Lab -6

PRML
AY 2020-21 Trimester - III

March 10, 2021

Linear Regression **J Jahnavi(B19CSE109)**

Linear regression is a **supervised learning** algorithm used when the target / dependent variable **continues** the real number. It establishes a relationship between the dependent variable ‘y’ and one or more independent variables using the best fit line.

Exploratory data analysis:

```
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
```

```
df.describe()
```

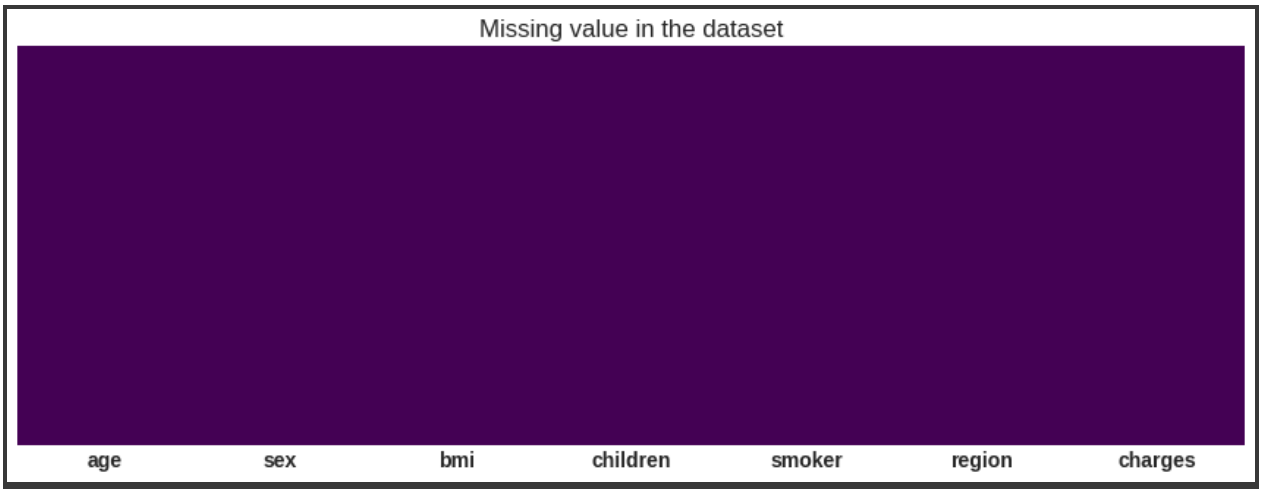
	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000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

Checking for missing values:

The number of missing values for each feature/column is as follows:

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

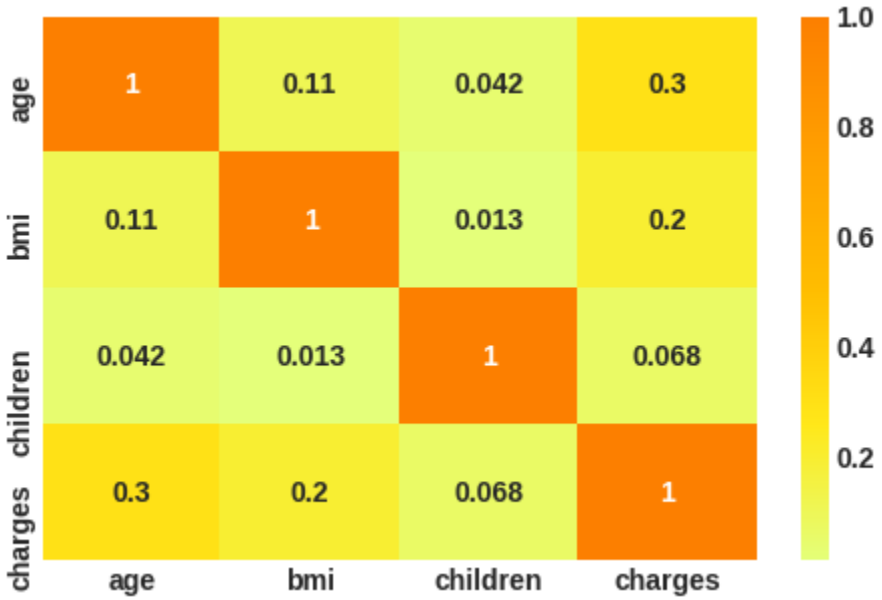
The plot for missing data:



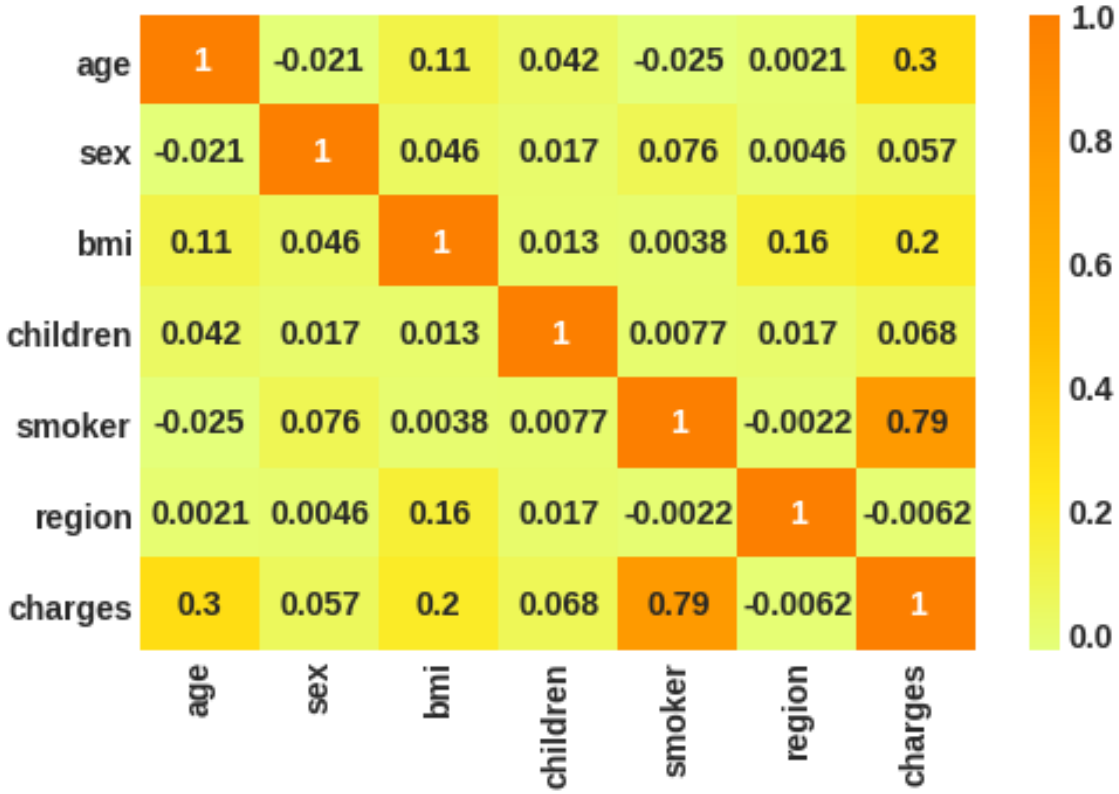
The above information implies that there are no missing values in the dataset.

Correlation Plot:

Before data encoding -

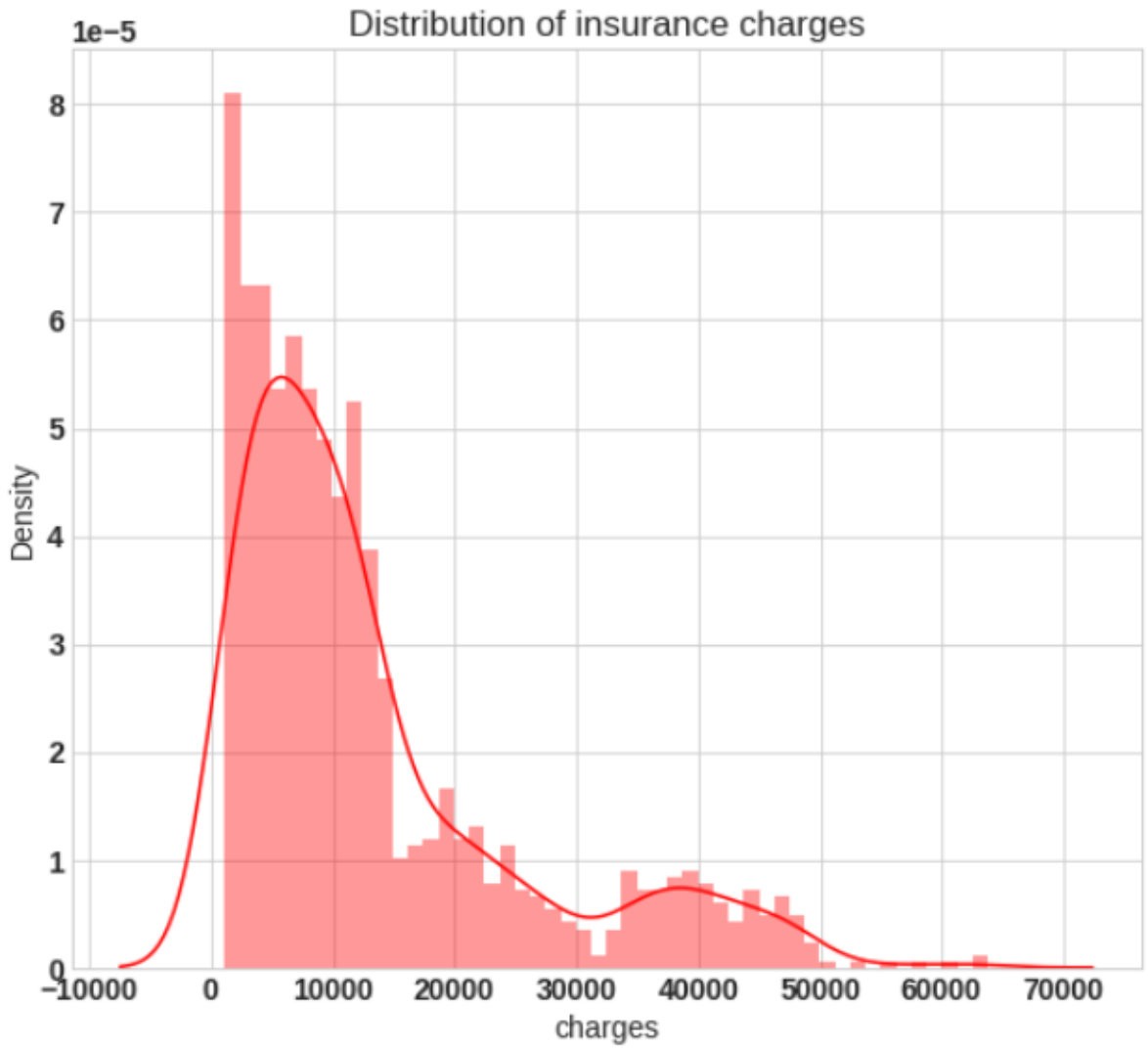


After data encoding (using label encoding)-



These plots show that there is **little or no correlation between any pairs of variables**.

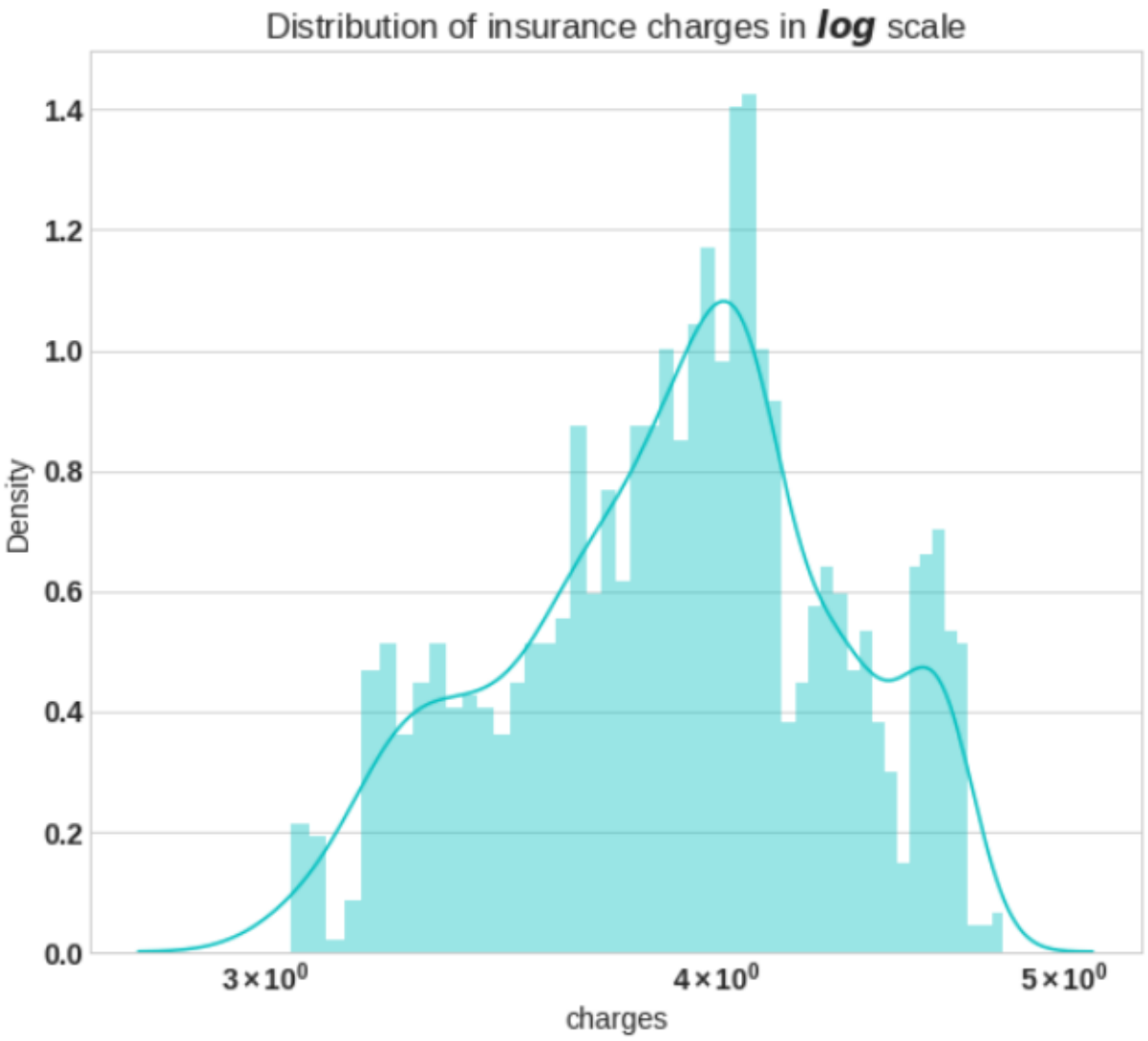
Distribution of insurance charges (i.e. target variable):



This plot is right-skewed therefore to convert this distribution into normal, we apply natural log to the distribution.

The plot is as follows:

Distribution of insurance charges in log scale:



Converting categorical data into numbers (using One hot Encoding):

One hot encoding is a representation of categorical variables as binary vectors. It allows the representation of categorical data to be more expressive. This first requires the categorical values to be mapped to integer values(i.e. label encoding). Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked as 1.

```
cat_columns = ['sex','children', 'smoker', 'region']
df_enc = pd.get_dummies(data = df, prefix = 'oneHotEnc', prefix_sep='_',
                columns = cat_columns,
                drop_first =True,
               dtype='int8')
```

The final data frame:

```
df_enc.head()
```

	age	bmi	charges	oneHotEnc_male	oneHotEnc_1	oneHotEnc_2	oneHotEnc_3	oneHotEnc_4	oneHotEnc_5	oneHotEnc_yes	oneHotEnc_northwest	oneHotEnc_southeast	oneHotEnc_southwest
0	19	27.900	9.734176	0	0	0	0	0	0	1	0	0	1
1	18	33.770	7.453302	1	1	0	0	0	0	0	0	1	0
2	28	33.000	8.400538	1	0	0	1	0	0	0	0	1	0
3	33	22.705	9.998092	1	0	0	0	0	0	0	1	0	0
4	32	28.880	8.260197	1	0	0	0	0	0	0	1	0	0

Split the data into training and testing sets with a ratio of 0.3:

```
#Split the data into training and testing sets with ratio 0.3
from sklearn.model_selection import train_test_split
X = df_enc.drop('charges',axis=1) # independent variable
y = df_enc['charges'] # dependent variable

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=42)
```

Built a model using linear regression equation $\theta=(X^TX)^{-1}X^Ty$ as follows:

- Firstly, added a feature X_0 =1 to the original dataset as follows:

```
# Step 1: add x0 =1 to dataset
X_train_0 = np.c_[np.ones((X_train.shape[0],1)),X_train]
X_test_0 = np.c_[np.ones((X_test.shape[0],1)),X_test]
```

Here, the function numpy.c_() is used to concatenate X_0 and X (test and train sets) along the second axis.

- Built the model as follows:

```
# Step2: build model
theta = np.matmul(np.linalg.inv( np.matmul(X_train_0.T,X_train_0) ), np.matmul(X_train_0.T,y_train))
```

Here, the functions used are as follows:

- numpy.matmul(): to compute matrix product of two arrays
- numpy.linalg.inv(): to compute the (multiplicative) inverse of a matrix

Built a linear regression model using the sklearn library:

```
# Scikit Learn module
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train,y_train)
```

Parameters of both the models:

	Parameter	Columns	theta	sklearn_theta
0	theta_0	intersect:x_0=1	7.017547	7.017547
1	theta_1	age	0.034570	0.034570
2	theta_2	bmi	0.013721	0.013721
3	theta_3	oneHotEnc_male	-0.080313	-0.080313
4	theta_4	oneHotEnc_1	0.146353	0.146353
5	theta_5	oneHotEnc_2	0.263423	0.263423
6	theta_6	oneHotEnc_3	0.208758	0.208758
7	theta_7	oneHotEnc_4	0.554547	0.554547
8	theta_8	oneHotEnc_5	0.414970	0.414970
9	theta_9	oneHotEnc_yes	1.557606	1.557606
10	theta_10	oneHotEnc_northwest	-0.071081	-0.071081
11	theta_11	oneHotEnc_southeast	-0.164031	-0.164031
12	theta_12	oneHotEnc_southwest	-0.127162	-0.127162

The above table shows that the parameters for both models perfectly match which is the ideal case.

Predictions from both the models:

- Prediction for the model using linear regression equation $\theta=(X^TX)^{-1}X^Ty$:

```
#prediction
y_pred =  np.matmul(X_test_0,theta)
```

```
y_pred
```

```
array([ 9.18206967,  8.60291208, 11.0854699 ,  9.0184013 ,  9.51857961,
        8.7204616 ,  7.91223788,  9.59876054,  8.22275801,  9.16252167,
        9.94834565,  8.96669485,  8.40842315, 10.71090052, 10.96650508,
       10.75334603,  9.39651234, 10.68489279,  8.97498755, 10.35949333,
        8.41723945,  9.00733448,  7.88403663,  8.3242728 ,  9.26958411,
        9.33275804,  9.44622321,  8.50575164,  9.27076325,  7.80903001,
        9.22978047,  9.35365573,  8.05795634,  8.46607425,  8.26689341,
        9.03907669,  8.12495822,  8.9335299 , 10.77917691, 10.20468447,
        8.35462132,  8.20321696,  9.46390006,  9.25514652,  8.66798492,
        9.37341031,  8.39731487,  8.2968297 , 10.51964723,  8.63506871,
        9.52953104,  7.94481769,  9.04397011,  7.91796988,  9.18156658,
        9.36976563,  8.32368893, 10.17607341,  9.34963984,  9.30600891,
        9.50465362,  8.7457138 ,  9.62295983,  9.00467019,  9.23885034,
        8.41665092, 10.00083983,  9.14186097,  8.2660129 ,  8.11261204,
        8.90563526,  9.31440475,  8.97022388,  8.82861692,  8.96288454,
        8.50859025,  8.52650101,  9.23435595,  8.5377129 ,  9.00744588,
        7.92194181, 11.00449039,  8.66895451, 10.1368555 ,  9.95917752,
       10.66003726,  8.45225358,  9.21723709,  9.11745231,  9.35325802,
        9.9470451 , 10.93907546, 10.70478096,  8.65166376, 10.10852029,
```

- Prediction for the linear regression model using the sklearn library:

```
# sklearn regression module
y_pred_sk = lin_reg.predict(X_test)
```

```
y_pred_sk
```

```
array([ 9.18206967,  8.60291208, 11.0854699 ,  9.0184013 ,  9.51857961,
        8.7204616 ,  7.91223788,  9.59876054,  8.22275801,  9.16252167,
        9.94834565,  8.96669485,  8.40842315, 10.71090052, 10.96650508,
       10.75334603,  9.39651234, 10.68489279,  8.97498755, 10.35949333,
        8.41723945,  9.00733448,  7.88403663,  8.3242728 ,  9.26958411,
        9.33275804,  9.44622321,  8.50575164,  9.27076325,  7.80903001,
        9.22978047,  9.35365573,  8.05795634,  8.46607425,  8.26689341,
        9.03907669,  8.12495822,  8.9335299 , 10.77917691, 10.20468447,
        8.35462132,  8.20321696,  9.46390006,  9.25514652,  8.66798492,
        9.37341031,  8.39731487,  8.2968297 , 10.51964723,  8.63506871,
        9.52953104,  7.94481769,  9.04397011,  7.91796988,  9.18156658,
        9.36976563,  8.32368893, 10.17607341,  9.34963984,  9.30600891,
        9.50465362,  8.7457138 ,  9.62295983,  9.00467019,  9.23885034,
        8.41665092, 10.00083983,  9.14186097,  8.2660129 ,  8.11261204,
        8.90563526,  9.31440475,  8.97022388,  8.82861692,  8.96288454,
        8.50859025,  8.52650101,  9.23435595,  8.5377129 ,  9.00744588,
        7.92194181, 11.00449039,  8.66895451, 10.1368555 ,  9.95917752,
       10.66003726,  8.45225358,  9.21723709,  9.11745231,  9.35325802,
        9.9470451 , 10.93907546, 10.70478096,  8.65166376, 10.10852029,
```

Evaluation using the MSE of both models:

Here, both the models have **low MSE which implies a higher accuracy of prediction**.

- Calculations for the model using linear regression equation $\theta=(X^TX)^{-1}X^Ty$:

```
J_mse = np.sum((y_pred - y_test)**2)/ X_test_0.shape[0]
```

```
print('The Mean Square Error(MSE) or J(theta) is: ',J_mse)
```

```
The Mean Square Error(MSE) or J(theta) is:  0.1805122975889298
```

- For the linear regression model using the sklearn library:

```
from sklearn.metrics import mean_squared_error
J_mse_sk = mean_squared_error(y_pred_sk, y_test)
```

```
print('The Mean Square Error(MSE) or J(theta) is: ',J_mse_sk)
```

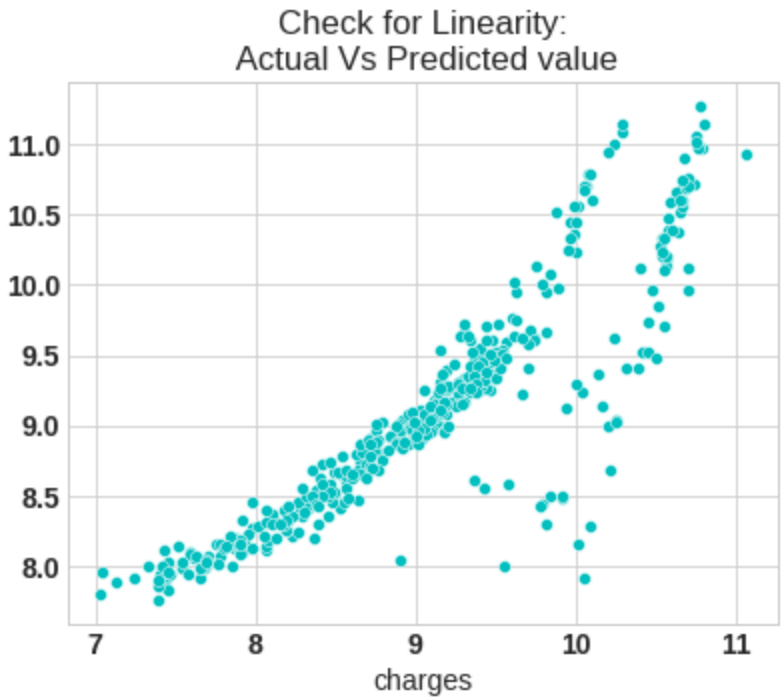
```
The Mean Square Error(MSE) or J(theta) is:  0.1805122975889313
```

Model Validation

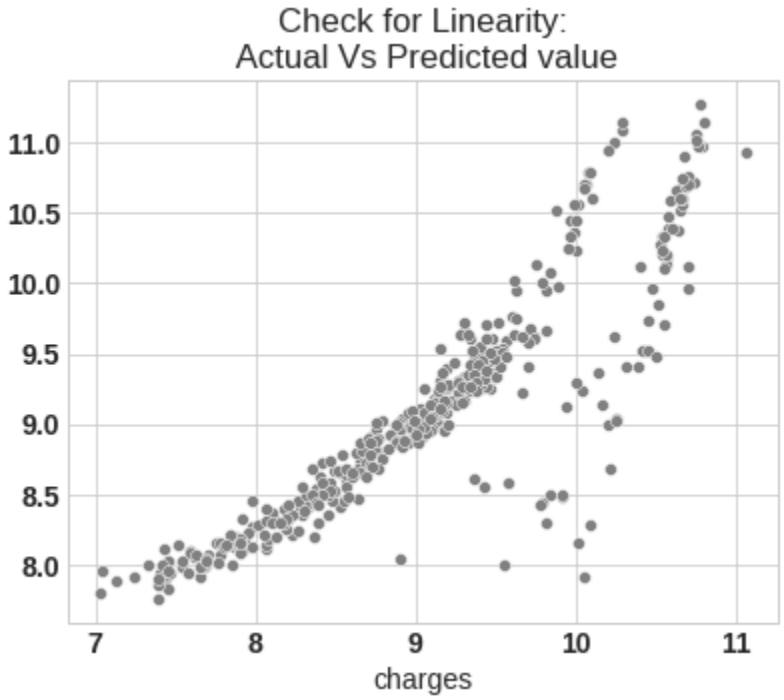
Check for Linearity:

In linear regression, the relationship between the dependent and independent variable is expected to be linear. This can be checked by a scatter plot between the actual values and the predicted value.

- For the model using linear regression equation $\theta=(X^TX)^{-1}X^Ty$:

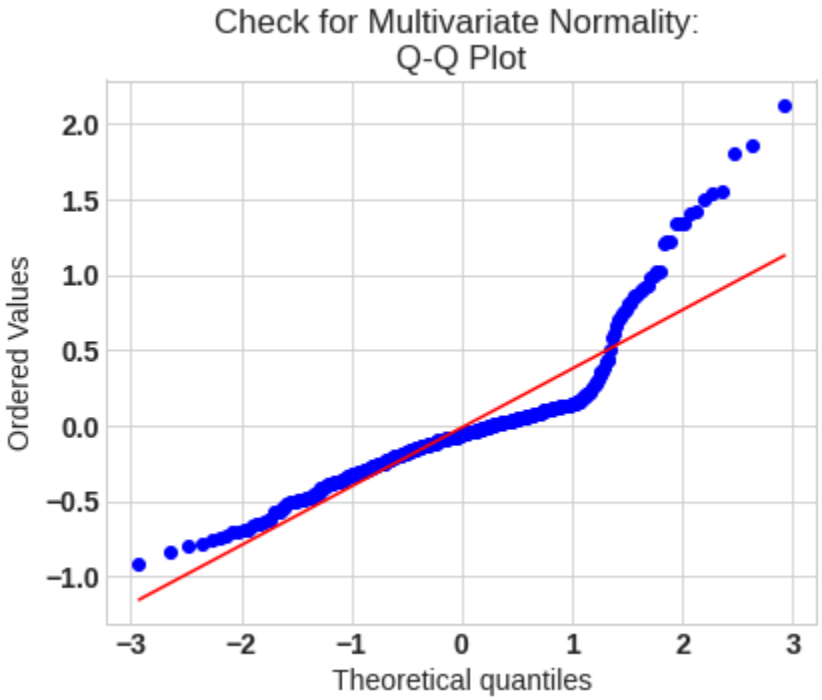


- For linear regression model using the sklearn library:

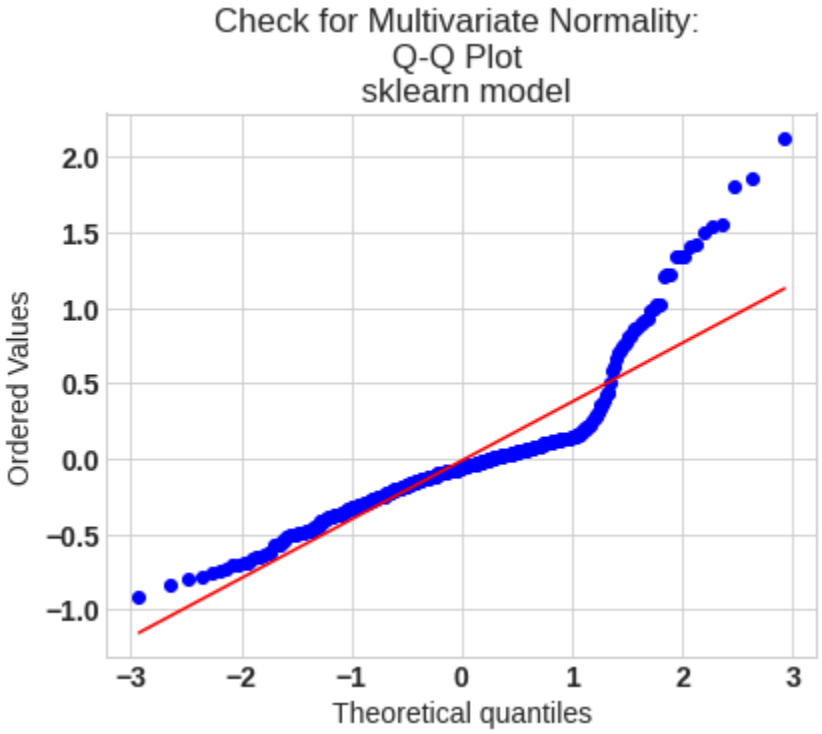


The linear regression requires all variables to be multivariate normal. This assumption can be best checked with a Quantile-Quantile plot as follows:

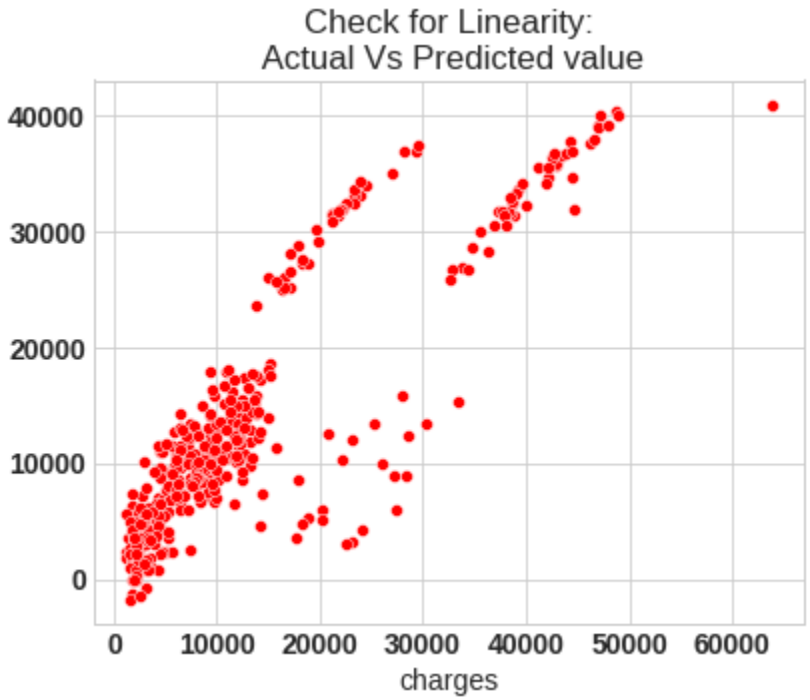
- For the model using linear regression equation $\theta=(X^TX)^{-1}X^Ty$:



- For linear regression model using the sklearn library:



Model Validation for distribution without applying natural log:
Here, the distribution plot is right-skewed and not normally distributed.



MSE: A comparatively higher MSE implies a lower accuracy of prediction.

```
print('For distribution without applying normal log -the Mean Square Error(MSE) or J(theta) is: ',J_mseN)
```

```
For distribution without applying normal log -the Mean Square Error(MSE) or J(theta) is:  34083811.64751998
```

Parameters:

	Parameter_N	Columns_N	theta_N	sklearn_theta_N
0	theta_0	intersect:x_0=1	-12390.771228	-12390.771228
1	theta_1	age	262.646552	262.646552
2	theta_2	bmi	347.697254	347.697254
3	theta_3	oneHotEnc_male	85.709964	85.709964
4	theta_4	oneHotEnc_1	351.904470	351.904470
5	theta_5	oneHotEnc_2	1506.874887	1506.874887
6	theta_6	oneHotEnc_3	492.911285	492.911285
7	theta_7	oneHotEnc_4	4020.808684	4020.808684
8	theta_8	oneHotEnc_5	806.603012	806.603012
9	theta_9	oneHotEnc_yes	23654.470990	23654.470990
10	theta_10	oneHotEnc_northwest	-545.740837	-545.740837
11	theta_11	oneHotEnc_southeast	-1005.697880	-1005.697880
12	theta_12	oneHotEnc_southwest	-958.744351	-958.744351