```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

Introduction Many factors that affect how much you pay for health insurance are not within your control. Nonetheless, it's good to have an understanding of what they are. Here are some factors that affect how much health insurance premiums cost

age: age of primary beneficiary

sex: insurance contractor gender, female, male

bmi: Body mass index, providing an understanding of body, weights that are relatively high or low relative to height, objective index of body weight (kg / m ^ 2) using the ratio of height to weight, ideally 18.5 to 24.9

children: Number of children covered by health insurance / Number of dependents

smoker: Smoking

region: the beneficiary's residential area in the US, northeast, southeast, southwest, northwest

charges: insurance premium costs

```
#import all the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
#read the data
df = pd.read_csv('/content/Medical Price Dataset.csv')
```

```
df.head()
```

	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

```
df.shape
```

```
(1338, 7)
```

```
df.describe()
```

	age	bmi	children	charges

```
df.dtypes
```

```
age           int64
sex          object
bmi         float64
children      int64
smoker       object
region       object
charges     float64
dtype: object
```

75%	51.000000	34.693750	2.000000	16639.912515

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

```
#checking for null values
df.isnull().sum()
```

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

```
##Converting objects labels into categorical
df[['sex', 'smoker', 'region']] = df[['sex', 'smoker', 'region']].astype('category')
df.dtypes
```

```
age            int64
sex         category
bmi          float64
children       int64
smoker      category
region      category
charges      float64
dtype: object
```

```
##Converting category labels into numerical using LabelEncoder
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
label.fit(df.sex.drop_duplicates())
df.sex = label.transform(df.sex)
label.fit(df.smoker.drop_duplicates())
df.smoker = label.transform(df.smoker)
label.fit(df.region.drop_duplicates())
df.region = label.transform(df.region)
df.dtypes
```

```
age           int64
sex           int64
bmi         float64
children      int64
smoker        int64
region        int64
charges     float64
dtype: object
```

1. Linear Regression Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

sklearn.linear_model.LinearRegression(, fit_intercept=True, normalize='deprecated', copy_X=True, n_jobs=None, positive=False)*

```
#importing the required libraries and splitting the data
from sklearn.model_selection import train_test_split as holdout
from sklearn.linear_model import LinearRegression
from sklearn import metrics
x = df.drop(['charges'], axis = 1)
y = df['charges']

x_train, x_test, y_train, y_test = holdout(x, y, test_size=0.2, random_state=0)
Lin_reg = LinearRegression()
Lin_reg.fit(x_train, y_train)

print(Lin_reg.intercept_)
print(Lin_reg.coef_)
print(Lin_reg.score(x_test, y_test))
```

```
-11661.98390882441
[  253.99185244   -24.32455098   328.40261701   443.72929547
  23568.87948381  -288.50857254]
0.7998747145449959
```

```
#polynomial regression
from sklearn.preprocessing import PolynomialFeatures
x = df.drop(['charges', 'sex', 'region'], axis = 1)
y = df.charges
pol = PolynomialFeatures (degree = 2)
x_pol = pol.fit_transform(x)
x_train, x_test, y_train, y_test = holdout(x_pol, y, test_size=0.2, random_state=0)
Pol_reg = LinearRegression()
Pol_reg.fit(x_train, y_train)
y_train_pred = Pol_reg.predict(x_train)
y_test_pred = Pol_reg.predict(x_test)
print(Pol_reg.intercept_)
print(Pol_reg.coef_)
print(Pol_reg.score(x_test, y_test))
```

```
-5325.8817052531285
[ 0.00000000e+00 -4.01606591e+01  5.23702019e+02  8.52025026e+02
 -9.52698471e+03  3.04430186e+00  1.84508369e+00  6.01720286e+00
  4.20849790e+00 -9.38983382e+00  3.81612289e+00  1.40840670e+03
 -1.45982790e+02 -4.46151855e+02 -9.52698471e+03]
0.8812595703345227
```

```
##Predicting the charges
y_test_pred = Pol_reg.predict(x_test)
##Comparing the actual output values with the predicted values
#df = pd.DataFrame({'Actual': y_test, 'Predicted': y_test_pred})
#df
```

```
#model evaluation for LR
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt

RMSE = float(format(np.sqrt(mean_squared_error(y_test, y_test_pred)),'.3f'))
MSE = mean_squared_error(y_test, y_test_pred)
MAE = mean_absolute_error(y_test, y_test_pred)
r2 = r2_score(y_test, y_test_pred)

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2)
```

```
RMSE = 4346.856
MSE = 18895160.098780397
MAE = 2824.495045477652
R2 = 0.8812595703345227
```

```
# splitting data into train and test datatest

x = df.drop('charges', axis = 1)
y = df['charges']

#splitting into train and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
x_train.shape, x_test.shape

#fitting the model
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=0, max_depth=2)
regressor.fit(x_train,y_train)
y_pred = regressor.predict(x_test)
```

```
#model evaluation for DTR
from sklearn import metrics
print(" R Squared error:", metrics.r2_score(y_test,y_pred))
print("Mean absolute error:", metrics.mean_absolute_error(y_test,y_pred))
print("Mean squared error:", metrics.mean_squared_error(y_test,y_pred))
```

```
 R Squared error: 0.8553666902985595
Mean absolute error: 3248.4796599366896
Mean squared error: 23015493.123314563
```

```
#splitting into train-test
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.2, random_state=2)

print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)

#scaling the data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
x_train_scaled = scaler.fit_transform(X_train)
X_train = pd.DataFrame(x_train_scaled)
x_test_scaled = scaler.fit_transform(X_test)
X_test = pd.DataFrame(x_test_scaled)
```

```
(1070, 6) (268, 6) (1070,) (268,)
```

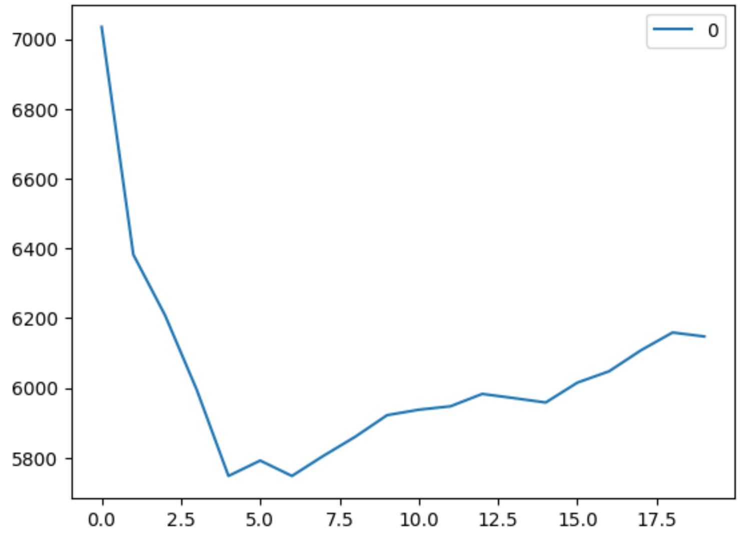
```
#import required packages
from sklearn.neighbors import KNeighborsRegressor
from sklearn import neighbors
from sklearn.metrics import mean_squared_error
from math import sqrt
import matplotlib.pyplot as plt
%matplotlib inline

rmse_val = [] #to store rmse values for different
for K in range(20):
    K = K+1
    model = neighbors.KNeighborsRegressor(n_neighbors = K)
    model.fit(X_train, Y_train)
    #fit the model
    pred=model.predict(X_test)
    #make prediction on test set
    error = sqrt(mean_squared_error(Y_test,pred))
    #calculate rmse
    rmse_val.append(error)
#store rmse values
print('RMSE value for k= ' , K , 'is:', error)
```

```
RMSE value for k=  20 is: 6147.382834636041
```

```
#plotting the rmse values against k values
curve = pd.DataFrame(rmse_val) #elbow curve
curve.plot()
```

```
<Axes: >
```



```
#fitting the KNN regressor
knn_model = KNeighborsRegressor(n_neighbors=7).fit(X_train, Y_train)
```

```
# Score
score_knn = knn_model.score(X_test, Y_test)
score_knn
```

```
0.7800406983410726
```