Importing Libraries

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # for data processing like in this project csv loading
import matplotlib.pyplot as plt  # for data visualization
import seaborn as sns # for data visualization i.e pairplot
from sklearn import metrics # for calculating rootmean square
```

Data Loading and Insites

In [2]:
```
# reading the data

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

# checking the shape
print(df.shape)
```

```
(1338, 8)
```

In [3]:
```
# checking data points
print(df.size)
```

```
10704
```

In [4]:
```
# previewing 1st 5 dataset checking wether its loaded or not sucessfully
df.head()
```

Out[4]:

	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 [ ]:

In [5]:
```
#description about data set

df.describe()
```

Out[5]:

	age	sex	bmi	children	smoker	region	charges	insuranceclaim
count	1338.000000	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	0.585202
std	14.049960	0.500160	6.098187	1.205493	0.403694	1.104885	12110.011237	0.492871
min	18.000000	0.000000	15.960000	0.000000	0.000000	0.000000	1121.873900	0.000000
25%	27.000000	0.000000	26.296250	0.000000	0.000000	1.000000	4740.287150	0.000000
50%	39.000000	1.000000	30.400000	1.000000	0.000000	2.000000	9382.033000	1.000000
75%	51.000000	1.000000	34.693750	2.000000	0.000000	2.000000	16639.912515	1.000000
max	64.000000	1.000000	53.130000	5.000000	1.000000	3.000000	63770.428010	1.000000

In [6]:
```
#checing information about data

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
```

In [7]:
```
# checking number of null value in this data
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 if any null value is present or not
df.isnull().any()
```

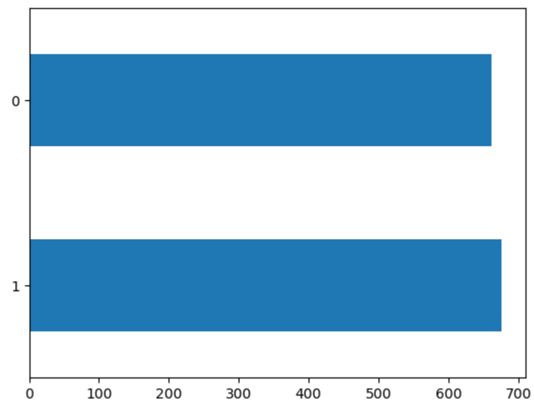
Out[8]:
```
age               False
sex               False
bmi               False
children          False
smoker            False
region            False
charges           False
insuranceclaim    False
dtype: bool
```

In [9]:
```
# from this data we can get insites that :
# 1. data belongs to middle age people (mostly)
# 2. maximum age of any person is 64 where as minimum age is 18 only
# 3. maximum bmi is 53.13 which is a deep sign of obasity
# 4. there is no null value in this data
# 5. There are 676 male and 662 female
```

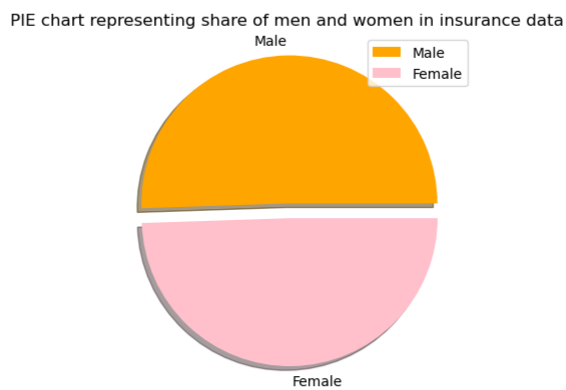
In [10]:
```
# checking value count of male and female in data
df['sex'].value_counts()
```

Out[10]:
```
1    676
0    662
Name: sex, dtype: int64
```

In [11]:
```
# ploting a bar graph showing about number of male and female
df.sex.value_counts(normalize=False).plot.barh()
plt.show()
```



In [12]:
```
#pie chart: with Label and explode
mylables=["Male","Female"] # here label is "Male - is 1 where as Female - is 0"
colors = ['orange', 'pink']
myexplode=[0.10,0]
size = [676, 662]
plt.pie(size,colors = colors,labels =mylables,explode = myexplode, shadow = True)
plt.title('PIE chart representing share of men and women in insurance data ')
plt.legend()
plt.show()
```

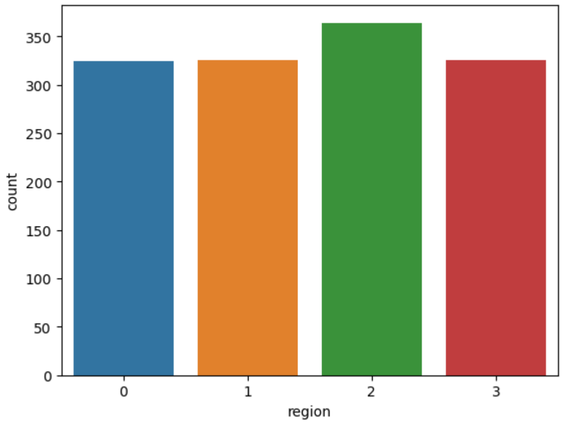


In [13]:
```
# checking customer belonging
df['region'].value_counts()
```

Out[13]:
```
2    364
3    325
1    325
0    324
Name: region, dtype: int64
```

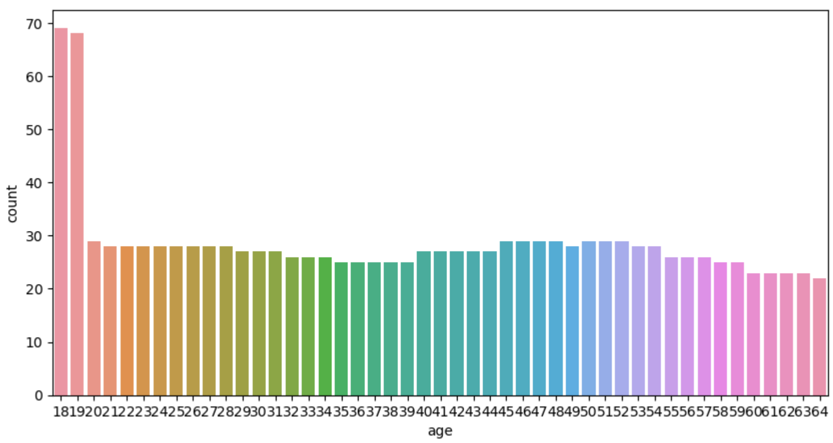
In [14]:
```
#ploting a Countplot showing region
sns.countplot("region",data = df)
plt.show()
```

```
D:\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argume
nt will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

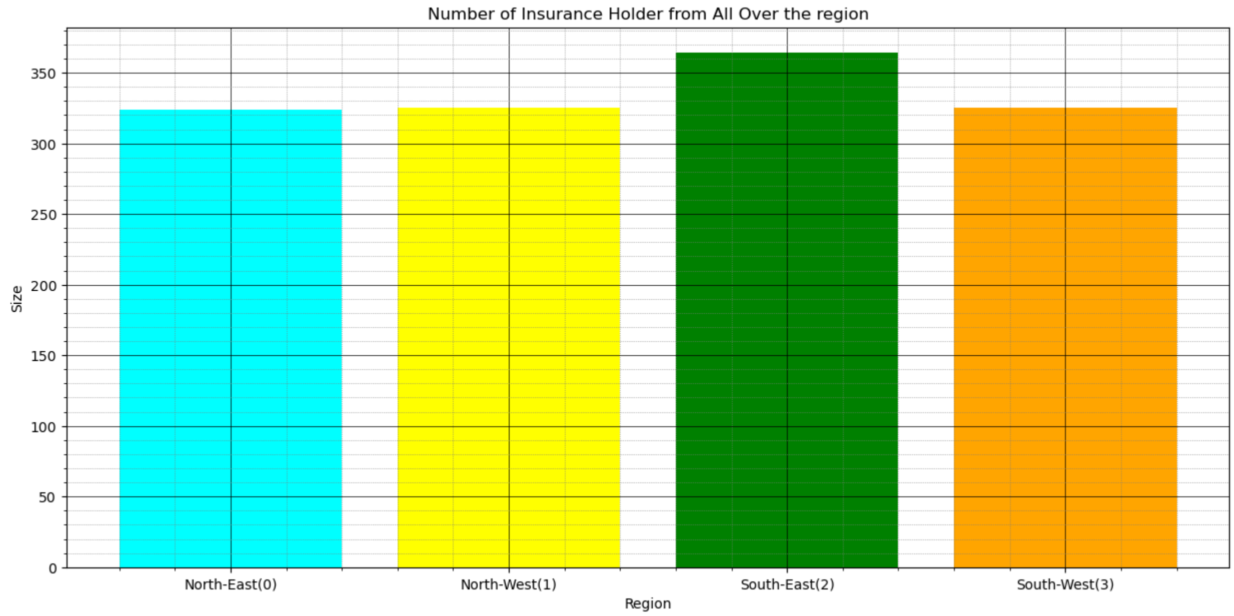


In [15]:
```
#ploting a Countplot showing age
plt.figure(figsize = (10,5))
sns.countplot("age",data = df)
plt.show()
```

```
D:\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argume
nt will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [16]:
```python
# ploting a bar graph showing about region wise with labels grid and minor grids and title

x = ['North-East(0)', 'North-West(1)', 'South-East(2)', 'South-West(3)']
size = [324, 325, 364, 325]
plt.figure(figsize = (15,7))
x_pos = [i for i, _ in enumerate(x)]
plt.bar(x_pos, size, color=['cyan', 'yellow', 'green', 'orange'])
plt.xlabel("Region")
plt.ylabel("Size")
plt.title("Number of Insurance Holder from All Over the region")
plt.xticks(x_pos, x)
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='black')
# Customize the minor grid
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='grey')
plt.show()
```



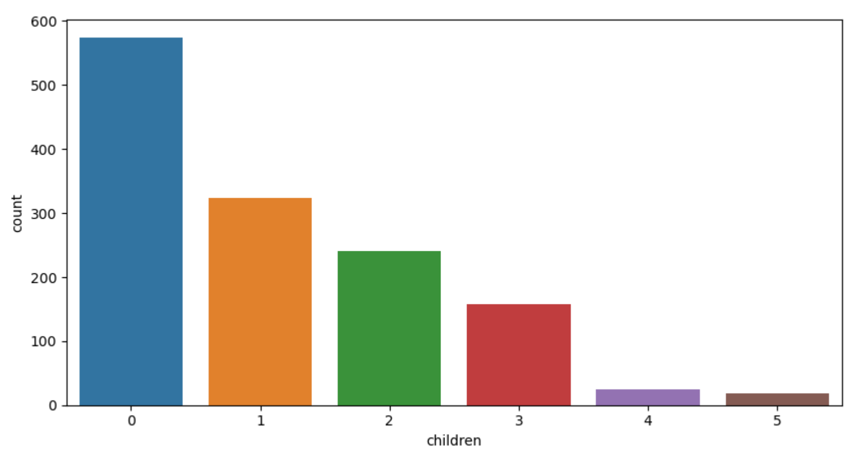
In [17]:
```python
# 6.people belonging residential area from  northeast(0) are 324 person ; northwest(1) are 325 person ; southeast(2) are 364 person ; southwest(3) are 325 person
```

In [18]:
```python
# checking children count
df['children'].value_counts()
```

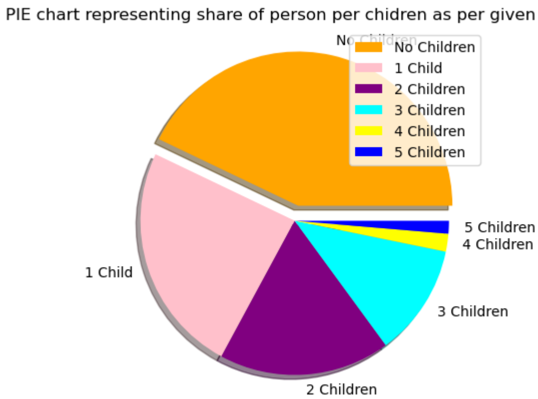
Out[18]:
```
0    574
1    324
2    240
3    157
4     25
5     18
Name: children, dtype: int64
```

In [19]:
```python
#ploting a Countplot showing number of children
plt.figure(figsize = (10,5))
sns.countplot("children",data = df)
plt.show()
```

