We will build a Linear regression model for Medical cost dataset. The dataset consists of age, sex, BMI(body mass index), children, smoker and region feature, which are independent and charge as a dependent feature. We will predict individual medical costs billed by health insurance.

Importing required libraries

In [1]:

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

Import and read data

In [2]:

```
file_path = '/Users/anant/Downloads/insurance.csv'
df = pd.read_csv(file_path)
df
```

Out[2]:

	age	sex	bmi	children	smoker	region	expenses
0	19	female	27.9	0	yes	southwest	16884.92
1	18	male	33.8	1	no	southeast	1725.55
2	28	male	33.0	3	no	southeast	4449.46
3	33	male	22.7	0	no	northwest	21984.47
4	32	male	28.9	0	no	northwest	3866.86
...	...	...	...	...	...	...	...
1333	50	male	31.0	3	no	northwest	10600.55
1334	18	female	31.9	0	no	northeast	2205.98
1335	18	female	36.9	0	no	southeast	1629.83
1336	21	female	25.8	0	no	southwest	2007.95
1337	61	female	29.1	0	yes	northwest	29141.36

1338 rows × 7 columns

Our dataset has 1338 rows and 7 columns

In [3]:

```
df.describe()
```

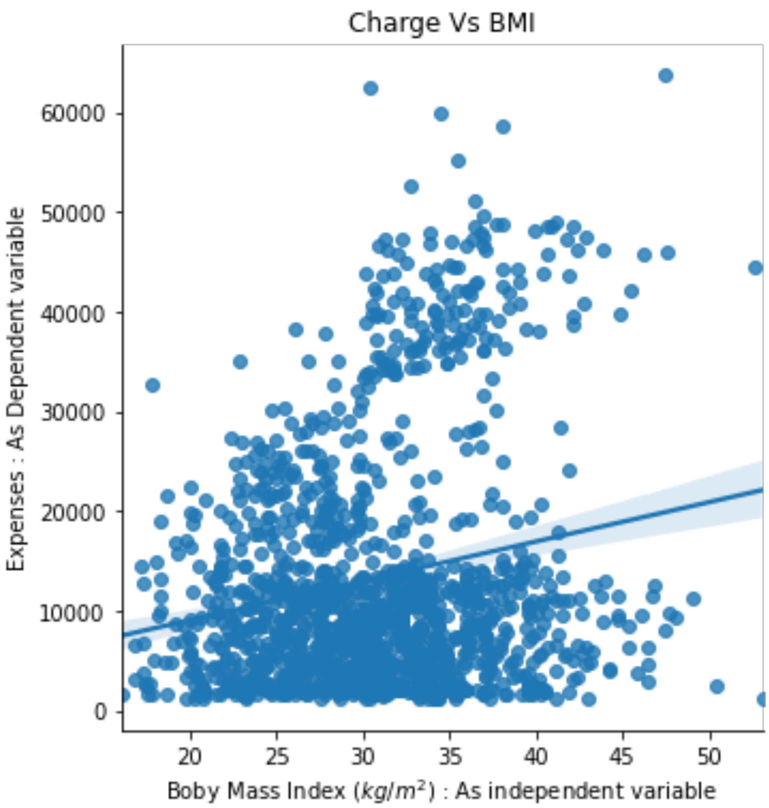
Out[3]:

	age	bmi	children	expenses
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.665471	1.094918	13270.422414
std	14.049960	6.098382	1.205493	12110.011240
min	18.000000	16.000000	0.000000	1121.870000
25%	27.000000	26.300000	0.000000	4740.287500
50%	39.000000	30.400000	1.000000	9382.030000
75%	51.000000	34.700000	2.000000	16639.915000
max	64.000000	53.100000	5.000000	63770.430000

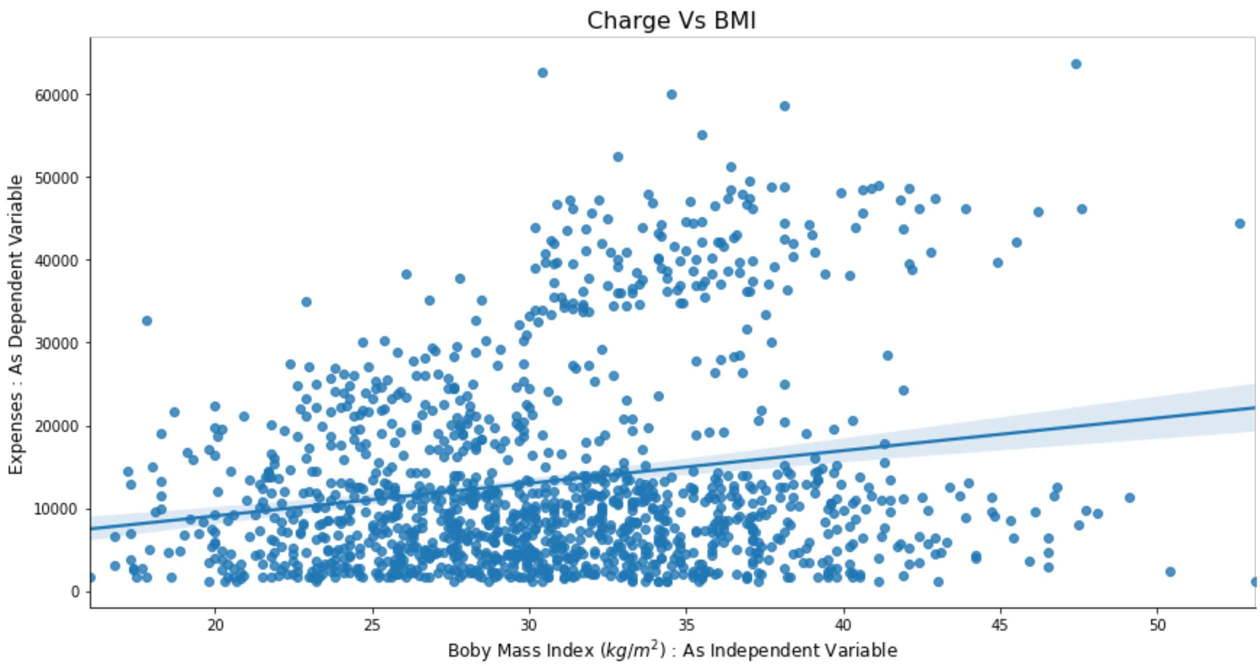
Data Analysis

In [4]:

```
sns.lmplot(x = 'bmi', y = 'expenses', data=df)

plt.xlabel('Boby Mass Index $(kg/m^2)$ : As independent variable')
plt.ylabel('Expenses : As Dependent variable')
plt.title('Charge Vs BMI')
plt.show()
```



In [5]:
```
sns.lmplot(x = 'bmi', y = 'expenses', data=df, aspect = 2, height = 6)

plt.xlabel('Boby Mass Index $(kg/m^2)$ : As Independent Variable', fontsiz
plt.ylabel('Expenses : As Dependent Variable', fontsize = 12)
plt.title('Charge Vs BMI', fontsize = 16)
plt.show()
```



In [ ]:
```

```

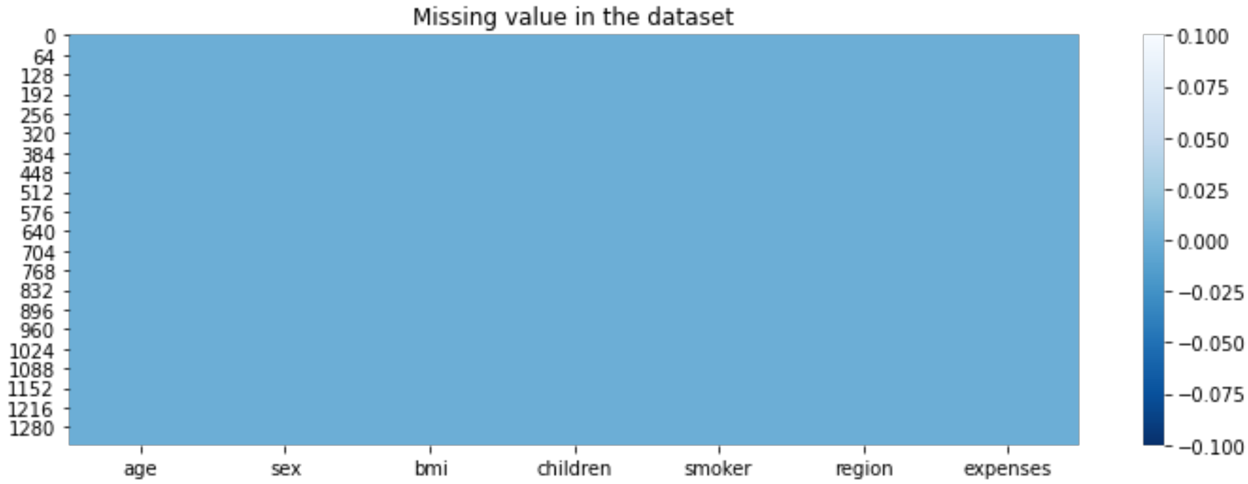
Check for missing value

In [6]:
```
if True in df.notnull():
    print("yes")
else:
    print("No")
```

No

Another Method

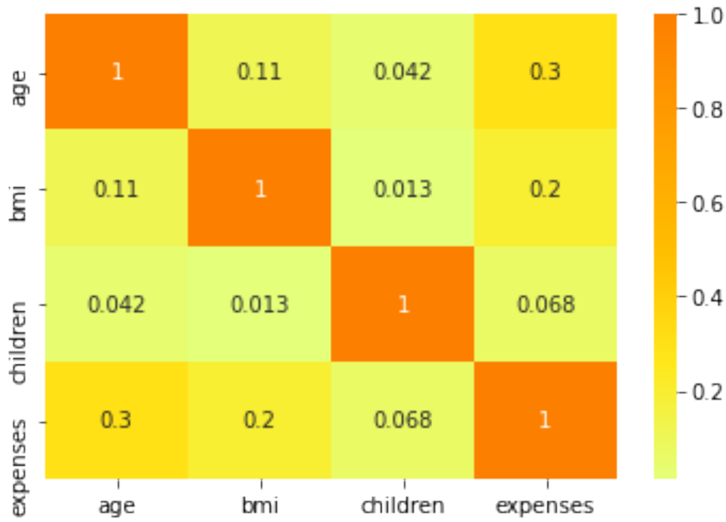
In [7]:
```
plt.figure(figsize=(12,4))

sns.heatmap(df.isnull(), cmap = 'Blues_r')

plt.title('Missing value in the dataset');
```



No NULL value

Finding correlation in data

In [8]:
```
sns.heatmap(df.corr(), annot = True, cmap = 'Wistia')
plt.show()
```



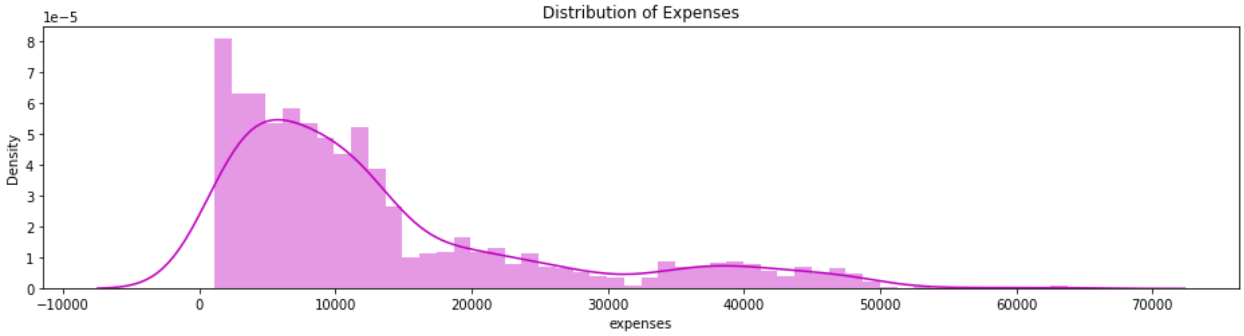
There is as such no correlation

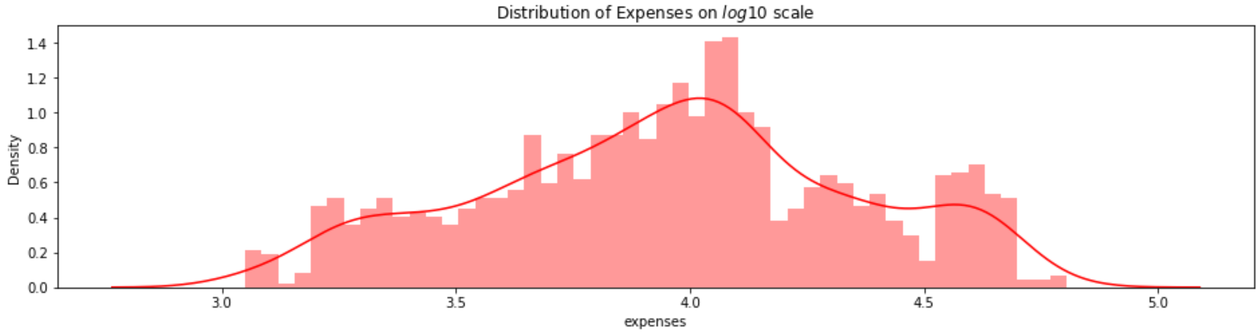
In [ ]:
```

```

In [9]:

```
plt.figure(figsize = (16, 9))

plt.subplot(2,1, 1)
sns.distplot(df['expenses'], color = 'm', bins = 50)
plt.title('Distribution of Expenses')

plt.subplot(2, 1, 2)
sns.distplot(np.log10(df['expenses']), color = 'r', bins = 50)
plt.title('Distribution of Expenses on $log10$ scale')

plt.subplots_adjust(hspace = 0.5)
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packag
es/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to
use either `displot` (a figure-level function with similar flexibility) or
`histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packag
es/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to
use either `displot` (a figure-level function with similar flexibility) or
`histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





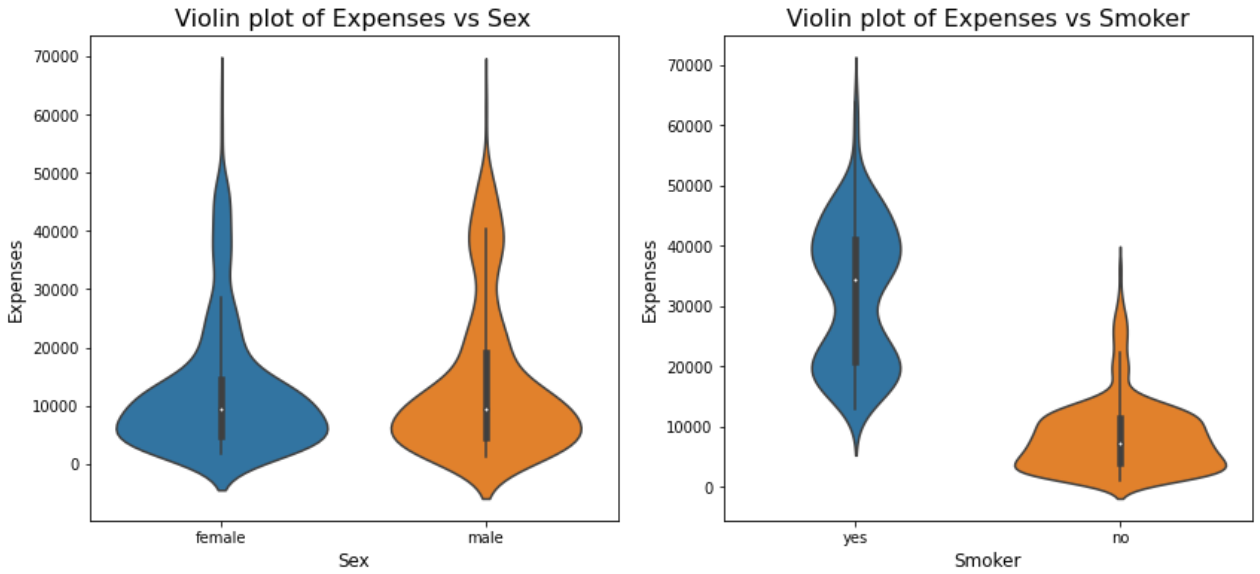
If we look at the left plot the charges varies from 1120 to 63500, the plot is right skewed. In right plot we will apply natural log, then plot approximately tends to normal.

For further analysis we will apply log on target variable charges.

In [10]:

```
f = plt.figure(figsize=(14,6))

ax = f.add_subplot(1, 2, 1)
sns.violinplot(x = 'sex', y = 'expenses', data = df, ax = ax)
ax.set_title('Violin plot of Expenses vs Sex', fontsize = 16)
ax.set_xlabel('Sex', fontsize = 12)
ax.set_ylabel('Expenses', fontsize = 12)

ax = f.add_subplot(1, 2, 2)
sns.violinplot(x='smoker', y='expenses',data = df, ax = ax)
ax.set_title('Violin plot of Expenses vs Smoker', fontsize = 16);
ax.set_xlabel('Smoker', fontsize = 12)
ax.set_ylabel('Expenses', fontsize = 12)
```

Out[10]: Text(0, 0.5, 'Expenses')



From left plot the insurance charge for male and female is approx. in same range,it is average around 5000 bucks. In right plot the insurance charge for smokers is much wide range compare to non smokers, the average charges for non smoker is approximately 5000 bucks. For smoker the minimum insurance charge is itself 5000 bucks.

In [ ]:

The Dummy variable trap is a scenario in which the independent variable are multicollinear, a scenario in which two or more variables are highly correlated. In simple term one variable can be predicted from the others.

By using pandas get_dummies function we can do all above three step in line of code. We will this fuction to get dummy variable for sex, children,smoker,region features. By setting drop_first =True function will remove dummy variable trap by droping one variable and original variable. The pandas makes our life easy.

In [11]:
```
categorical_columns = ['sex','children', 'smoker', 'region']

df_encode = pd.get_dummies(data = df, prefix = 'OHE', prefix_sep = '_', col
                           drop_first =True, dtype='int8')
```

In [12]:
```
# Lets verify the dummay variable process
print('Columns in original data frame:\n',df.columns.values)
print('\nNumber of rows and columns in the dataset:',df.shape)
print('\nColumns in data frame after encoding dummy variable:\n',df_encode
print('\nNumber of rows and columns in the dataset:',df_encode.shape)
```

```
Columns in original data frame:
 ['age' 'sex' 'bmi' 'children' 'smoker' 'region' 'expenses']

Number of rows and columns in the dataset: (1338, 7)

Columns in data frame after encoding dummy variable:
 ['age' 'bmi' 'expenses' 'OHE_male' 'OHE_1' 'OHE_2' 'OHE_3' 'OHE_4' 'OHE_5'
 'OHE_yes' 'OHE_northwest' 'OHE_southeast' 'OHE_southwest']

Number of rows and columns in the dataset: (1338, 13)
```

In [ ]:

Box -Cox transformation

A Box Cox transformation is a way to transform non-normal dependent variables into a normal shape. Normality is an important assumption for many statistical techniques; if your data isn’t normal, applying a Box-Cox means that you are able to run a broader number of tests. All that we need to perform this transformation is to find lambda value and apply the rule shown, below to your variable.

The trick of Box-Cox transformation is to find lambda value, however in practice this is quite affordable. The following function returns the transformed variable, lambda value,confidence interval.

In [13]:
```
from scipy.stats import boxcox
y_bc,lam, ci= boxcox(df_encode['expenses'],alpha=0.05)

#df['charges'] = y_bc
# it did not perform better for this model, so log transform is used
ci,lam
```

Out[13]:
```
((-0.011402950284988304, 0.09880965012231949), 0.043649029690595 08)
```

In [14]:
```
## Log transform
df_encode['expenses'] = np.log(df_encode['expenses'])
```

The original categorical variable are removed and also, one of the one hot encode varible column for perticular categorical variable is droped from the column. So we completed all three encoding step by using get dummies function.

Test Train Split

In [15]:
```
from sklearn.model_selection import train_test_split

X = df_encode.drop('expenses', axis = 1)
y = df_encode['expenses']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, ra
```

In [16]:
```
print(f'Training dataset size = {X_train.shape}\nTesting dataset size ={X_t
```

```
Training dataset size = (936, 12)
Testing dataset size =(402, 12)
```

In [ ]:

Model Building

In [17]:
```
print(type(X_train))
X_train.head()
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[17]:

	age	bmi	OHE_male	OHE_1	OHE_2	OHE_3	OHE_4	OHE_5	OHE_yes	OHE_northwes
923	34	35.8	1	0	0	0	0	0	0	
1121	46	38.2	1	0	1	0	0	0	0	
713	20	40.5	1	0	0	0	0	0	0	
552	62	21.4	1	0	0	0	0	0	0	
738	23	31.7	1	0	0	1	0	0	1	

In [ ]:

We will apply normal equation, so we need to add a colums of x0 so that our X matrix becomes m * (n+1) dimension. So, that we can multiply it theta(n+1 dimen. vector), to get the hypothesis.

hθ(x) = X * θ

In [18]:

```
X_train.insert(0, 'x_0', 1, True)
X_test.insert(0, 'x_0', 1, True)

X_train.head()
```

Out[18]:

	x_0	age	bmi	OHE_male	OHE_1	OHE_2	OHE_3	OHE_4	OHE_5	OHE_yes	OHE_nor
923	1	34	35.8	1	0	0	0	0	0	0	
1121	1	46	38.2	1	0	1	0	0	0	0	
713	1	20	40.5	1	0	0	0	0	0	0	
552	1	62	21.4	1	0	0	0	0	0	0	
738	1	23	31.7	1	0	0	1	0	0	1	

θ = (XT *X)^-1 (XT* y)

In [19]:

```
theta = np.dot( np.linalg.pinv(np.dot(X_train.transpose(), X_train)),np.dot
print(theta.shape)
theta
```

```
(13,)
```

Out[19]:

```
array([ 7.05952857,  0.03313419,  0.01350335, -0.06775339,  0.14948114,
        0.27295892,  0.24406603,  0.52341977,  0.46611078,  1.55043951,
       -0.05585593, -0.14652391, -0.13345835])
```

Hypothesis: h = X*θ

In [20]:

```
parameter = []
for i in range(0, theta.size):
             s = str(i)
             parameter.append('theta_'+s)
parameter
```

Out[20]:

```
['theta_0',
 'theta_1',
 'theta_2',
 'theta_3',
 'theta_4',
 'theta_5',
 'theta_6',
 'theta_7',
 'theta_8',
 'theta_9',
 'theta_10',
 'theta_11',
 'theta_12']
```

In [21]:

```
columns = list(X_train.columns.values)
columns
```

Out[21]:
```
['x_0',
 'age',
 'bmi',
 'OHE_male',
 'OHE_1',
 'OHE_2',
 'OHE_3',
 'OHE_4',
 'OHE_5',
 'OHE_yes',
 'OHE_northwest',
 'OHE_southeast',
 'OHE_southwest']
```

In [22]:
```
parameter_df = pd.DataFrame({'Parameter': parameter, 'Columns': columns, "t
parameter_df
```

Out[22]:

	Parameter	Columns	theta
0	theta_0	x_0	7.059529
1	theta_1	age	0.033134
2	theta_2	bmi	0.013503
3	theta_3	OHE_male	-0.067753
4	theta_4	OHE_1	0.149481
5	theta_5	OHE_2	0.272959
6	theta_6	OHE_3	0.244066
7	theta_7	OHE_4	0.523420
8	theta_8	OHE_5	0.466111
9	theta_9	OHE_yes	1.550440
10	theta_10	OHE_northwest	-0.055856
11	theta_11	OHE_southeast	-0.146524
12	theta_12	OHE_southwest	-0.133458

Using the Sk learn module

In [23]:
```
from sklearn.linear_model import LinearRegression

linear_reg = LinearRegression()
```

In [24]:
```
# Note: x_0 = 1 is no need to add, sklearn will take care of it.

X_train.drop('x_0', axis = 1, inplace = True)
X_train.head()
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packag
es/pandas/core/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

Out[24]:

	age	bmi	OHE_male	OHE_1	OHE_2	OHE_3	OHE_4	OHE_5	OHE_yes	OHE_northwes
923	34	35.8	1	0	0	0	0	0	0	
1121	46	38.2	1	0	1	0	0	0	0	
713	20	40.5	1	0	0	0	0	0	0	
552	62	21.4	1	0	0	0	0	0	0	
738	23	31.7	1	0	0	1	0	0	1	

In [25]:

```
linear_reg.fit(X_train,y_train)

linear_reg.coef_
```

Out[25]:

```
array([ 0.03313419,  0.01350335, -0.06775339,  0.14948114,  0.27295892,
        0.24406603,  0.52341977,  0.46611078,  1.55043951, -0.05585593,
       -0.14652391, -0.13345835])
```

In [26]:

```
intcpt = linear_reg.intercept_
intcpt
```

Out[26]:

```
7.059528567663586
```

In [27]:

```
Sk_learn_theta_list = [intcpt] + list(linear_reg.coef_)

parameter_df.insert(3, 'Sk_learn_theta_size', pd.Series(Sk_learn_theta_list
```

In [28]:

```
parameter_df
```

Out[28]:

	Parameter	Columns	theta	Sk_learn_theta_size
0	theta_0	x_0	7.059529	7.059529
1	theta_1	age	0.033134	0.033134
2	theta_2	bmi	0.013503	0.013503
3	theta_3	OHE_male	-0.067753	-0.067753
4	theta_4	OHE_1	0.149481	0.149481
5	theta_5	OHE_2	0.272959	0.272959
6	theta_6	OHE_3	0.244066	0.244066
7	theta_7	OHE_4	0.523420	0.523420
8	theta_8	OHE_5	0.466111	0.466111
9	theta_9	OHE_yes	1.550440	1.550440
10	theta_10	OHE_northwest	-0.055856	-0.055856
11	theta_11	OHE_southeast	-0.146524	-0.146524
12	theta_12	OHE_southwest	-0.133458	-0.133458

The parameter obtained from both the model are same. So we succefull build our model using normal equation and verified using sklearn linear regression module. Let's move ahead, next step is prediction and model evaluvation.

In [ ]:

```

```

Model evaluation

We will predict value for target variable by using our model parameter for test data set. Then compare the predicted value with actual valu in test set. We compute Mean Square Error.

$\mathbf{R}$^2 is statistical measure of how close data are to the fitted regression line. $\mathbf{R}$^2 is always between 0 to 100%. 0% indicated that model explains none of the variability of the response data around it's mean. 100% indicated that model explains all the variablity of the response data around the mean.

In [29]:

```
print(X_test.shape)
X_test.head()
```

(402, 13)

Out[29]:

	x_0	age	bmi	OHE_male	OHE_1	OHE_2	OHE_3	OHE_4	OHE_5	OHE_yes	OHE_nort
918	1	61	28.2	0	0	0	0	0	0	0	
755	1	31	27.6	1	0	1	0	0	0	0	
207	1	35	27.7	1	0	1	0	0	0	1	
520	1	50	27.4	0	0	0	0	0	0	0	
695	1	26	40.2	0	0	0	0	0	0	0	

In [30]:

```
# h = X*θ
h = np.dot(X_test, theta)
```

In [31]:

```
print(h.shape)
h
```

```
(402,)
```

Out[31]:

```
array([ 9.32805005,  8.66458629, 10.34891288,  9.08622966,  8.40799605,
        8.98058228, 10.75032114,  8.56052118,  9.48603048, 10.95606736,
        8.12785765, 11.03794019,  8.19898243,  9.44491415,  8.80664345,
        8.53343524,  8.63333705,  8.11097403, 10.44181875, 10.44535006,
       11.202992  ,  9.15833381,  8.86362536,  9.18813745, 10.58994956,
        8.51004869,  8.55359744,  9.61553214,  7.98433184,  9.11897867,
        9.38808895,  8.16733895,  8.40485668,  9.45114868,  8.77555608,
        9.82525033,  9.36916714,  9.02919433,  9.20857257,  8.38894405,
        9.19702638, 10.59534835,  9.19110653, 10.55958277,  9.2601245 ,
        9.29877713,  8.47155953,  8.55708172,  9.06506726,  8.69377145,
        9.41448569,  8.84660162,  9.13857681,  8.30125144,  8.44160397,
        9.35333029,  9.25801621,  9.16717352,  8.82323265,  9.61945118,
        9.95122179,  8.4934474 ,  7.90618177,  8.44172443,  8.13696631,
        8.62640203,  8.02659821,  7.94534148,  8.43064771,  9.45560958,
        8.67928113, 10.57360518,  9.07366898, 10.33109984,  8.02140083,
        8.50708542,  9.16944243,  8.51226198,  8.36544333,  8.32232341,
        9.24992337,  8.06667963,  9.97751655,  9.62525744,  8.89662759,
        8.97040378,  9.51272483, 11.13244572,  8.05795344, 10.8002391 ,
        8.48244211,  9.30393947,  8.05377047,  8.89542456, 10.66445463,
        7.96951564,  7.94086185,  8.0393427 ,  9.79513787,  7.75523266,
        9.34011025,  8.63597626,  8.14455702,  9.11378128,  9.45890971,
        9.20559899,  9.00986025,  8.08859597,  8.27805376,  8.77844917,
        9.63749876,  8.8937768 ,  8.462685  ,  9.14887576,  8.8606014 ,
        8.04483761, 11.05507655,  9.43895446,  9.8341629 ,  8.10854   ,
        9.2838921 ,  9.058505  ,  8.48412691,  8.6344887 ,  8.19649723,
        8.06625024,  8.39664119, 10.45997311,  8.99625493,  9.81233277,
       10.44958495,  7.86055876,  9.15665671,  8.53063394,  8.8384912 ,
        8.11659668,  8.02333424,  9.12640957,  8.09398309,  8.0312023 ,
        9.52475755,  9.56828411,  9.57615217,  9.20514261,  8.68399683,
        8.97376165, 10.67357418,  8.16257601,  7.85434686,  8.56158325,
        9.07426872,  9.5863551 ,  8.2026432 ,  9.49427679,  9.75099045,
        8.11981641,  8.40730918,  8.83426927,  8.45148238,  8.24499308,
        8.16479197,  8.71592932,  8.03641736,  9.62386269,  8.79627016,
        8.50617288, 10.38709343,  8.33456418,  8.11353534,  9.00633145,
        8.35199552,  9.47440748,  7.89132809,  8.14114282,  8.08910395,
        8.95828884,  8.99971196,  8.40615732,  9.28248871,  8.20012881,
```

```
 8.87090764,  9.01225531, 10.27211103,  7.9702745 ,  9.32597356,
 9.48738081,  8.07524115,  9.09888953,  9.64519688,  9.6581029 ,
 9.11068187, 10.34696522,  9.16441439,  8.80732909,  8.77091057,
10.64768789,  8.83674168,  9.11321787,  9.08773458,  9.23661482,
 9.92847006, 10.32555779,  7.91956748,  7.96588486,  8.55933128,
 8.20084803, 10.02230003,  9.31797353, 10.39614403,  9.2071356 ,
10.02061031,  8.48762046,  8.80790938,  8.24520209,  8.62709917,
10.23426912,  9.56580642,  8.50018485,  9.2316598 ,  9.34323955,
 9.47563322,  9.74766875,  8.28748604, 10.48250076,  8.53805871,
 8.96797502,  8.54958755,  9.37707359,  7.95801951,  9.82512732,
 9.88222645,  9.96286817,  8.8587843 ,  8.33013615,  8.49631666,
 9.0082107 ,  8.6950582 ,  8.7641279 ,  8.05360491, 10.66644949,
 8.20221382,  8.94911699,  8.74793784,  9.4966555 ,  9.20195994,
 9.24898334, 10.36483705,  8.42582869,  9.03060298,  8.52576847,
 7.89876753,  8.95359662,  8.19877848,  8.5745753 ,  8.66121331,
 9.97632965,  9.21232604,  9.25384241,  8.67098043, 10.79898896,
 9.37778766, 10.53864692,  9.05392594, 10.22209972,  8.24275832,
10.31758227,  9.42766015,  7.79028965,  8.74260649,  9.18437199,
 8.49875592,  8.9104559 ,  8.12973935,  9.44293047,  8.2070691 ,
 8.13168388,  9.17050555,  8.69548456, 10.44951657,  9.87382767,
 8.49210291,  8.97221576,  8.70387552,  8.75634378,  9.48280266,
 8.29413855,  8.76967787,  8.18475862,  8.4979689 ,  9.75150412,
 8.82438754,  8.76763227,  9.98746424,  8.69465914,  7.87064445,
 8.19638374,  8.76198346,  8.9038709 ,  7.95238151,  8.90370422,
 8.71127767,  9.15598866,  8.55448858,  9.46368921,  8.6933472 ,
 9.04626758,  7.84811738,  9.67924834,  9.4142909 ,  9.69602011,
 8.78523354,  9.25639174, 10.85189377,  8.75519605,  7.80177738,
 8.66172288,  8.51029061,  8.87103929,  9.68200719, 10.12772465,
 9.01120608,  9.27521484,  8.38041266,  8.65746542,  8.62242849,
 9.10880933,  8.42307631,  9.19102662,  9.11595438,  9.12368776,
 8.33587341, 10.80101553,  9.71266548,  8.02646636,  9.27973271,
 8.27038652,  9.85506607,  8.83344954,  9.06319775,  9.84553805,
 8.91463887,  7.87851251,  9.09489684,  9.17069648,  8.86486276,
 8.00067532,  9.94666584,  7.97867122,  8.86989116,  9.19457446,
 9.68461157,  8.99596656,  9.59754945, 10.73127922, 10.5853738 ,
 8.16493581,  8.54488974,  8.50958549,  9.43006047,  8.86165356,
 8.86886082,  8.78884462,  8.24855458,  8.86039333,  9.28712864,
 8.87216318,  8.54114121,  7.9458665 ,  8.88719427, 11.29271142,
 9.37107438,  9.22414324,  9.5032341 ,  9.26140555,  9.5088557 ,
 9.29274656,  8.20442675, 10.1219755 , 10.54742059,  9.19746082,
 9.16637627,  9.8794501 ,  9.01217058,  9.03755984,  9.37228434,
 8.08244494,  9.22720939,  9.28022493,  9.46885155,  8.88184197,
 9.51144177,  8.97852682,  8.74670115,  7.98315367,  9.1119995 ,
10.65539262, 10.5267722 ,  8.18477566,  8.86188437, 10.48196507,
 7.89672943,  8.25051902])
```

In [32]:

```
#MSE

J_mse = np.sum((h - y_test)**2)/ X_test.size
J_mse
```

Out[32]:

```
0.014406571289403074
```

In [33]:
```
# R_square
sse = np.sum((h - y_test)**2)        #Sum of square error
sst = np.sum((y_test - y_test.mean())**2)     #Sum of square total

R_square = 1 - (sse/sst)
R_square
```

Out[33]:
```
0.7795814253322737
```

In [34]:
```
print(f'Mean square error = {J_mse}\nR square = {R_square}')
```

```
Mean square error = 0.014406571289403074
R square = 0.7795814253322737
```

In [ ]:
```

```

Using the Sk learn library

In [35]:
```
X_test.drop('x_0', axis = 1, inplace = True)

h_sk = linear_reg.predict(X_test)

#MSE
from sklearn.metrics import mean_squared_error
J_mse_sk = mean_squared_error(h, y_test)
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packag
es/pandas/core/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

In [36]:
```
#MSE
from sklearn.metrics import mean_squared_error
J_mse_sk = mean_squared_error(h, y_test)

# R_square
R_square_sk = linear_reg.score(X_test,y_test)

print(f'Mean square error = {J_mse_sk}\nR square = {R_square_sk}')
```

```
Mean square error = 0.18728542676223997
R square = 0.7795814253322967
```

Model Validation

In order to validated model we need to check few assumption of linear regression model. The common assumption for Linear Regression model are following:-

1 - Linear Relationship: In linear regression the relationship between the dependent and independent variable to be linear. This can be checked by scatter ploting Actual value Vs Predicted value

2 - The residual error plot should be normally distributed.

3 - The mean of residual error should be 0 or close to 0 as much as possible

4 - The linear regression require all variables to be multivariate normal. This assumption can best checked with Q-Q plot.

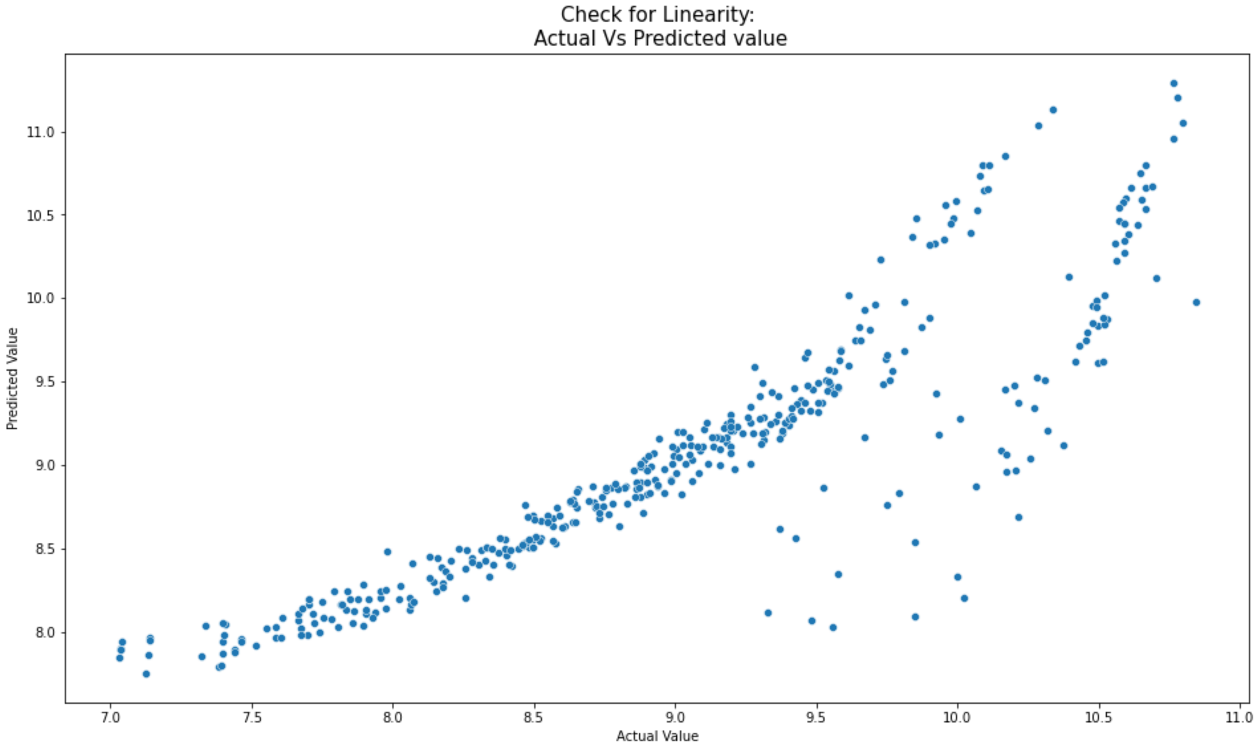
5 - Linear regession assumes that there is little or no Multicollinearity in the data. Multicollinearity occurs when the independent variables are too highly correlated with each other. The variance inflation factor VIF* identifies correlation between independent variables and strength of that correlation. $\mathbf{VIF} = \frac{1}{1-\mathbf{R}^2}$ VIF =11−R2, If VIF >1 & VIF <5 moderate correlation, VIF < 5 critical level of multicollinearity.

6 - Homoscedasticity: The data are homoscedastic meaning the residuals are equal across the regression line. We can look at residual Vs fitted value scatter plot. If heteroscedastic plot would exhibit a funnel shape pattern.

In [37]:

```
#Point 1
plt.figure(figsize = (16,9))
sns.scatterplot(x = y_test, y = h)

plt.title('Check for Linearity:\n Actual Vs Predicted value', fontsize = 1!
plt.xlabel('Actual Value', fontsize = 10)
plt.ylabel('Predicted Value', fontsize = 10)
```

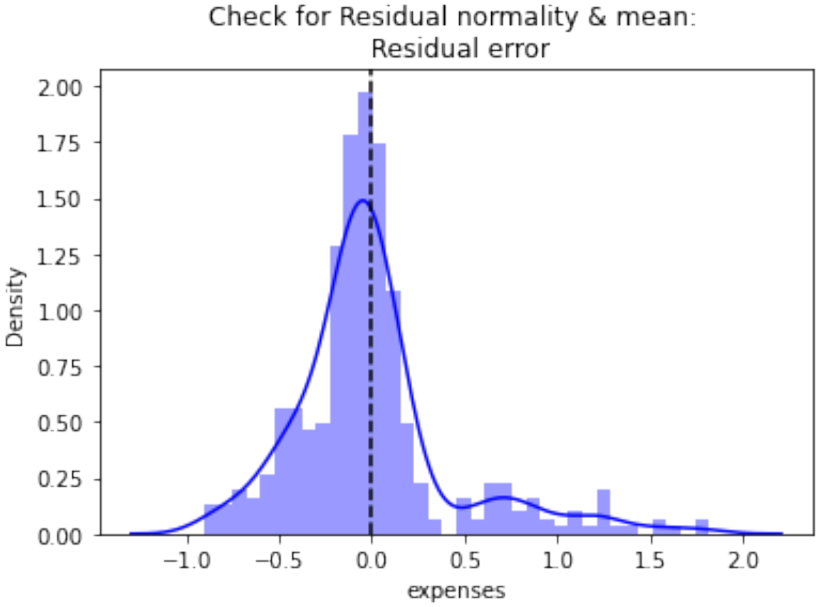
Out[37]: Text(0, 0.5, 'Predicted Value')



In [38]:

```
#Point 2
# Check for Residual normality & mean

sns.distplot((y_test - h),color='b')

plt.axvline((y_test - h).mean(),color='k',linestyle='--')

plt.title('Check for Residual normality & mean: \n Residual error');
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packag
es/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to
use either `displot` (a figure-level function with similar flexibility) or
`histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [ ]: