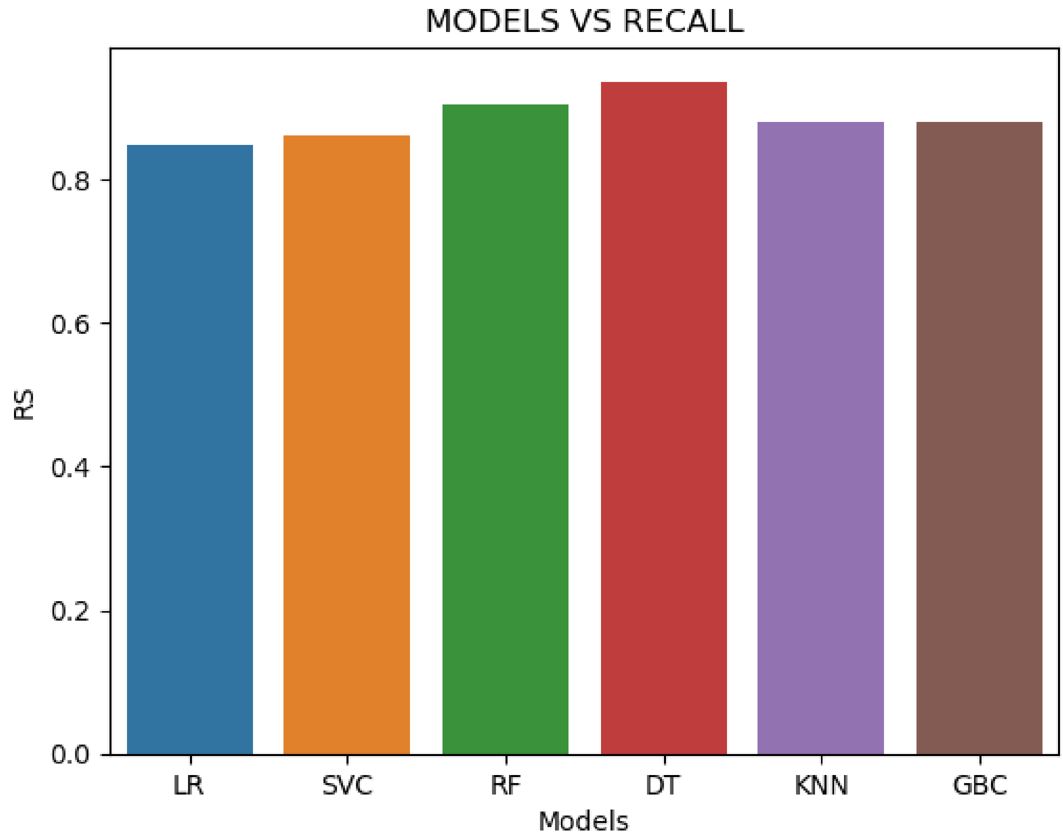
```
C:\Users\WOLF\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [89]:

```
sns.barplot(final_df['Models'],final_df['RS'])
plt.title('MODELS VS RECALL')
plt.show()
```

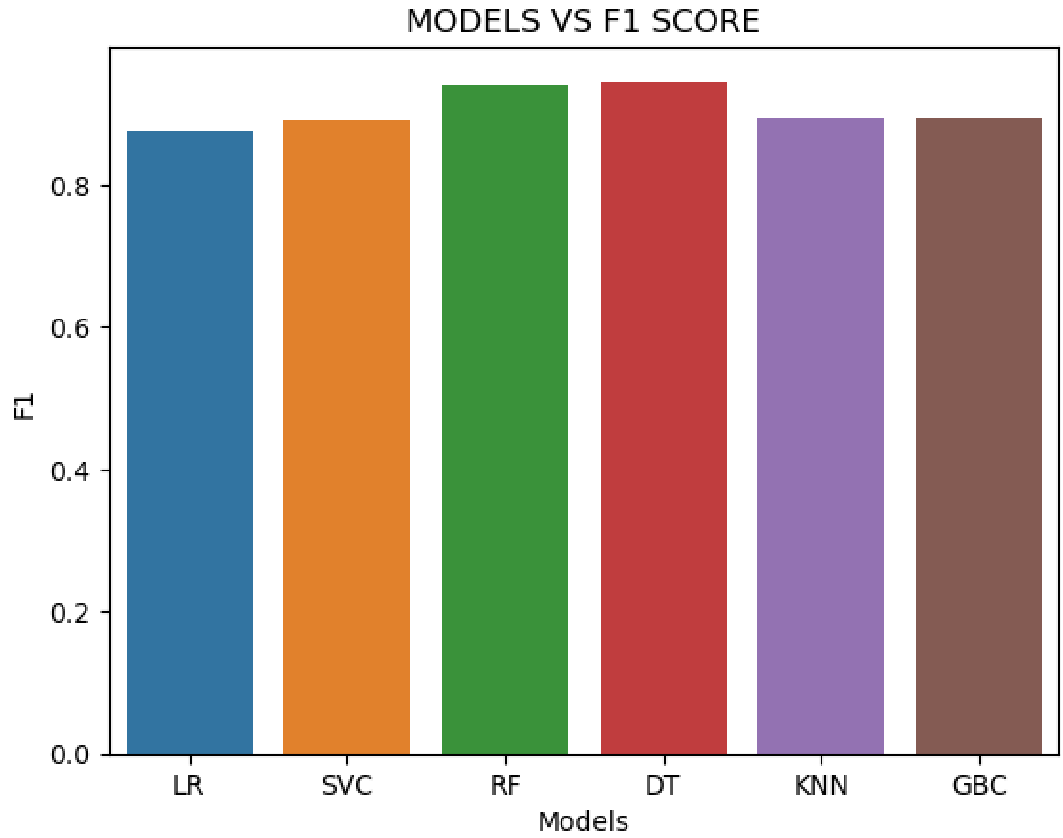
```
C:\Users\WOLF\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [90]:

```
sns.barplot(final_df['Models'],final_df['F1'])
plt.title('MODELS VS F1 SCORE')
plt.show()
```

```
C:\Users\WOLF\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



As we can see that Decision Tree Classifier is working well for this data set

therefore, we will save this model with DTC

In [91]:

```
# model saving

X_res = std.fit_transform(X_res)
```

In [92]:

```
dtc.fit(X_res,y_res)
```

Out[92]:

```
DecisionTreeClassifier()
```

Saving model using joblib library

In [93]:

```
import joblib
```

In [94]:

```
# first we need to use dump function of library
# then provide the chosen algorithm and the name we want to save the model


joblib.dump(dtc,'insurance_claim_prediction_model')
```

Out[94]:

```
['insurance_claim_prediction_model']
```

In [95]:

```
# to check wether trh emodel is saved or not

model = joblib.load('insurance_claim_prediction_model')
```

In [96]:

```
# now to verify the model is working or not we will provide it values of first row
# then cross check wether its working fine or not

df.head(1)
```

Out[96]:

	age	sex	bmi	children	smoker	region	charges	insuranceclaim
0	19	0	27.9	0	1	3	16884.924	1

In [97]:

```
model.predict([[19,0,27.9,0,1,3,16887.924]])
```

Out[97]:

```
array([1], dtype=int64)
```

insight :

As we can see our model's prediction is perfectly fine

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: