Introduction: Going to analyse the Regression data of insurance charges on how much money they got for insurance based on their Age,Sex,children, smoker,region,bmi. Algorithms used :- Linear Regression, Decision Tree, Random Forest, KNN(KNearest Neighbours), Support Vector Machine

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Analyse Dataset

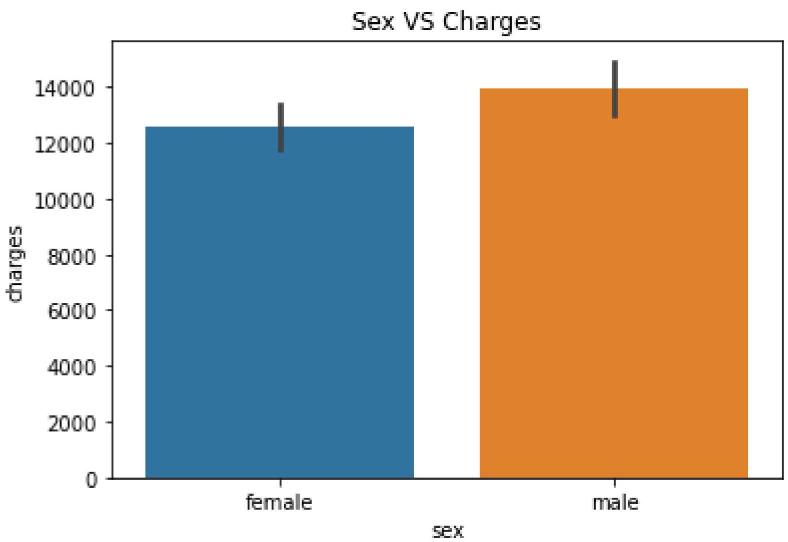
In [2]:
```
data=pd.read_csv(r'C:\Users\rahuj\Downloads\insurance.csv')
data.head(10)
```

Out[2]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
5	31	female	25.740	0	no	southeast	3756.62160
6	46	female	33.440	1	no	southeast	8240.58960
7	37	female	27.740	3	no	northwest	7281.50560
8	37	male	29.830	2	no	northeast	6406.41070
9	60	female	25.840	0	no	northwest	28923.13692

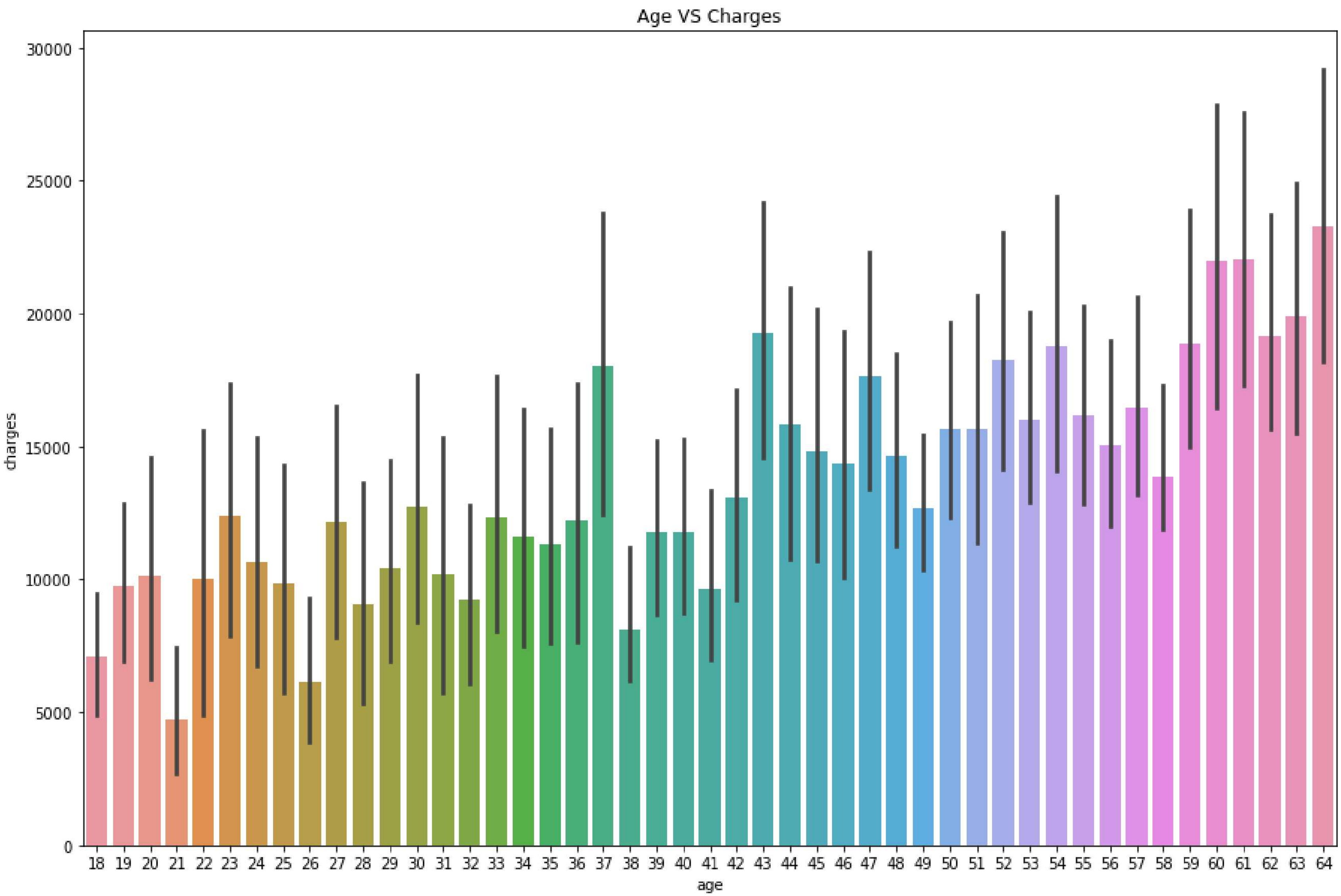
In [3]:
```
sns.barplot(x='sex',y='charges',data=data)
plt.title('Sex VS Charges')
```

Out[3]:
```
Text(0.5, 1.0, 'Sex VS Charges')
```



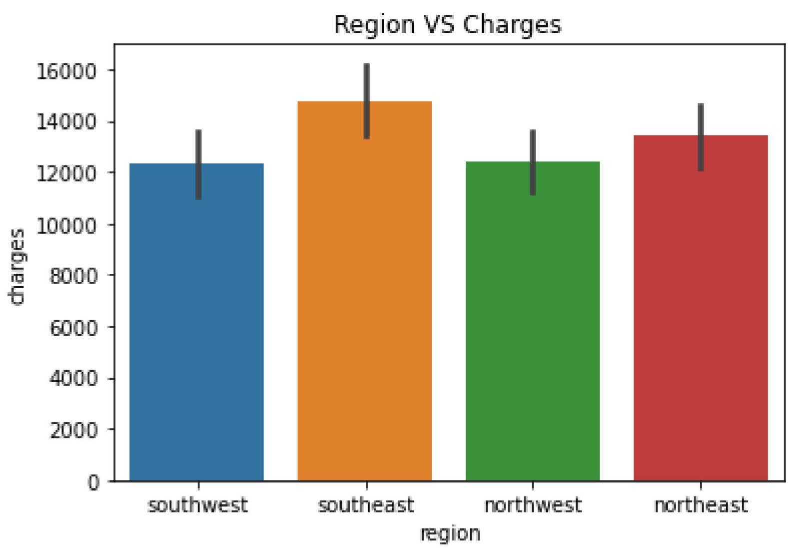
In [4]:
```
plt.figure(figsize=(15,10))
sns.barplot(x='age',y='charges',data=data)
plt.title('Age VS Charges')
```

Out[4]:
```
Text(0.5, 1.0, 'Age VS Charges')
```



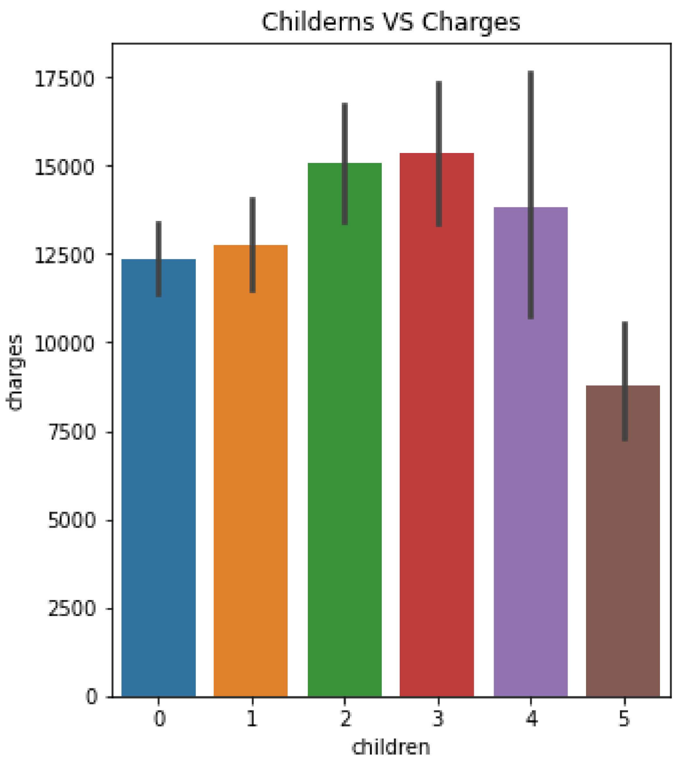
In [5]:
```
sns.barplot(x='region',y='charges',data=data)
plt.title('Region VS Charges')
```

Out[5]:
```
Text(0.5, 1.0, 'Region VS Charges')
```



In [6]:
```
plt.figure(figsize=(5,6))
sns.barplot(x='children',y='charges',data=data)
plt.title('Childerns VS Charges')
```

Out[6]:
```
Text(0.5, 1.0, 'Childerns VS Charges')
```



Cleaning the dataset

In [7]:
```
#changing the labels into int
Smoker=pd.get_dummies(data["smoker"],drop_first=True)
Region=pd.get_dummies(data["region"])
Male=pd.get_dummies(data["sex"],drop_first=True)
Male.head(5)
```

Out[7]:

	male
0	0
1	1
2	1
3	1
4	1

In [8]:
```
#adding the dummy to original dataset and droping the label
data=pd.concat([data,Smoker,Region,Male],axis=1)
data.drop(['smoker','region','northwest','southwest','sex'],axis=1,inplace=True)
data.head(5)
```

Out[8]:

	age	bmi	children	charges	yes	northeast	southeast	male
0	19	27.900	0	16884.92400	1	0	0	0
1	18	33.770	1	1725.55230	0	0	1	1
2	28	33.000	3	4449.46200	0	0	1	1
3	33	22.705	0	21984.47061	0	0	0	1
4	32	28.880	0	3866.85520	0	0	0	1

In [9]:
```
data.shape
```

Out[9]:
```
(1338, 8)
```

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
age          0
bmi          0
children     0
charges      0
yes          0
northeast    0
southeast    0
male         0
dtype: int64
```

Linear Regression

In [11]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
```

In [12]:
```
#assigning the values to input and target
inputs=data.drop('charges',axis=1)
charges=data['charges']
inputs.head(5)
```

Out[12]:

	age	bmi	children	yes	northeast	southeast	male
0	19	27.900	0	1	0	0	0
1	18	33.770	1	0	0	1	1
2	28	33.000	3	0	0	1	1
3	33	22.705	0	0	0	0	1
4	32	28.880	0	0	0	0	1

In [13]:
```
##Splitting data for training and testing
Xtrain,Xtest,ytrain,ytest=train_test_split(inputs,charges,test_size=0.2,random_state=1)
```

In [14]:
```
#model:
Lr=LinearRegression()
Lr.fit(Xtrain,ytrain)
ypre=Lr.predict(Xtest)
```

In [15]:
```
Xtest.shape
```

Out[15]:
```
(268, 7)
```

In [16]:
```
ytest
```

Out[16]:
```
559      1646.42970
1087    11353.22760
1020     8798.59300
460     10381.47870
802      2103.08000
           ...
682     40103.89000
629     42983.45850
893     44202.65360
807      2136.88225
1165     5227.98875
Name: charges, Length: 268, dtype: float64
```

In [17]:
```
ypre
```

Out[17]:
```
array([ 4.10680807e+03,  1.26216022e+04,  1.28176644e+04,  1.32852220e+04,
        8.13631091e+02,  3.18530938e+04,  1.29119133e+04,  1.23183865e+04,
        3.78833093e+03,  2.94827705e+04,  1.10251683e+04,  1.77494345e+04,
        8.68734136e+03,  8.60202156e+03,  3.12244410e+03,  1.06988261e+04,
        3.62221931e+03,  7.20712865e+03,  1.50030910e+04,  1.46815072e+04,
        1.25301807e+04,  3.29484947e+04,  8.81906482e+03,  9.24307579e+03,
        3.01579872e+03,  7.91204378e+03,  9.56754297e+03,  1.07411297e+04,
        7.93917890e+03,  4.37922060e+03,  1.43767245e+04,  6.07232443e+03,
        3.46559437e+04,  2.67405356e+04,  3.33745526e+04,  9.28856985e+03,
        3.06517591e+04,  2.69171734e+04,  1.51411213e+04,  3.36366505e+04,
        6.30729774e+03,  1.37881576e+04,  1.07360705e+04,  1.53213980e+04,
        4.45786680e+03,  1.31059946e+04,  4.32957822e+03,  2.86060915e+04,
        7.01630339e+03,  1.42818977e+04,  1.32854596e+04,  1.25849357e+04,
        1.60715226e+03,  9.14486640e+03,  2.60909184e+04,  1.00934349e+04,
        3.42268585e+04,  1.47723768e+04,  3.24770200e+03,  5.85899047e+03,
        6.54911635e+03,  1.49391446e+04,  2.69548241e+04,  3.01632095e+03,
        1.57723731e+04,  1.09572729e+04,  1.08861194e+04,  1.04883977e+04,
        1.27631628e+03,  2.52860180e+04,  3.72922110e+04,  3.31057492e+04,
        2.23484695e+03,  1.11136865e+04,  1.34246198e+04,  3.49539672e+04,
        2.94261724e+03,  3.88568243e+03,  1.06164669e+04,  1.01871941e+04,
       -4.21416649e+01,  1.38080890e+04,  1.00940488e+04,  3.40759565e+03,
        3.34843846e+04,  3.30766527e+04,  7.42192581e+03,  3.76867286e+04,
        1.28531097e+04,  1.00507119e+04,  2.98618923e+04,  3.40195896e+04,
        1.47530835e+04,  1.08016277e+04, -1.39587255e+01,  1.05583631e+04,
        9.89946282e+03,  1.49574954e+04,  1.46973952e+04,  6.09605676e+03,
        1.34093959e+04,  2.58013320e+04,  2.84092725e+04,  2.76590966e+04,
        3.53493625e+04,  2.68620387e+04,  8.95278797e+02,  9.51480063e+03,
        4.68860635e+03,  1.24547206e+04,  5.34597115e+03,  4.80982229e+03,
        8.04134683e+02,  1.85218652e+04,  3.01838144e+03,  1.92841510e+03,
        1.19605233e+04,  1.23492381e+04,  1.18573466e+04,  3.45562298e+03,
        9.16677389e+03,  1.39027726e+04,  7.73482537e+03,  6.84136222e+03,
        3.66756065e+04,  1.24178370e+04,  1.22646438e+04,  2.93275031e+04,
        3.60489174e+04,  1.18692942e+04,  2.81004159e+04, -1.48017096e+02,
        8.26253374e+03,  3.16033477e+04,  8.53176965e+03, -4.12394877e+02,
        9.25333706e+02,  4.59302446e+03,  7.36476540e+03,  1.25757716e+04,
        1.48636549e+04,  8.69993836e+03,  2.89392746e+04,  1.57041478e+04,
        1.46863782e+04,  1.08641868e+04,  1.91992065e+03,  1.03153346e+04,
        3.77936883e+03,  5.92988591e+03,  1.13881820e+04,  5.24773365e+03,
        1.43079905e+04,  1.36841513e+04,  1.26736904e+04,  7.27825735e+03,
        1.23809577e+04,  1.09265694e+04,  1.02753251e+04,  4.77665139e+03,
```

```
       5.65269586e+03,  4.03850530e+04,  1.30644443e+04,  4.55731971e+03,
       8.17428432e+03,  4.66697871e+03,  3.22006163e+04,  1.15139204e+04,
       1.12224620e+04,  6.89443404e+03,  6.69288729e+03,  6.42511753e+03,
       3.31035841e+04,  3.46496091e+04,  1.92558897e+03,  7.66939797e+03,
       5.44314446e+03,  1.55370767e+04,  1.50095902e+03,  1.14081261e+04,
       1.34392919e+04,  1.15155171e+04,  1.02886474e+04,  1.31840128e+04,
       2.15509805e+03,  2.75415984e+04,  2.36047011e+03,  1.47190524e+04,
       6.06154219e+03,  1.06022861e+04,  1.47169257e+04,  3.88379930e+04,
       2.37192608e+03,  1.24601652e+03,  4.94515865e+03,  7.82961263e+03,
       7.92745269e+03,  4.22825194e+03,  1.04490931e+04,  8.95331298e+03,
       9.38755637e+03,  1.13392489e+04,  1.06017801e+04,  9.24868421e+03,
       7.82339061e+03,  9.09212042e+02,  1.01234630e+04,  7.32663527e+03,
       6.59095468e+03,  1.19584419e+04,  5.40965715e+03,  3.28924258e+04,
       7.08497214e+03,  6.55086799e+03,  8.17254631e+03,  3.91984302e+04,
       1.19393799e+04,  2.83268661e+04,  2.87685660e+03,  3.34536934e+04,
       3.68242297e+03,  3.16069349e+04,  1.35656910e+04,  2.74986095e+03,
       1.65252785e+03,  1.52582134e+03,  5.84948141e+03,  4.70034948e+03,
       2.58285983e+04,  1.57272565e+04,  5.08125579e+03,  1.30541351e+04,
       3.89312695e+04,  4.82574274e+03,  1.27168348e+04,  1.15834073e+04,
       2.75482231e+04,  2.53122077e+03,  1.33656257e+04,  5.73630250e+03,
       1.51780880e+04,  5.75163587e+03,  1.69111789e+04,  3.89597409e+03,
       1.21961742e+04,  3.47011337e+04,  1.06648584e+04,  1.08466696e+04,
       4.87768481e+03,  1.64546250e+04,  1.41239163e+04,  5.50261691e+03,
       1.11684486e+04,  1.25051868e+04,  4.62155941e+03,  7.13662475e+03,
       2.76490653e+04,  3.22423395e+04, -7.10853110e+02,  4.02872357e+04,
       9.41086759e+03,  7.50301188e+03,  1.06724254e+04,  3.37924982e+04,
       3.56644288e+04,  3.66379972e+04,  4.68274828e+03,  6.12177546e+03])
```

In [18]:
```
# calculating the mean squared error
mse = np.mean((ytest - ypre)**2, axis = None)
```

In [19]:
```
# Calculating the root mean squared error
rmse = np.sqrt(mse)
```

In [20]:
```
# Calculating the r2 score
r2 = r2_score(ytest, ypre)
```

In [21]:
```
#result
print('MSE:',mse)
print('RMSE:',rmse)
print('R2 score:',r2*100)
```

```
MSE: 35568780.716343656
RMSE: 5963.956800341838
R2 score: 76.17321254167256
```

Decision Tree

In [22]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

In [23]:
```
#assigning variables to dependent and independend values

inputs=data.drop('charges',axis=1)
charges=data['charges']
charges
```

Out[23]:
```
0       16884.92400
1        1725.55230
2        4449.46200
3       21984.47061
4        3866.85520
           ...
1333    10600.54830
1334     2205.98080
1335     1629.83350
1336     2007.94500
1337    29141.36030
Name: charges, Length: 1338, dtype: float64
```

In [24]:
```
##Splitting data for training and testing
Xtrain,Xtest,ytrain,ytest=train_test_split(inputs,charges,test_size=0.3,random_state=1)
```

In [25]:
```
#model:
model=DecisionTreeRegressor()
model.fit(Xtrain,ytrain)
ypred=model.predict(Xtest)
```

In [26]:
```
#Calculation:
mse = np.mean((ytest - ypred)**2, axis = None)
rmse = np.sqrt(mse)
r2 = r2_score(ytest, ypred)
```

In [27]:
```
#result
print('MSE:',mse)
print('RMSE:',rmse)
print('R2 score:',r2*100)
```

```
MSE: 40030145.8649413
RMSE: 6326.93811135697
R2 score: 71.75339708408124
```

Random Forest

In [28]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

In [29]:
```
#assigning variables to dependent and independend values

inputs=data.drop('charges',axis=1)
charges=data['charges']
charges
```

Out[29]:
```
0       16884.92400
1        1725.55230
2        4449.46200
3       21984.47061
4        3866.85520
            ...
1333    10600.54830
1334     2205.98080
1335     1629.83350
1336     2007.94500
1337    29141.36030
Name: charges, Length: 1338, dtype: float64
```

In [30]:
```
##Splitting data for training and testing
Xtrain,Xtest,ytrain,ytest=train_test_split(inputs,charges,test_size=0.3,random_state=1)
```

In [31]:
```
#model:
model=RandomForestRegressor()
model.fit(Xtrain,ytrain)
ypred=model.predict(Xtest)
```

In [32]:
```
#Calculation:
mse = np.mean((ytest - ypred)**2, axis = None)
rmse = np.sqrt(mse)
r2 = r2_score(ytest, ypred)
```

In [33]:
```
#result
print('MSE:',mse)
print('RMSE:',rmse)
print('R2 score:',r2*100)
```

```
MSE: 23615533.288140927
RMSE: 4859.581595995784
R2 score: 83.33609391061468
```

KNN

In [34]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [35]:
```
#assigning variables to dependent and independend values

inputs=data.drop('charges',axis=1)
charges=data['charges']
charges
```

Out[35]:
```
0       16884.92400
1        1725.55230
2        4449.46200
3       21984.47061
4        3866.85520
           ...
1333    10600.54830
1334     2205.98080
1335     1629.83350
1336     2007.94500
1337    29141.36030
Name: charges, Length: 1338, dtype: float64
```

In [36]:
```
##Splitting data for training and testing
Xtrain,Xtest,ytrain,ytest=train_test_split(inputs,charges,test_size=0.3,random_state=1)
```

In [37]:
```
#Standardising the data to convert the mean into 0 and SD into 1
Sr=StandardScaler()
S_Xtrain=Sr.fit_transform(Xtrain)
S_Xtest=Sr.fit_transform(Xtest)
S_Xtrain
```

Out[37]:
```
array([[ 0.79715222, -0.70211414, -0.90400228, ..., -0.56254395,
        -0.60038747,  0.97676557],
       [-1.27108519, -0.70375759, -0.08567913, ..., -0.56254395,
        -0.60038747, -1.02378711],
       [-0.98581107, -0.73333977, -0.90400228, ...,  1.77763888,
        -0.60038747,  0.97676557],
       ...,
       [ 0.86847075,  0.70303946,  0.73264401, ...,  1.77763888,
        -0.60038747, -1.02378711],
       [ 0.0839669 , -1.39072157,  0.73264401, ..., -0.56254395,
         1.66559105, -1.02378711],
       [ 1.29638193, -0.4506656 , -0.08567913, ..., -0.56254395,
         1.66559105,  0.97676557]])
```

In [38]:
```
# knn model and prediction
knn = KNeighborsRegressor(n_neighbors=7)
knn.fit(S_Xtrain,ytrain)
ypre=knn.predict(S_Xtest)
```

In [39]:
```
#Calculation:
mse = np.mean((ytest - ypre)**2, axis = None)
rmse = np.sqrt(mse)
r2 = r2_score(ytest, ypre)
```

In [40]:
```
#result
print('MSE:',mse)
print('RMSE:',rmse)
print('R2 score:',r2*100)
```

```
MSE: 29599020.6668095
RMSE: 5440.498200239524
R2 score: 79.11394611710175
```

Support Vector Machine

In [41]:
```
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
```

In [42]:
```
#assigning the values to input and target
X=data.drop('charges',axis=1)
y=data['charges']
X.head(5)
```

Out[42]:

	age	bmi	children	yes	northeast	southeast	male
0	19	27.900	0	1	0	0	0
1	18	33.770	1	0	0	1	1
2	28	33.000	3	0	0	1	1
3	33	22.705	0	0	0	0	1
4	32	28.880	0	0	0	0	1

In [43]:
```
##Splitting data for training and testing
Xtrain,Xtest,y_train,y_test=train_test_split(inputs,charges,test_size=0.2,random_state=1)
```

In [44]:
```
#Model
model=SVR()
model.fit(Xtrain,y_train)
ypred=model.predict(Xtest)
```

In [45]:
```
#Calculation:
mse = np.mean((y_test - ypred)**2, axis = None)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, ypred)
```

In [46]:
```
#result
print('MSE:',mse)
print('RMSE:',rmse)
print('R2 score:',r2*100)
```

```
MSE: 166558631.96321929
RMSE: 12905.759643012854
R2 score: -11.574168223143054
```

Inference:

Among all the algorithms we got least rmse on Random Forest with score on 83%

Second most algorithm performed well is KNN with score 79%