```
D:\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argume
nt will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [20]:
```python
#pie chart: with Label and explode
plt.figure(figsize = (10,5))
mylables=["No Children","1 Child","2 Children","3 Children","4 Children","5 Children"]
colors = ['orange','pink','purple','cyan','yellow','blue']
myexplode=[0.10,0,0,0,0,0]
size = [574, 324,240,157,25,18]
plt.pie(size,colors = colors,labels =mylables,explode = myexplode, shadow = True)
plt.title('PIE chart representing share of person per chidren as per given data ')
plt.legend()
plt.show()
```

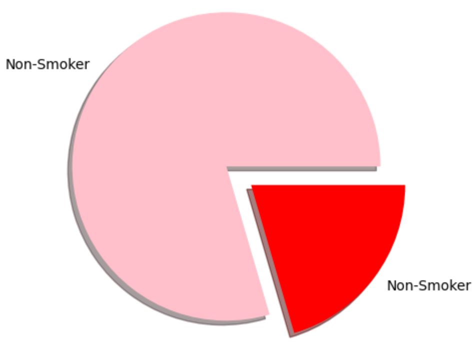


In [21]:
```python
# 7. count of people having no children are 574 ; with 1 children are 324 ; with 2 children are 240 ; with 3 children are  157; with 4 children are 25 and with 5 cildren are
```

In [22]:
```python
# checking number of smokers
df['smoker'].value_counts()
```

Out[22]:
```
0    1064
1     274
Name: smoker, dtype: int64
```

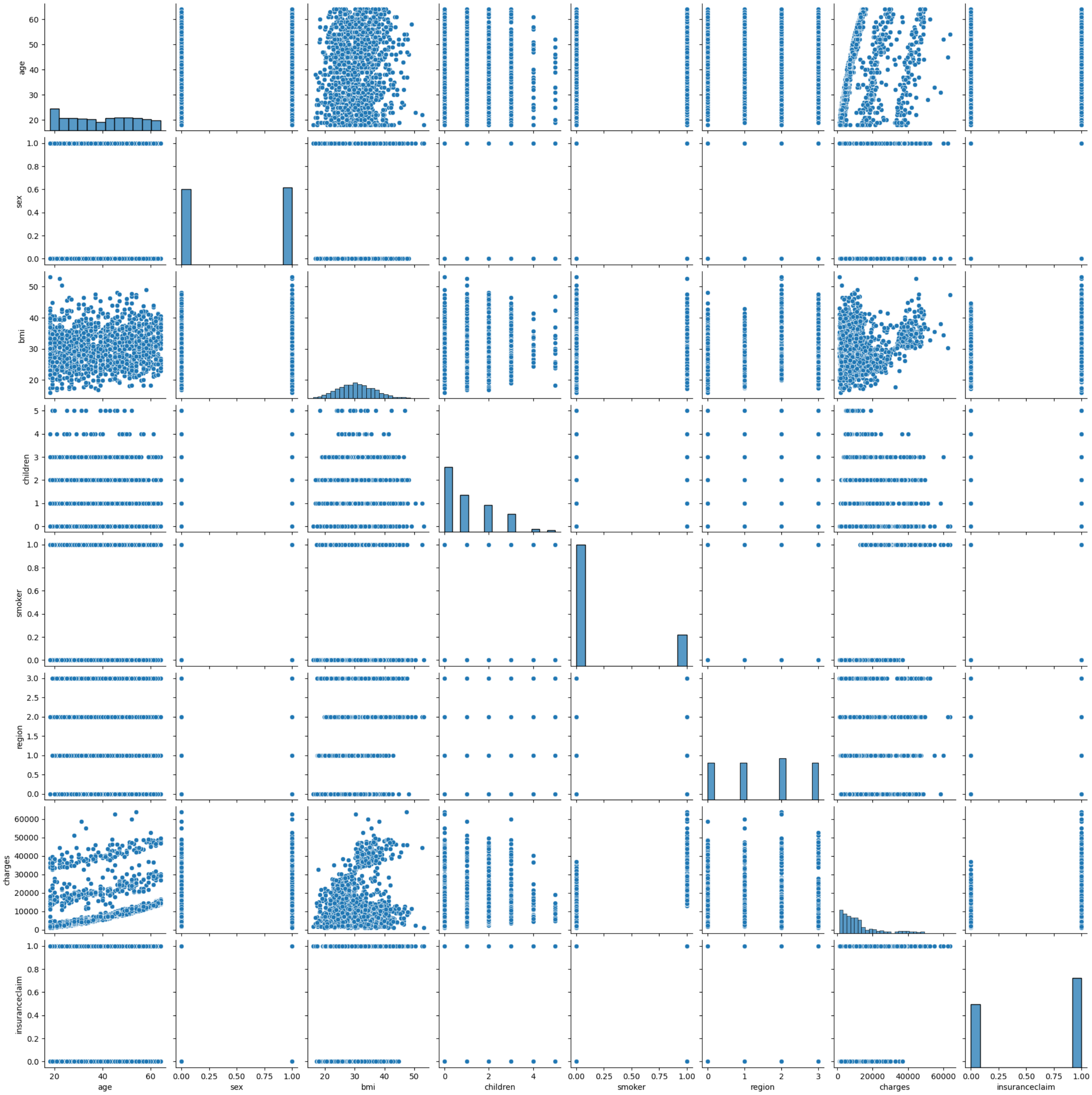
In [23]:
```python
#ploting a bar grap showing number of smoker
plt.figure(figsize = (10,5))
mylables=['Non-Smoker','Non-Smoker']
colors = ['pink','Red']
myexplode=[0.10,0.10]
size = [1064,274]
plt.pie(size,colors = colors,labels =mylables,explode = myexplode, shadow = True)

plt.show()
```



In [24]:
```python
# 8. count of non smokers are represented as 0 which is 1064 where as 274 peopple are smoker
```

In [25]:
```
# pairplot
sns.pairplot(df)
```

Out[25]: <seaborn.axisgrid.PairGrid at 0x1c696bf81c0>



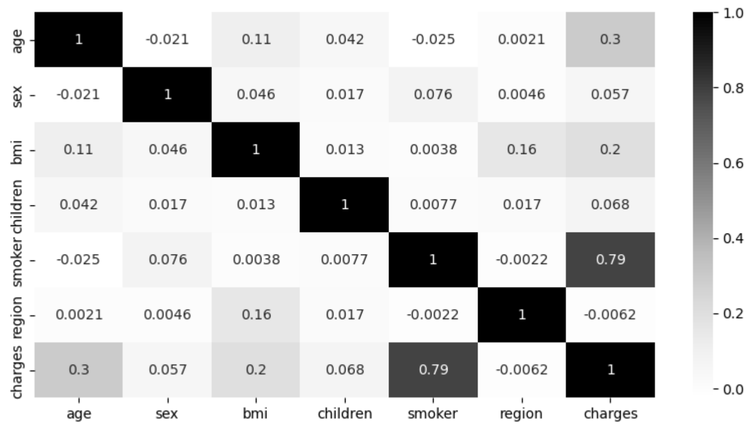
In [26]:
```
# Corelation Between Diffrent Feature
df[['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges']].corr()
```

Out[26]:

	age	sex	bmi	children	smoker	region	charges
age	1.000000	-0.020856	0.109272	0.042469	-0.025019	0.002127	0.299008
sex	-0.020856	1.000000	0.046371	0.017163	0.076185	0.004588	0.057292
bmi	0.109272	0.046371	1.000000	0.012759	0.003750	0.157566	0.198341
children	0.042469	0.017163	0.012759	1.000000	0.007673	0.016569	0.067998
smoker	-0.025019	0.076185	0.003750	0.007673	1.000000	-0.002181	0.787251
region	0.002127	0.004588	0.157566	0.016569	-0.002181	1.000000	-0.006208
charges	0.299008	0.057292	0.198341	0.067998	0.787251	-0.006208	1.000000

In [27]:
```
# Insite from this Corelation are : 1. Charges are Dependent Upon Age (Higher the Age More will be  Insurance Charges) ; 2. Charges are Dependent upon Smokers and BMI
```

In [28]:
```
#plot the correlation matrix of salary, balance and age in data dataframe.
plt.figure(figsize = (10,5))
sns.heatmap(df[['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges']].corr(), annot=True, cmap = 'Greys')
plt.show()
```



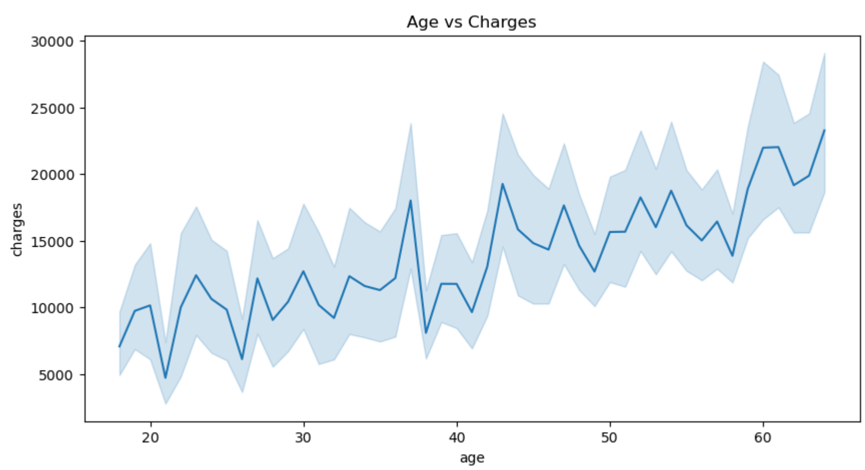
In [29]:
```
df.columns
```

Out[29]:
```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges',
       'insuranceclaim'],
      dtype='object')
```

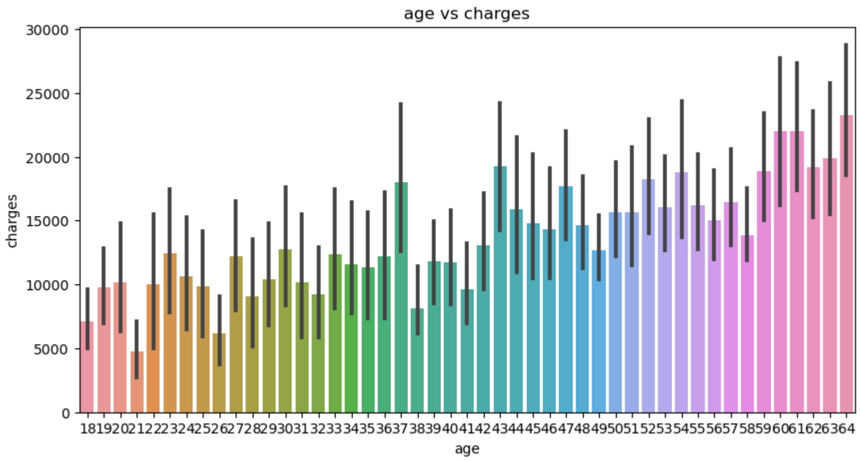
In [30]:
```
# Insites From this Heat Map :- 1. Smoker Tends to Pay More Insurance Charges; 2. Age is Positively Related to Charge; 3. Charges are also propotional to bmi
```

In [ ]:

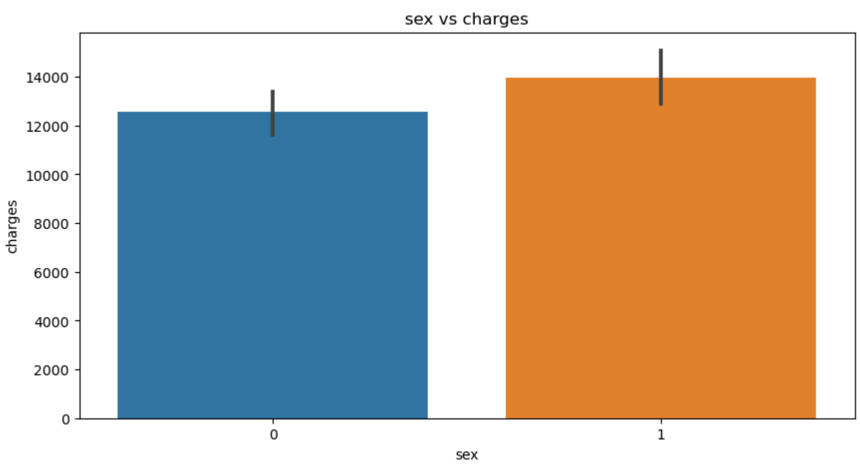
In [31]:
```
# Age vs Charges
# the more the age the more will be insurance charge (roughly estimated)

plt.figure(figsize = (10, 5))
sns.lineplot(x = 'age', y = 'charges', data = df)

plt.title("Age vs Charges")
plt.show()
```



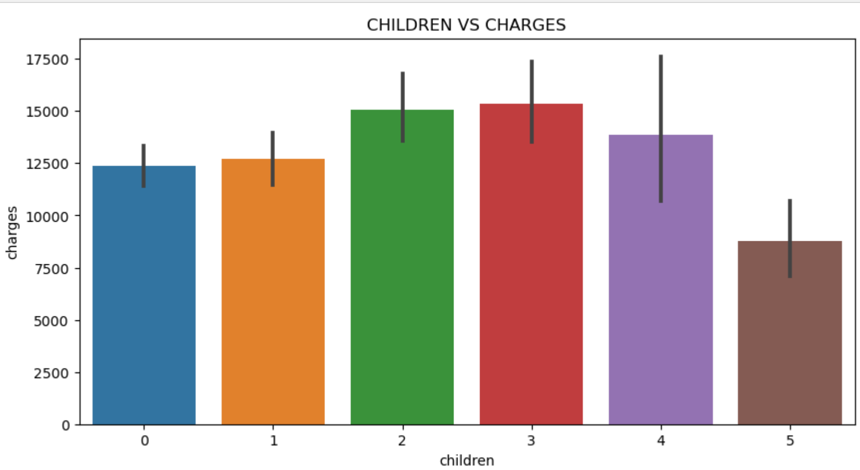
In [32]:
```
#bax plot for age vs charge

plt.figure(figsize = (10, 5))
sns.barplot(x = 'age', y = 'charges', data = df)

plt.title('age vs charges')
plt.show()
```



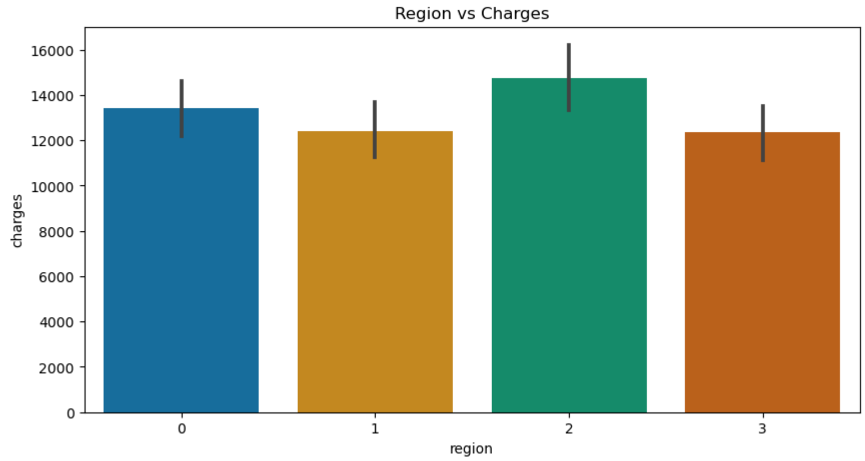
In [33]:
```
#plot the box plot of sex and charges
# as 1 belongs to men : it shows that men are paying more insurance charges then Women (in general)
#bar plot

plt.figure(figsize = (10, 5))
sns.barplot(x = 'sex', y = 'charges', data = df)

plt.title('sex vs charges')
plt.show()
```



In [34]:
```
# children vs charges
# no. of childrens of a person has a weird dependency on insurance charge. i.e(parents of more children tends to pay less insurance charge whereas parents of 2 and 3 children

plt.figure(figsize = (10, 5))
sns.barplot(x = 'children', y = 'charges', data = df)

plt.title('CHILDREN VS CHARGES')
plt.show()
```

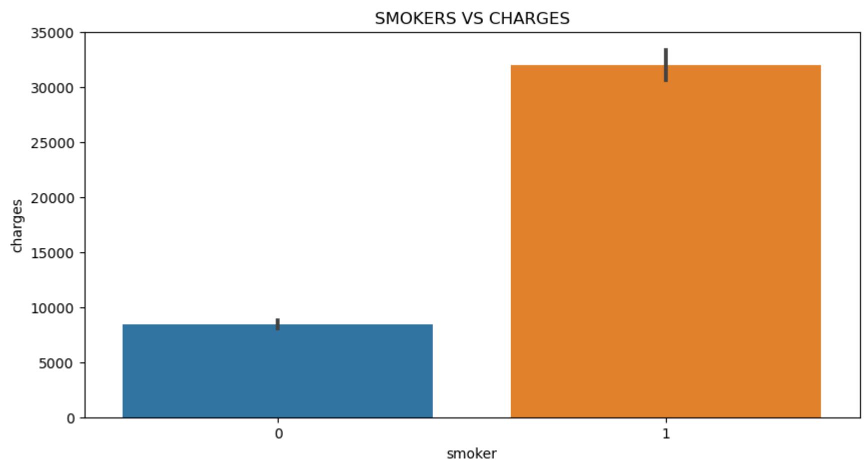


In [35]:
```
# region vs charges BAR GRAPh

plt.figure(figsize = (10, 5))
sns.barplot(x = 'region', y = 'charges', data = df, palette = 'colorblind')

plt.title('Region vs Charges')
plt.show()
```



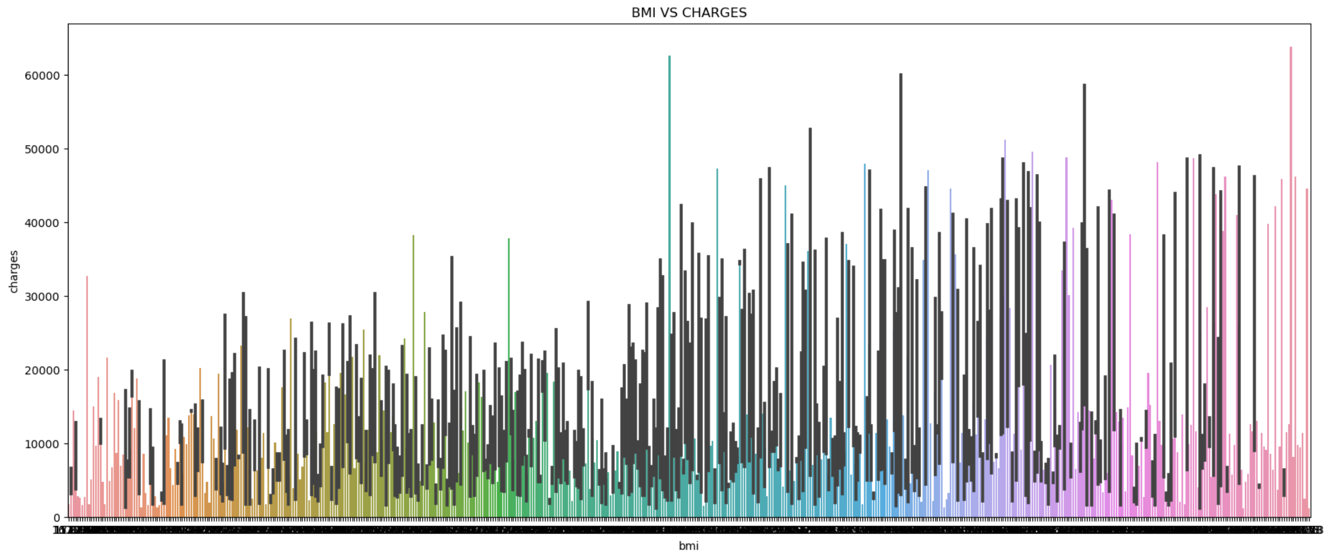
In [36]:
```
# from the graph we can clearly state that region dont play any role in charges it is highly independent (Should be Drop as it feels Unnecessary)
```

In [37]:
```
# smoker vs charges
plt.figure(figsize = (10, 5))
sns.barplot(x = 'smoker', y = 'charges', data = df)

plt.title('SMOKERS VS CHARGES')
plt.show()
```



In [38]:
```
# from the graph where 0 represents non smoker and 1 represent smoker it is clear that smoker tends to pay higher primium than non smokers
```

In [39]:
```
# BMI vs charges
plt.figure(figsize = (20,8))
sns.barplot(x = 'bmi', y = 'charges', data = df)

plt.title('BMI VS CHARGES')
plt.show()
```



In [40]:
```
# From Graph we can canclude that higher the BMI more will be Insurance Premium Charges
```

Data Cleaning

In [41]:
```
# removing un required columns from the insurance data
# As from the above grph we can clearly state that region dont play any role in charges it is highly independent (Should be Drop as it feels Unnecessary)

df = df.drop('region', axis = 1)
```

In [42]:
```
df.shape
```

Out[42]: (1338, 7)

In [43]:
```
#as earlier there was 10704 data point the new one has 9366 data point after removing region
df.size
```

Out[43]: 9366

In [44]:
```
# seperate out features and target value from dataset

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

In [45]:
```
X.shape
```

Out[45]: (1338, 6)

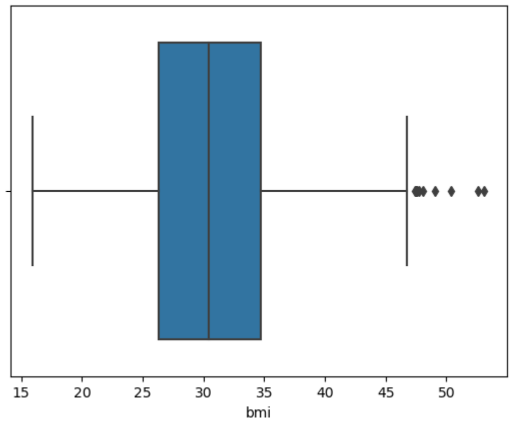
In [46]:
```
y.shape
```

Out[46]: (1338,)

Finding an Outlier

In [47]:
```
#bmi outlier

sns.boxplot(df["bmi"])
plt.show()
```

```
D:\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argume
nt will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



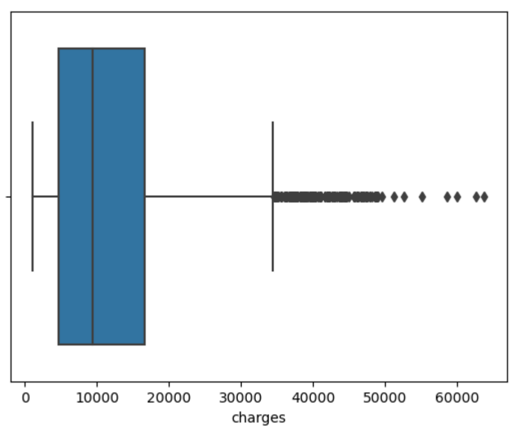
In [48]:
```
# Finding Position of Outlier
#position plot of outlier

print(np.where(df["bmi"]>45))
```

```
(array([ 116,  286,  292,  401,  438,  454,  543,  547,  549,  582,  660,
        847,  860,  930,  941, 1024, 1047, 1088, 1131, 1317], dtype=int64),)
```

In [49]:
```
# bmi can be more or less as per medical condition of person so no need to treat it as per this data
```

In [50]:
```
#Charges outlier

sns.boxplot(df["charges"])
plt.show()
```

```
D:\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argume
nt will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [51]:
```
# Charges can be More or Less as per required by insurance company
```

Spliting Data (Training and Testing Data) and Importing Sklearn Modules

In [52]:
```
#spliting data into training and testing data set

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.20, random_state =42)  # for optimal value and inhance the testing accuracy test size can be increases or d
```

In [53]:
```
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 :  (1070, 6)
X_test shape :  (268, 6)
y_train shape :  (1070,)
y_test shape :  (268,)
```

Importing and Using Decision Tree (Supervised Learning) Algorithm

In [54]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [55]:
```
# model

dtc = DecisionTreeClassifier()

#fitting

dtc.fit(X_train,y_train)
```

Out[55]:
```
DecisionTreeClassifier()
```

In [56]:
```
#predicting via Decision Tree Algorithm

y_pred=dtc.predict(X_test)

y_pred
```

Out[56]:
```
array([0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 1], dtype=int64)
```

In [57]:
```
#Calculating RMSE Root MEan Square Error

rmse= np.sqrt(metrics.mean_squared_error(y_test,y_pred))
print("Root Mean Square Error = ",rmse)
```

```
Root Mean Square Error =  0.16161498378886355
```

Checking Out Training and Testing Data Accuracy (Actual vs Predicted)

In [58]:
```
# compute accuracy on training set

dtc_train= dtc.score(X_train,y_train)

print("Training Data Accuracy by Decision Tree Algorithm is : " , dtc_train)

# compute accuracy on testing set

dtc_test= dtc.score(X_test,y_test)

print("Testing Data Accuracy by Decision Tree Algorithm is : " , dtc_test)
```

```
Training Data Accuracy by Decision Tree Algorithm is :  1.0
Testing Data Accuracy by Decision Tree Algorithm is :  0.9738805970149254
```

In [59]:
```
# Here Training Accuracy is Greater than Testing Accuracy means our Model is Overfited
```

In [60]:
```
# calculating the mean squared error
mse = np.mean((y_test - y_pred)**2, axis = None)
print("MSE :", mse)

# Calculating the root mean squared error
rmse = np.sqrt(mse)
print("RMSE :", rmse)
```

```
MSE : 0.026119402985074626
RMSE : 0.16161498378886355
```

Using Hyperparameter Tuning for Decision Tree

In [61]:
```
parameters = {"splitter" : ["best", "random"],
              "max_depth" : [1,3,5,7,9,11],
              "min_samples_leaf" : [1,2,3,4,5,6,7,8,9,10],
              "min_weight_fraction_leaf": [0.2,0.3],
              "max_features":["auto","log2","sqrt",None],
              "max_leaf_nodes":[None,10,20,30]
}
```

In [62]:
```
#using grid search cv

from sklearn.model_selection import GridSearchCV
```

In [63]:
```
tuning_model = GridSearchCV(dtc,param_grid = parameters,
                            scoring= "neg_mean_squared_error",
                            cv=3,verbose=3)
```

In [64]:
```
tuning_model.fit(X,y)
```

```
Fitting 3 folds for each of 3840 candidates, totalling 11520 fits
[CV 1/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=best;, score=-0.300 total time=   0.0s
[CV 2/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=best;, score=-0.265 total time=   0.0s
[CV 3/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=best;, score=-0.417 total time=   0.0s
[CV 1/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=random;, score=-0.415 total time=   0.0s
[CV 2/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=random;, score=-0.415 total time=   0.0s
[CV 3/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=random;, score=-0.415 total time=   0.0s
[CV 1/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.3, splitter=best;, score=-0.379 total time=   0.0s
[CV 2/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.3, splitter=best;, score=-0.280 total time=   0.0s
[CV 3/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.3, splitter=best;, score=-0.410 total time=   0.0s
[CV 1/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.3, splitter=random;, score=-0.415 total time=   0.0s
[CV 2/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.3, splitter=random;, score=-0.415 total time=   0.0s
[CV 3/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.3, splitter=random;, score=-0.350 total time=   0.0s
[CV 1/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=2, min_weight_fraction_leaf=0.2, splitter=best;, score=-0.300 total time=   0.0s
[CV 2/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=2, min_weight_fraction_leaf=0.2, splitter=best;, score=-0.448 total time=   0.0s
[CV 3/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=2, min_weight_fraction_leaf=0.2, splitter=best;, score=-0.350 total time=   0.0s
[CV 1/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=2, min_weight_fraction_leaf=0.2, splitter=random;, score=-0.321 total time=   0.0s
[CV 2/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=2, min_weight_fraction_leaf=0.2, splitter=random;, score=-0.280 total time=   0.0s
[CV 3/3] END max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=2, min_weight_fraction_leaf=0.2, splitter=random;, score=-0.415 total time=   0.0s
```

In [82]:
```
dtc.get_params().keys()
```

Out[82]:
```
dict_keys(['ccp_alpha', 'class_weight', 'criterion', 'max_depth', 'max_features', 'max_leaf_nodes', 'min_impurity_decrease', 'min_samples_leaf', 'min_samples_split', 'min_we
ight_fraction_leaf', 'random_state', 'splitter'])
```

In [83]:
```
#best parameters

tuning_model.best_params_
```

Out[83]:
```
{'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_samples_leaf': 1,
 'min_weight_fraction_leaf': 0.2,
 'splitter': 'best'}
```

In [84]:
```
#using this type of hyper parameters to train our model once again

tuned_model=DecisionTreeClassifier(max_depth=3,min_samples_leaf=1,min_weight_fraction_leaf=0.2,
                                   splitter="best")
```

In [85]:
```
#fitting model

tuned_model.fit(X_train,y_train)
```

Out[85]:
```
DecisionTreeClassifier(max_depth=3, min_weight_fraction_leaf=0.2)
```

In [86]:
```
#prediction

tuned_pred=tuned_model.predict(X_test)
```

In [87]:
```
# compute accuracy on training set

tuned_model_train= tuned_model.score(X_train,y_train)

print("Training Data Accuracy by Decision Tree Tuned Algorithm is : " , tuned_model_train)

# compute accuracy on testing set

tuned_model_test= tuned_model.score(X_test,y_test)

print("Testing Data Accuracy by Decision Tree Algorithm is : " , tuned_model_test)
```

```
Training Data Accuracy by Decision Tree Tuned Algorithm is :  0.8046728971962617
Testing Data Accuracy by Decision Tree Algorithm is :  0.8208955223880597
```

In [88]:
```
#Calculating RMSE

rmse= np.sqrt(metrics.mean_squared_error(y_test,tuned_pred))
print("Root Mean Square Error = ",rmse)
```

```
Root Mean Square Error =  0.42320736951515897
```

In [89]:
```
# calculating the mean squared error
mse = np.mean((y_test - y_pred)**2, axis = None)
print("MSE :", mse)
```

```
MSE : 0.1791044776119403
```

Imorting and Using Logistic Regression

In [90]:
```
from sklearn.metrics import accuracy_score, confusion_matrix # imporing for error calculation
from sklearn.linear_model import LogisticRegression # imporing Logistic Regression
```

In [91]:
```
# Logistic Regression model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
D:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[91]:
```
LogisticRegression()
```

In [92]:
```
y_pred = logreg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy: ", accuracy)
print("Confusion matrix: \n", conf_matrix)
```

```
Accuracy:  0.8208955223880597
Confusion matrix:
 [[ 81  26]
 [ 22 139]]
```

In [93]:
```
# compute accuracy on training set

logreg_train= logreg.score(X_train,y_train)

print("Training Data Accuracy by Logistics Regression Algorithm is : " ,logreg_train)

# compute accuracy on testing set

logreg_test= logreg.score(X_test,y_test)

print("Testing Data Accuracy by Logistics Regression is : " , logreg_test)
```

```
Training Data Accuracy by Logistics Regression Algorithm is :  0.8280373831775701
Testing Data Accuracy by Logistics Regression is :  0.8208955223880597
```

In [94]:
```
# Evaluate the model on the test data
score = logreg.score(X_test, y_test)
print("Accuracy of Logistic Regression is : ",score)
```

```
Accuracy of Logistic Regression is :  0.8208955223880597
```

In [95]:
```
# calculating the mean squared error
mse = np.mean((y_test - y_pred)**2, axis = None)
print("MSE :", mse)

# Calculating the root mean squared error
rmse = np.sqrt(mse)
print("RMSE :", rmse)
```

```
MSE : 0.1791044776119403
RMSE : 0.42320736951515897
```

Deciding a Model

In [96]:
```
# Usually We Select Only Those Model Whisch Has Highest Accuracy Among All those Prediction Results
```

In [100]:
```
if tuned_model_test > logreg_test :
    print (" For this Data Highest Accuracy belong to Decision Tree, out of 2 model with accuracy of  ",tuned_model_test)
else:
    print(" For this Highest Accuracy belong Data Logistics Regression, out of 2 model with accuracy of ",logreg_test)
```

```
 For this Highest Accuracy belong Data Logistics Regression, out of 4 model with accuracy of  0.8208955223880597
```

In [81]:

In [ ]:

In [ ]:

In [ ]: