Import Libraries

In [82]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
```

Data Collection & Analysis

In [83]:
```
insurance_data = pd.read_csv("insurance.csv")
insurance_data.head()
```

Out[83]:

	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 [84]:
```
# number of rows and columns
insurance_data.shape
```

Out[84]: (1338, 8)

In [85]:
```
# checking data points
print(insurance_data.size)
```

```
10704
```

In [5]:
```
# getting some informations about the data
insurance_data.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 [86]:
```
# Elucidation of data set
insurance_data.describe()
```

Out[86]:

	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 [87]:
```
# checking number of null value in the given data
insurance_data.isnull().sum()

```

Out[87]:
```
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 in the given data
insurance_data.isnull().any()
```

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

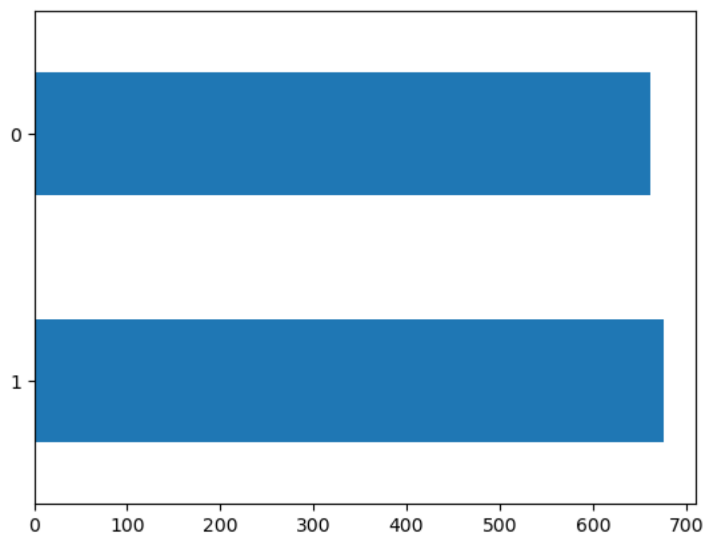
In [88]:
```
# checking value count of male and female in the given data
insurance_data['sex'].value_counts()

```

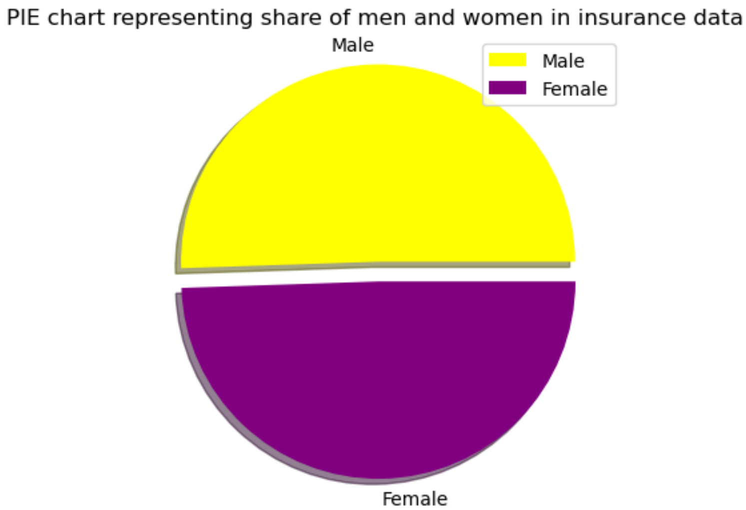
Out[88]:
```
1    676
0    662
Name: sex, dtype: int64
```

In [10]:
```
# Observation
# from the given data we can get the insights that :
# 1. data belongs to middle age people (mostly).
# 2. maximum age of person is 64 where as minimum age is 18.
# 3. maximum bmi is 53.13 which is a deep sign of obesity
# 4. there is no null value in the given data
# 5. There are 676 male and 662 female
```

In [89]:
```
# plotting a bar graph showing about number of male and female
insurance_data.sex.value_counts(normalize=False).plot.barh()
plt.show()
```



In [90]:
```
#pie chart: with Label and explode
# labels make a chart easier to understand because they show details about a data series or its individual data points
# To “explode” a pie chart means to make one of the wedges of the pie chart to stand out
mylables=["Male","Female"] # here label is "Male - is 1 where as Female - is 0"
colors = ['yellow', 'purple']
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()
```



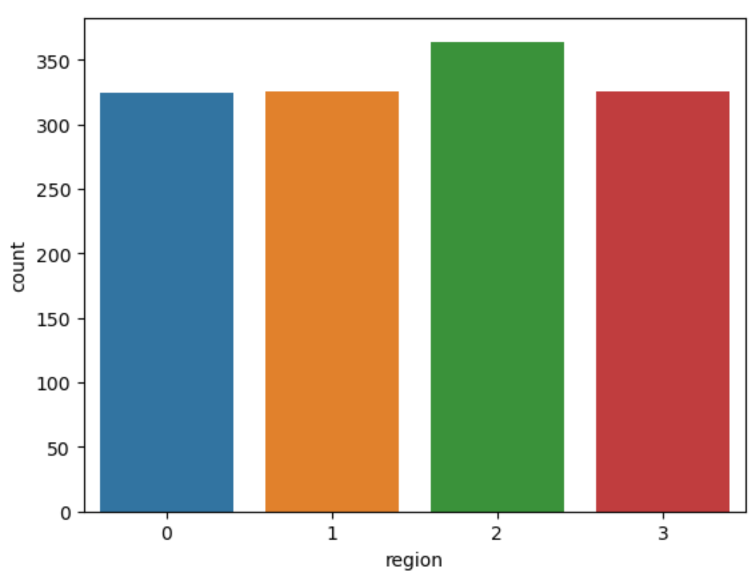
In [91]:
```
# checking customer belonging
insurance_data['region'].value_counts()

```

Out[91]:
```
2    364
3    325
1    325
0    324
Name: region, dtype: int64
```

In [92]:
```
#plotting a Countplot showing region
sns.countplot("region",data = insurance_data)
plt.show()
```

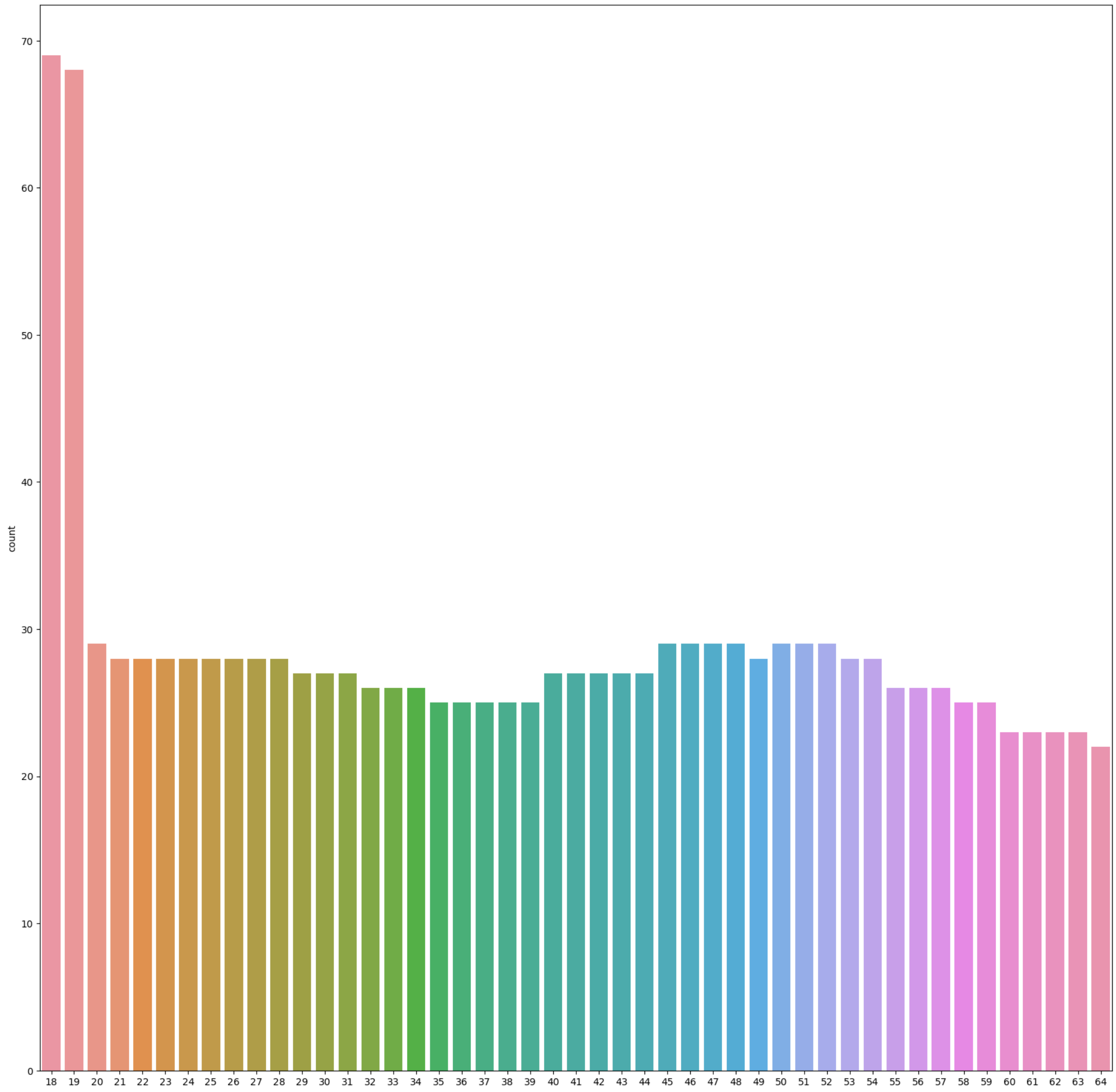
```
C:\Users\Aaditya\data science\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyw
ord arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explic
it keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [93]:

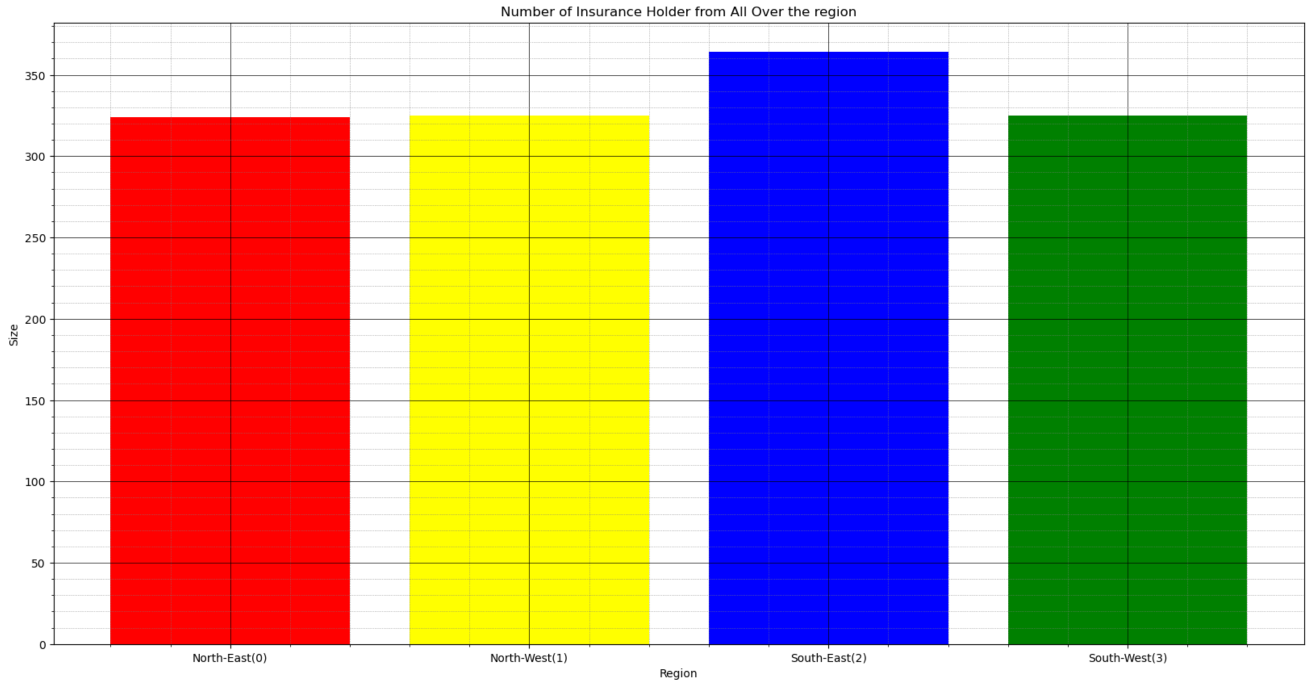
```
#plotting a countplot showing age
plt.figure(figsize = (20,20))
sns.countplot("age",data = insurance_data)
plt.show()
```

```
C:\Users\Aaditya\data science\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyw
ord arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explic
it keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [94]:

```
# plotting 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 = (20,10))
x_pos = [i for i, _ in enumerate(x)]
plt.bar(x_pos, size, color=['red', 'yellow', 'blue', 'green'])
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 [95]:

```
# 6.people belonging residential area from northeast(0) are 324 person ; northwest(1)
```

In [96]:

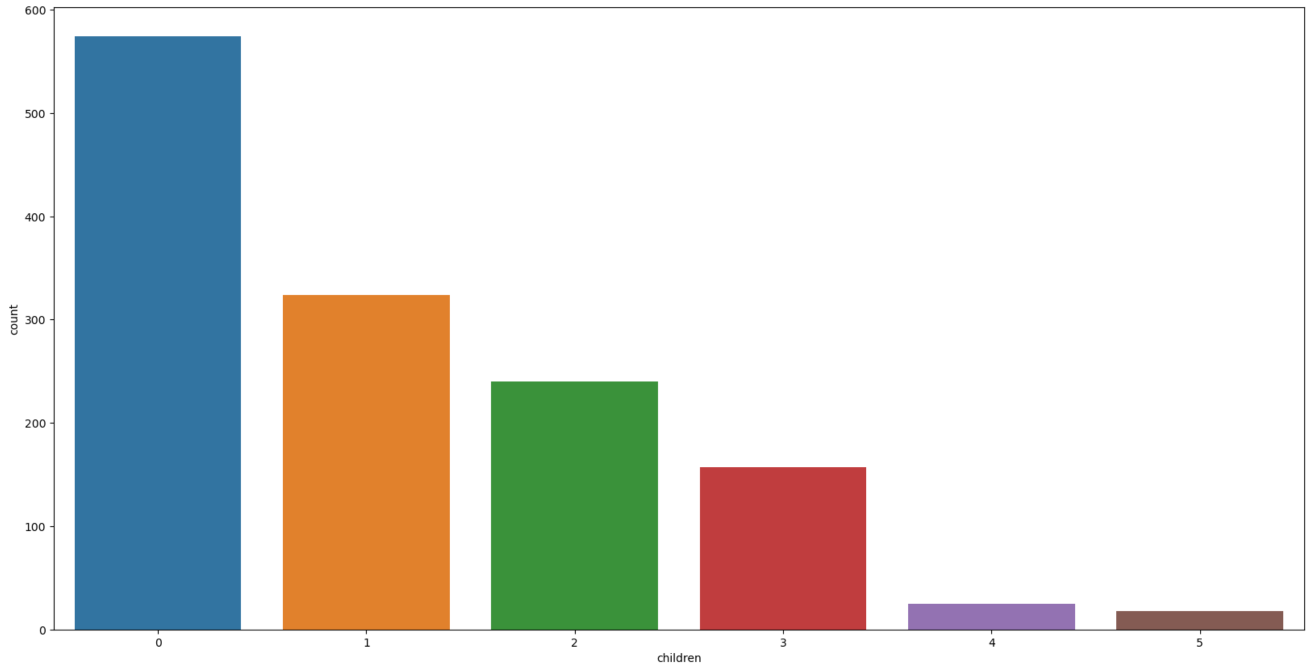
```
# checking children count
insurance_data['children'].value_counts()
```

Out[96]:

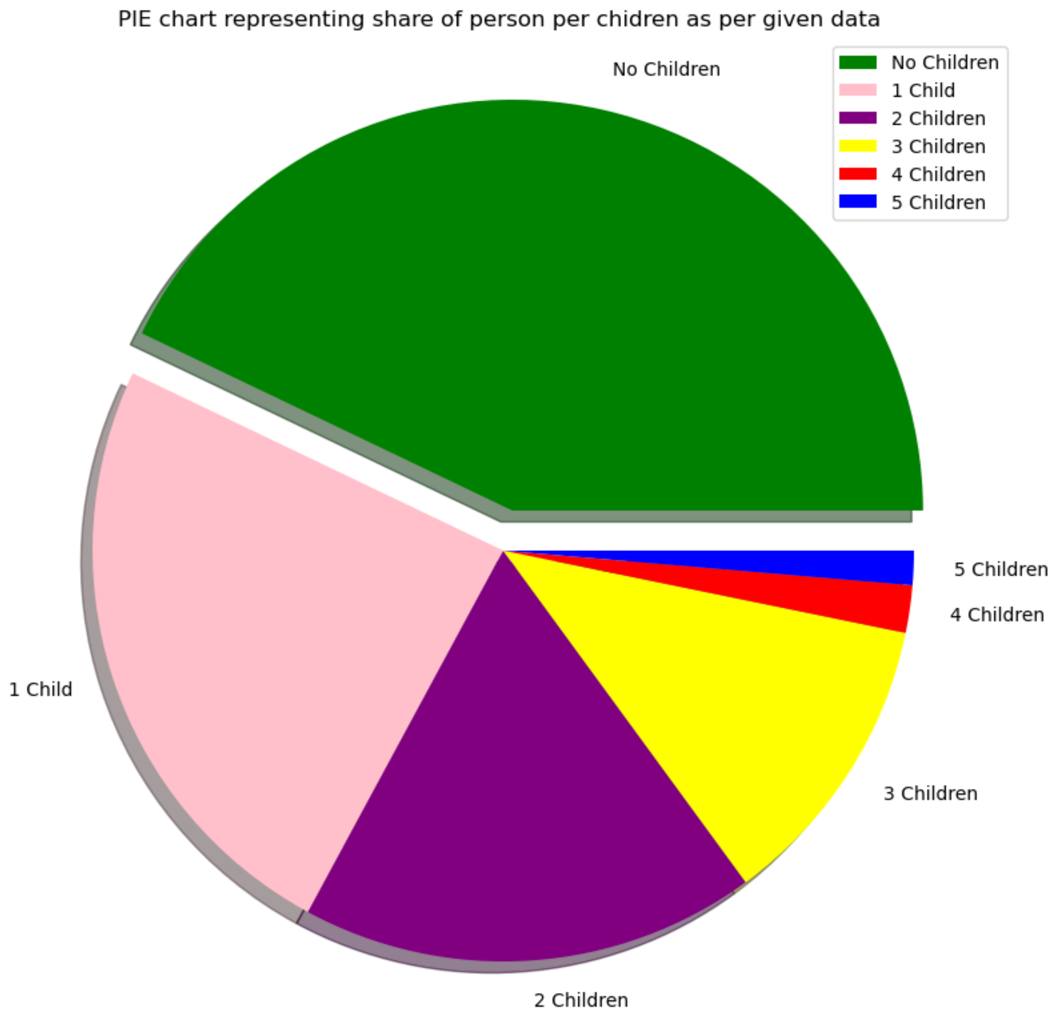
```
0    574
1    324
2    240
3    157
4     25
5     18
Name: children, dtype: int64
```

In [97]:
```
#plotting a countplot showing number of children
plt.figure(figsize = (20,10))
sns.countplot("children",data = insurance_data)
plt.show()
```

```
C:\Users\Aaditya\data science\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyw
ord arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explic
it keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [20]:
```
#pie chart: with Label and explode
plt.figure(figsize = (20,10))
mylables=["No Children","1 Child","2 Children","3 Children","4 Children","5 Children"]
colors = ['green','pink','purple','yellow','red','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()
```



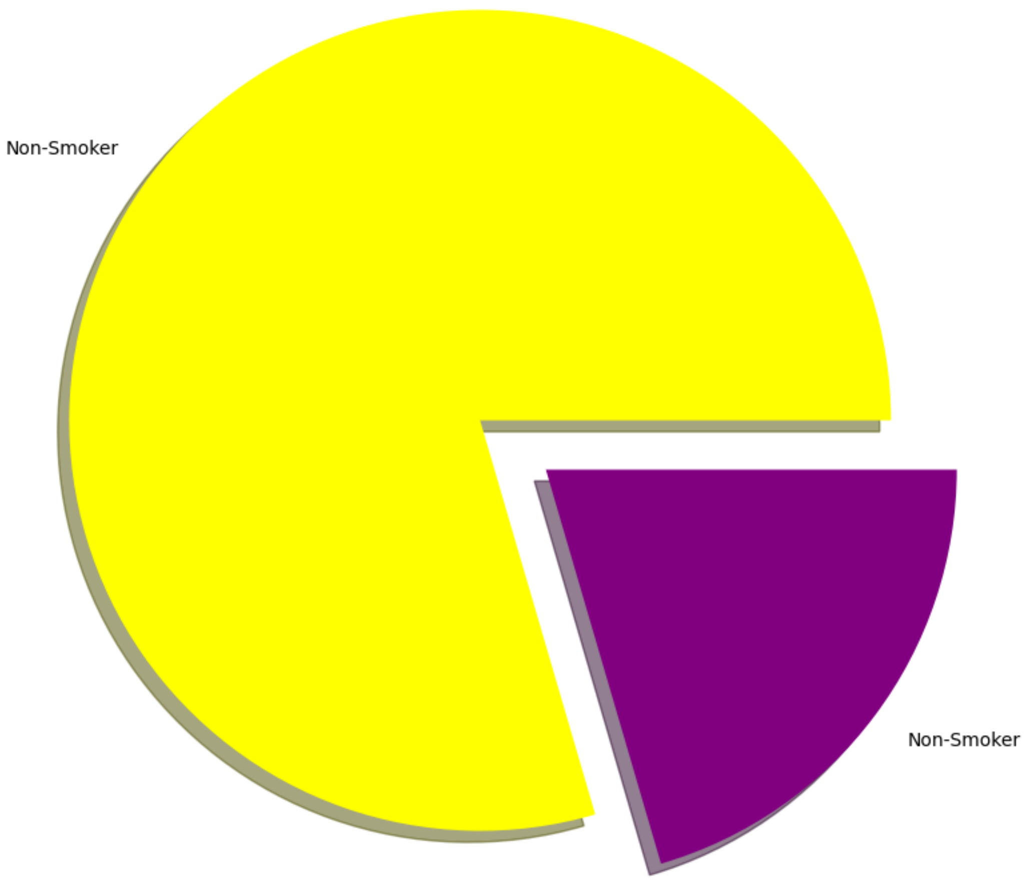
In [21]:
```
574 ; with 1 children are 324 ; with 2 chidren are 240 ; with 3 children are 157; with 4 children are 25; with 5 children are 18
```

In [98]:
```
# checking number of smokers
insurance_data['smoker'].value_counts()
```

Out[98]:
```
0    1064
1     274
Name: smoker, dtype: int64
```

In [23]:

```
#plotting a bar grap showing number of smoker
plt.figure(figsize = (20,10))
mylables=['Non-Smoker','Non-Smoker']
colors = ['yellow','purple']
myexplode=[0.10,0.10]
size = [1064,274]
plt.pie(size,colors = colors,labels =mylables,explode = myexplode, shadow = True)
plt.show()
```



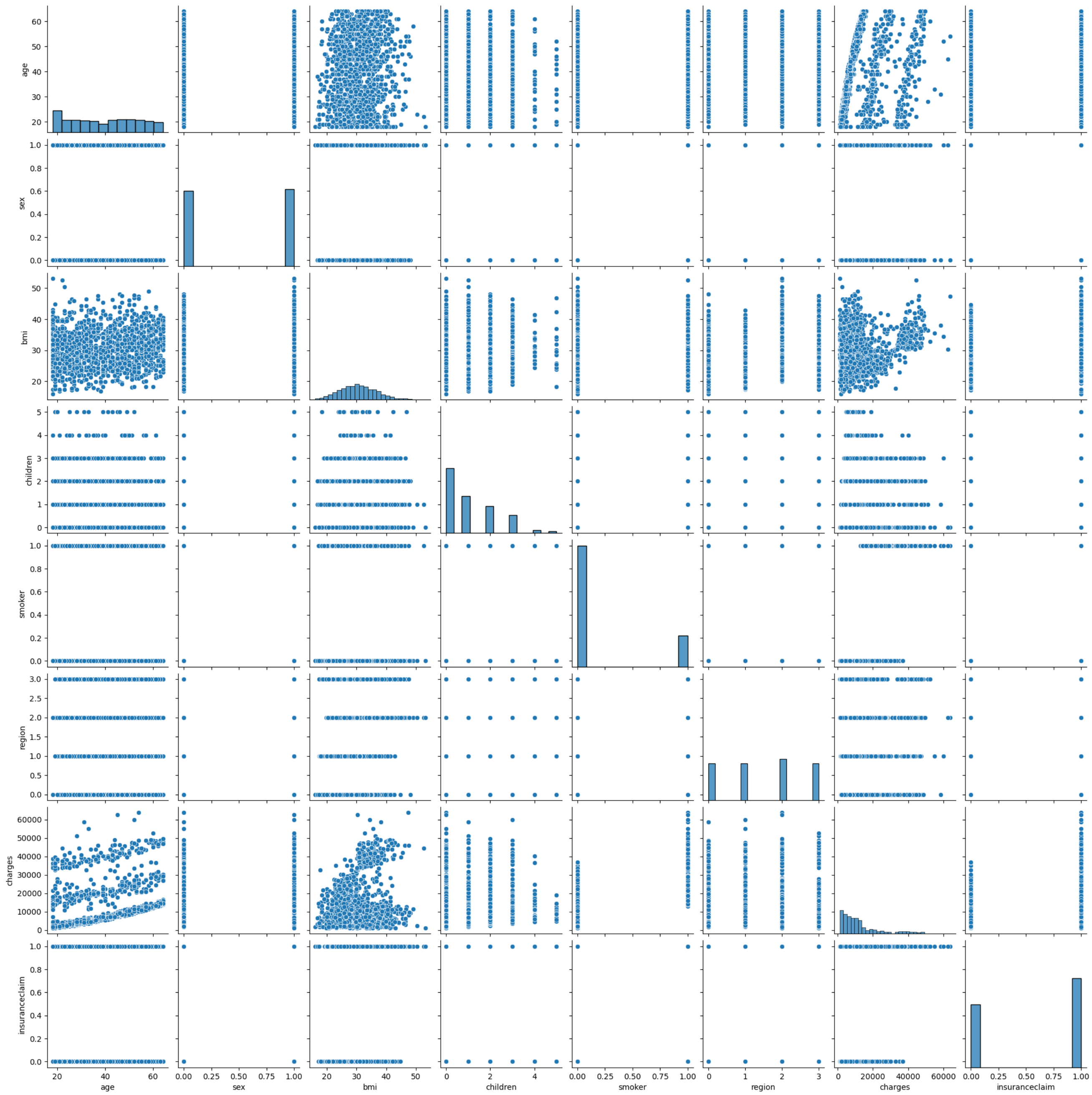
In [24]:

```
# 8. count of smokers are represented as 0 which is 1064 where as count of non-smokers are represented as 1 which is 274
```

In [99]:
```
# pairplot
plt.figure(figsize = (30,30))
sns.pairplot(insurance_data)
```

Out[99]: <seaborn.axisgrid.PairGrid at 0x29c6a6124c0>

<Figure size 3000x3000 with 0 Axes>



In [26]:
```
# Corelation Between Diffrent Features

insurance_data[['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges']].corr()

```

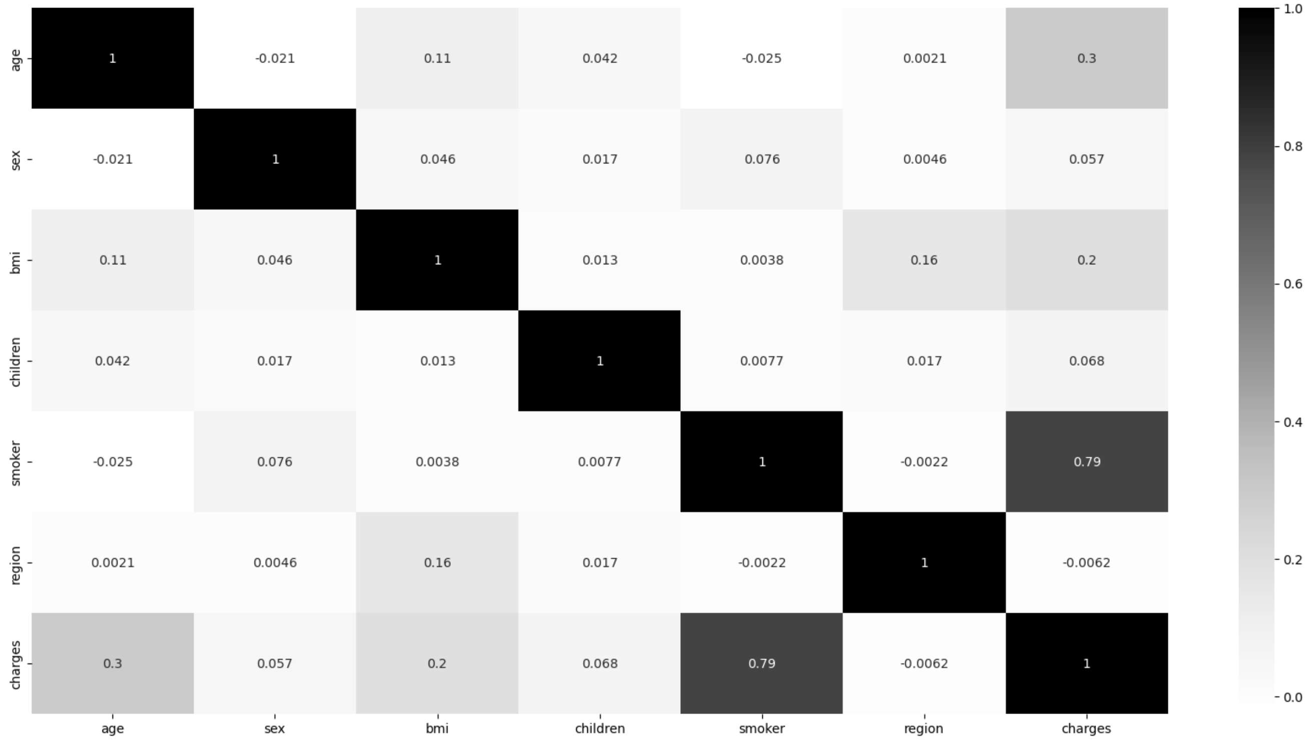
Out[26]:

	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]:
```
# Insights from this Corelation are :
# The table shows correlation coefficients between variables related to medical insurance charges. Age, BMI, and smoking stat
```

In [28]:
```
#plot the correlation matrix of salary, balance and age in data dataframe.
plt.figure(figsize = (20,10))
sns.heatmap(insurance_data[['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges']].corr(), annot=True,cmap = "Grey
plt.show()

```

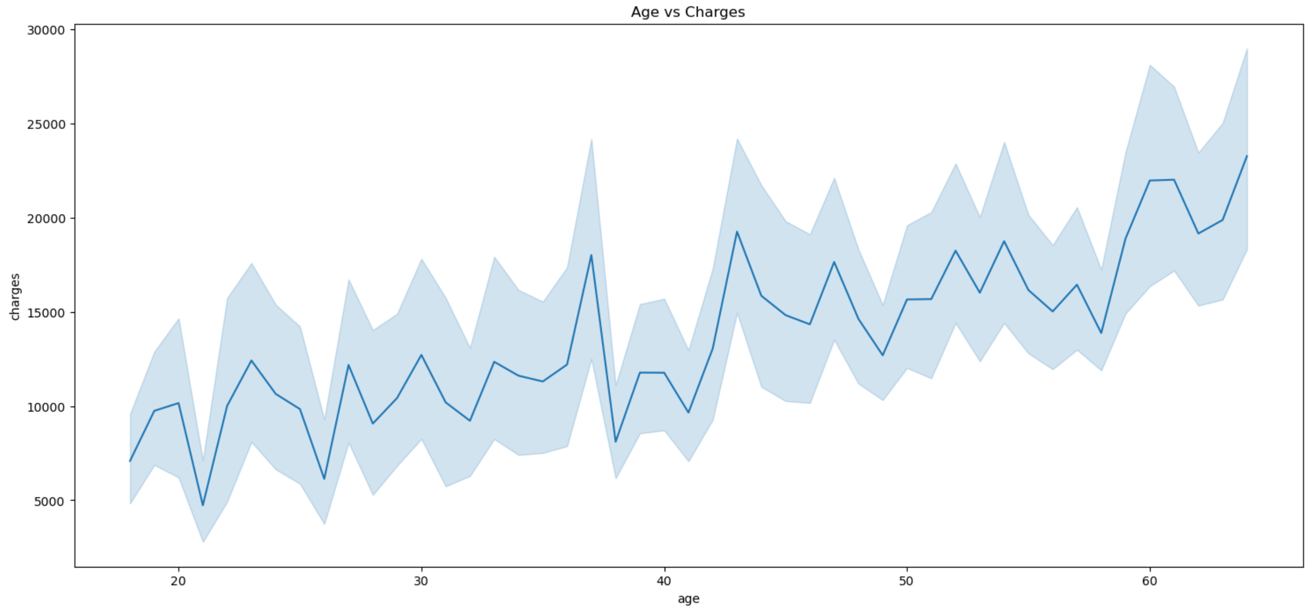


In [29]:
```
insurance_data.columns
```

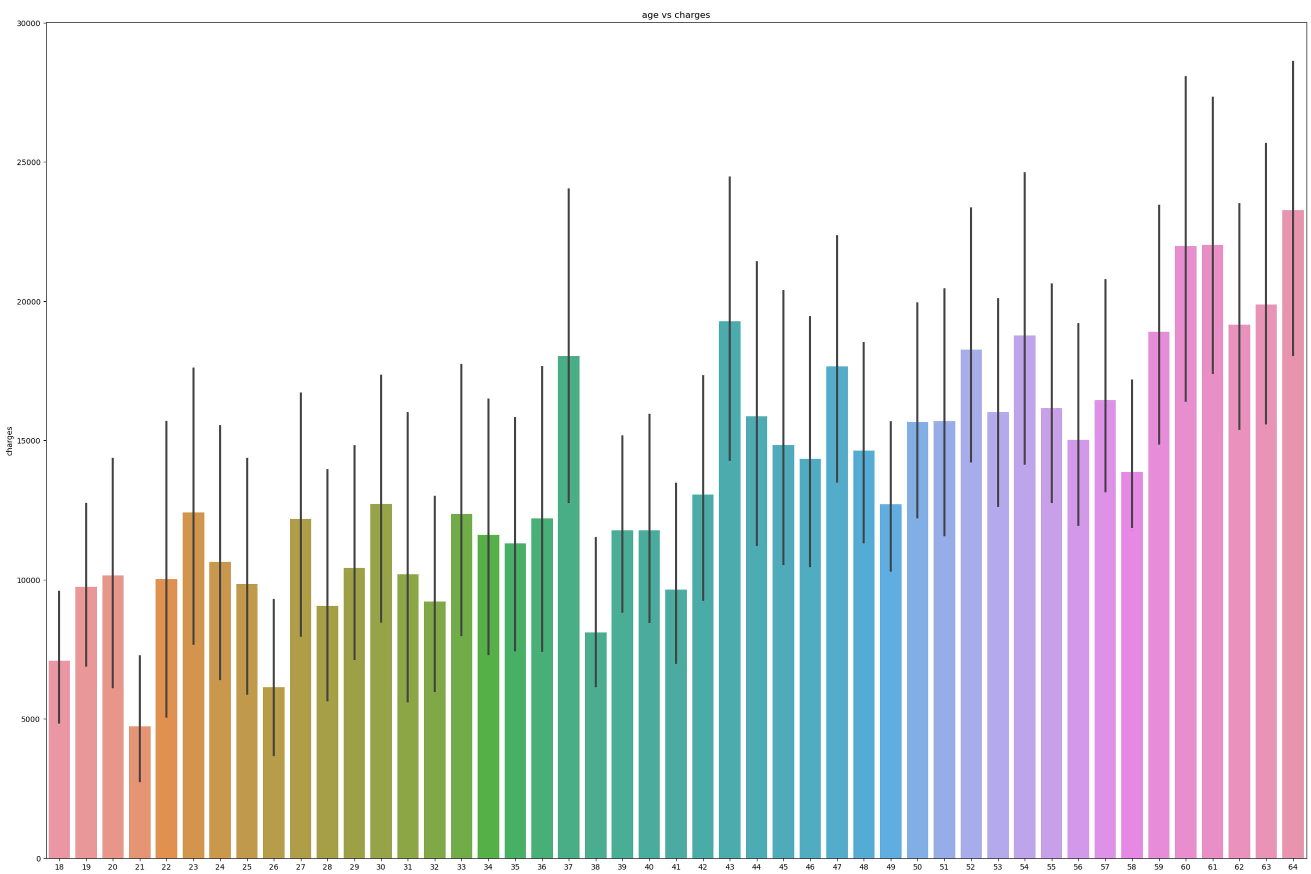
Out[29]:
```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges',
       'insuranceclaim'],
      dtype='object')
```

In [30]:
```
# Insigths From this Heat Map :-
# 1. Smoker Tends to Pay More Insurance Charges;
# 2. Age is positively correlated with charges, with a correlation coefficient of 0.30. This suggests that as age increases,
# 3.BMI is positively correlated with charges, with a correlation coefficient of 0.20. This suggests that as BMI increases,
# 4.The number of children does not appear to have a strong correlation with charges, with a correlation coefficient of 0.07
# 5.Sex and region also do not appear to have strong correlations with charges, with correlation coefficients of 0.06 and -0
```

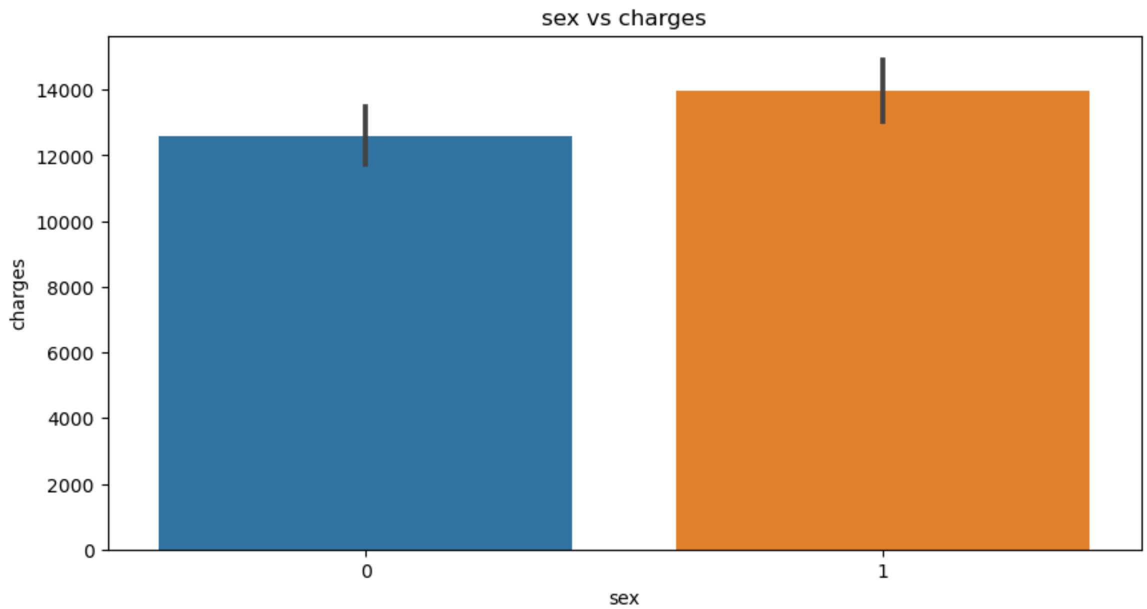
In [31]:
```
# Age vs Charges
# the more the age the more will be insurance charge (roughly estimated)
plt.figure(figsize = (18, 8))
sns.lineplot(x = 'age', y = 'charges', data = insurance_data)
plt.title("Age vs Charges")
plt.show()
```



In [32]:
```
#box plot for age vs charge
plt.figure(figsize = (30, 20))
sns.barplot(x = 'age', y = 'charges', data = insurance_data)
plt.title('age vs charges')
plt.show()
```

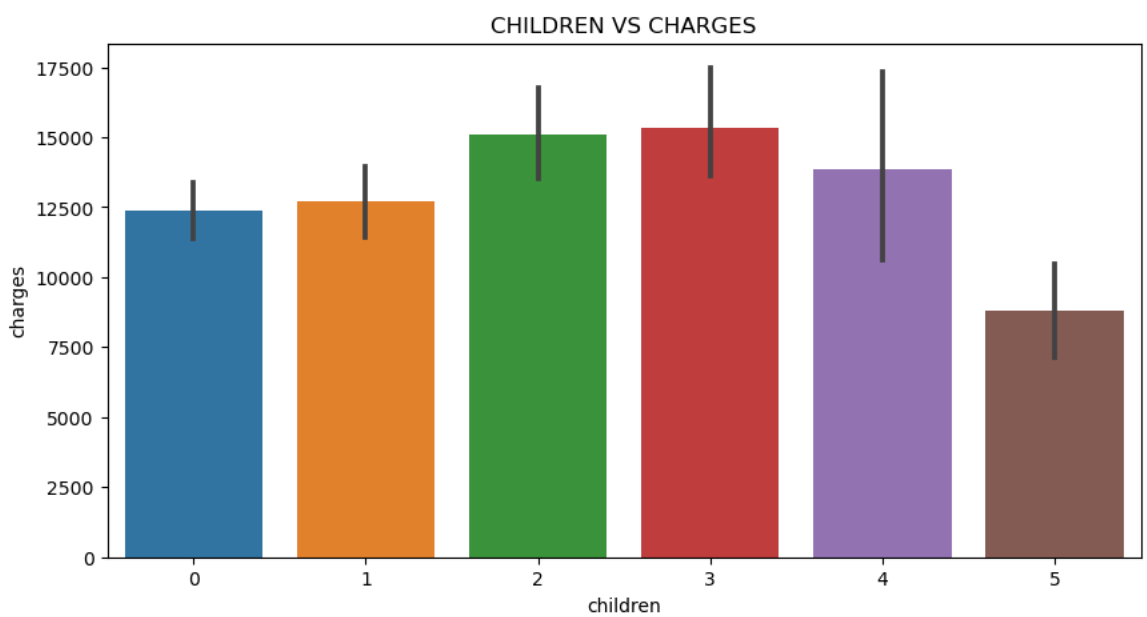


In [33]:
```
#plot the box plot of sex and charges
# as 1 belongs to men : it shows that men are paying more insurance charges then Women
#bar plot
plt.figure(figsize = (10, 5))
sns.barplot(x = 'sex', y = 'charges', data = insurance_data)
plt.title('sex vs charges')
plt.show()
```

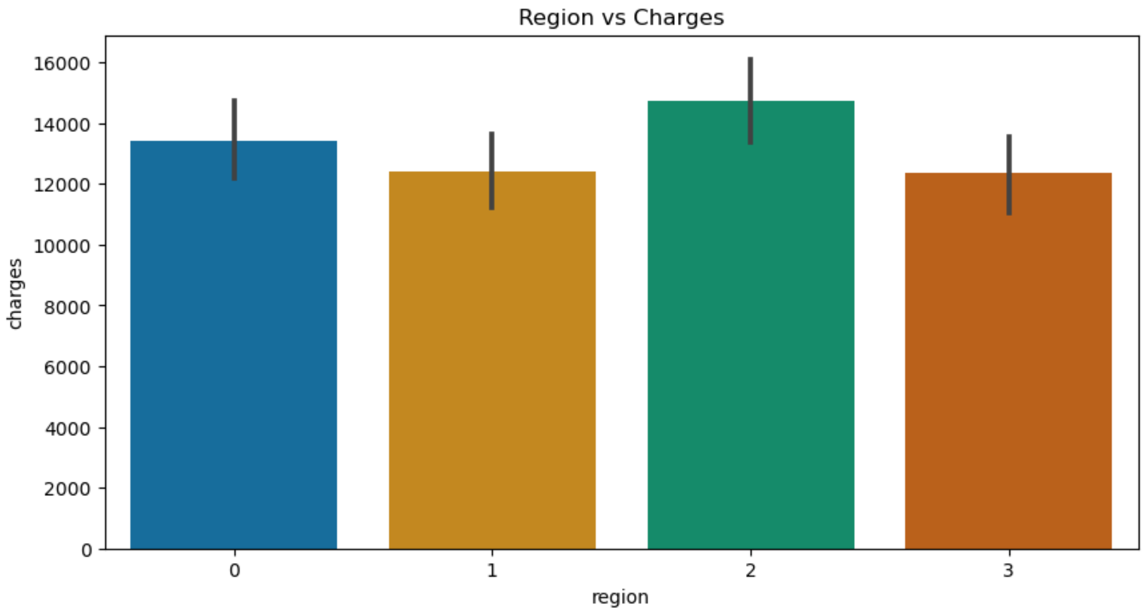


In [34]:
```
# children vs charges
# no. of childrens of a person has a weird dependency on insurance charge. i.e(parents of more children tends to pay less in
plt.figure(figsize = (10, 5))
sns.barplot(x = 'children', y = 'charges', data = insurance_data)
plt.title('CHILDREN VS CHARGES')
plt.show()

```

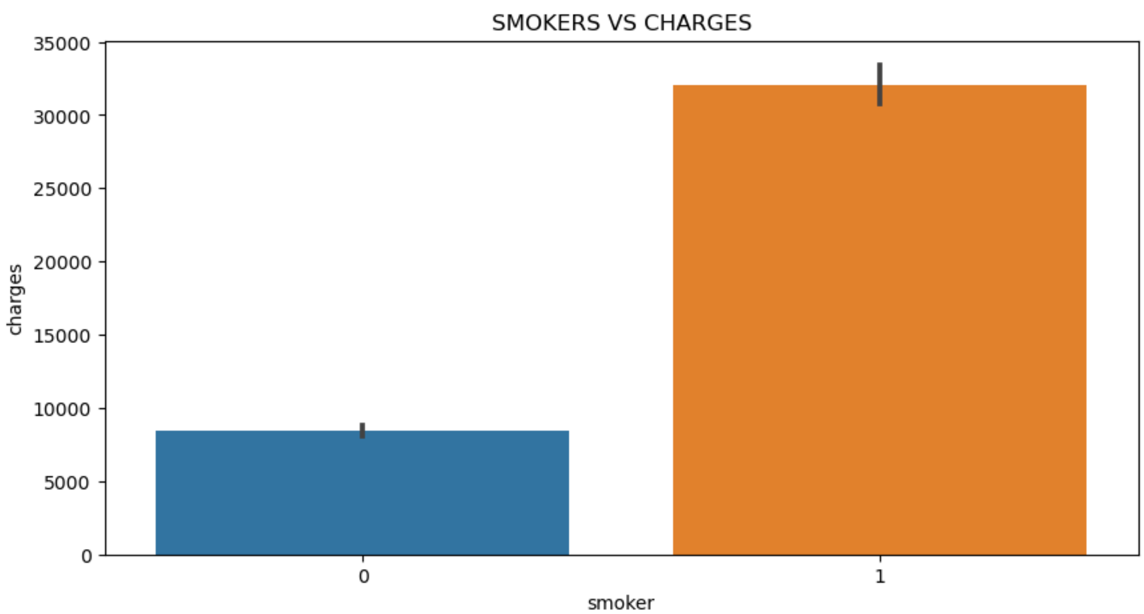


In [35]:
```
# region vs charges BAR GRAPh
plt.figure(figsize = (10, 5))
sns.barplot(x = 'region', y = 'charges', data = insurance_data, palette = 'colorblind')
plt.title('Region vs Charges')
plt.show()
```



In [36]:
```
# from the graph we can clearly state that region dont play any role in charges it is highly independent should be drop
```

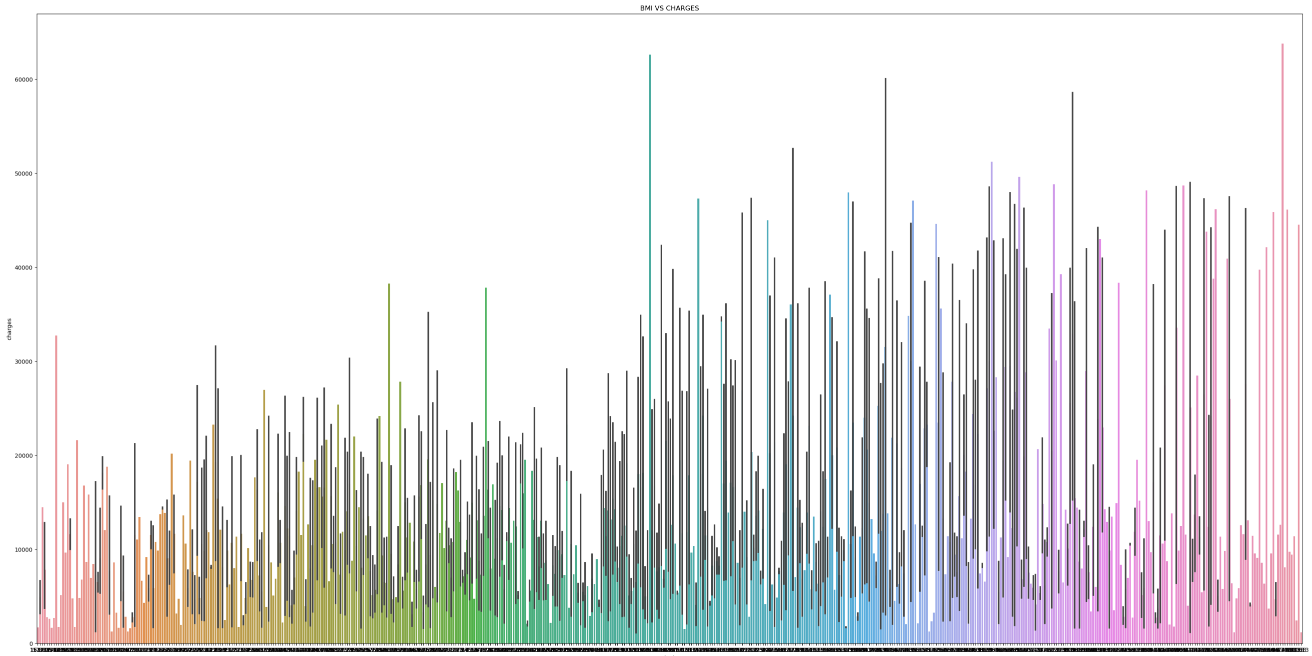
In [37]:
```
# smoker vs charges
plt.figure(figsize = (10, 5))
sns.barplot(x = 'smoker', y = 'charges', data = insurance_data)
plt.title('SMOKERS VS CHARGES')
plt.show()
```



In [ ]:
```

```

In [38]:
```
# BMI vs charges
plt.figure(figsize = (40,20))
sns.barplot(x = 'bmi', y = 'charges', data = insurance_data)
plt.title('BMI VS CHARGES')
plt.show()
```



In [ ]:
```

```

Data Cleaning

In [39]:
```
# removing unrequired columns from the insurance data
# As from the above grph we can clearly state that region dont play any role in charge it is highly independent and should b
insurance_data = insurance_data.drop('region', axis = 1)

```

In [40]:
```
insurance_data.shape
```

Out[40]: (1338, 7)

In [41]:
```
#as earlier there was 10704 data point the new one has 9366 data point after removing region
insurance_data.size
```

Out[41]: 9366

In [42]:
```
# seperate out features and target value from dataset
X=insurance_data.drop(["insuranceclaim"],axis=1).values
y=insurance_data["insuranceclaim"].values
```

In [43]:
```
X.shape
```

Out[43]: (1338, 6)

In [44]:
```
y.shape
```

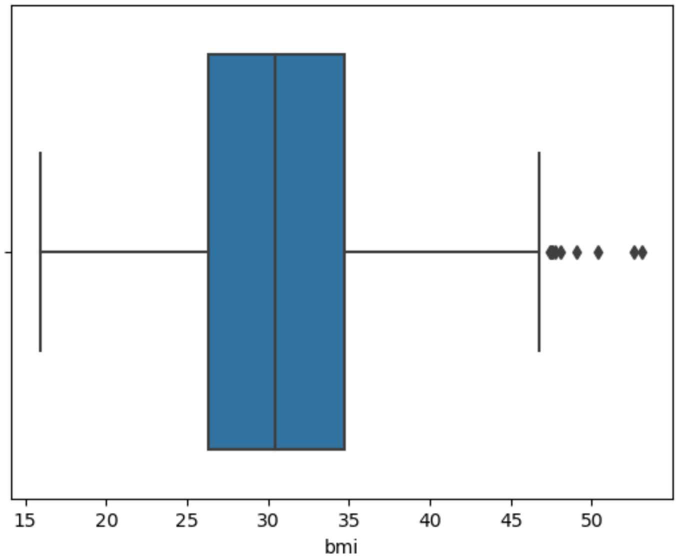
Out[44]: (1338,)

Finding out the outlier

In [45]:
```
#bmi outlier
sns.boxplot(insurance_data["bmi"])
plt.show()

```

```
C:\Users\Aaditya\data science\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyw
ord arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explic
it keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [46]:
```
# Finding Position of Outlier
#position plot of outlier
print(np.where(insurance_data["bmi"]>45))
```

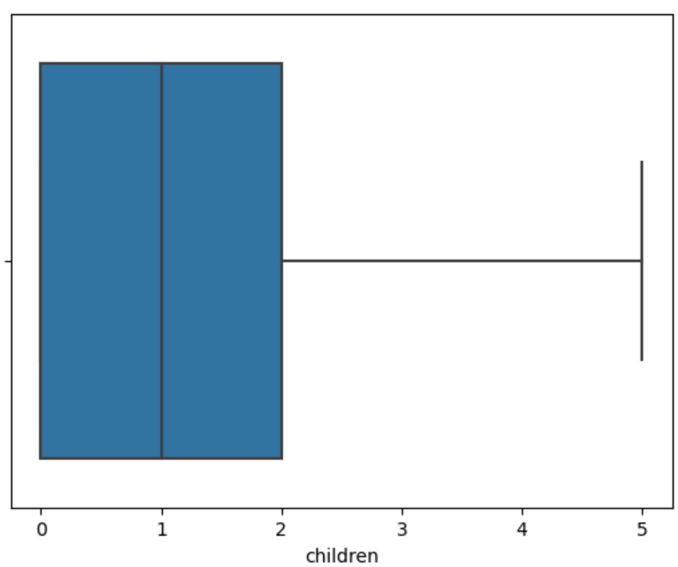
```
(array([ 116,  286,  292,  401,  438,  454,  543,  547,  549,  582,  660,
        847,  860,  930,  941, 1024, 1047, 1088, 1131, 1317], dtype=int64),)
```

In [ ]:
```

```

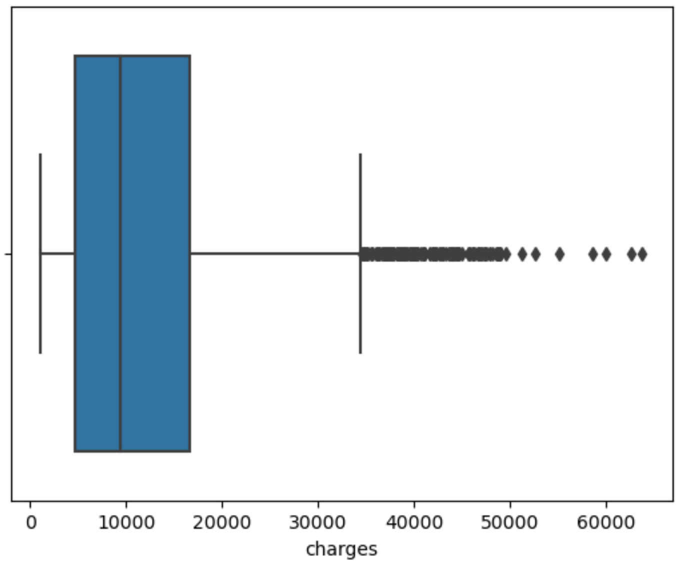
In [47]:
```
#Children outlier
sns.boxplot(insurance_data["children"])
plt.show()
```

```
C:\Users\Aaditya\data science\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyw
ord arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explic
it keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [48]:
```
#Charges outlier
sns.boxplot(insurance_data["charges"])
plt.show()
```

```
C:\Users\Aaditya\data science\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyw
ord arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explic
it keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [49]:
```

# Charges can be More or less as per required by insurance company

```

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

In [ ]:
```
#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 = 0.27, random_state =4)
```

In [ ]:
```
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)
```

Importing and Using Logistic Regression

In [52]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import LogisticRegression

```

In [53]:
```
# Logistics Regression model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

```

```
C:\Users\Aaditya\data science\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to conv
erge (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/line
ar_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[53]: LogisticRegression()

In [54]:
```
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)
print("Where ; True Positive: is 104; False Positive: is 40; True Negative : is 24; False Negative is 189")

```

```
Accuracy:  0.8646408839779005
Confusion matrix:
 [[124  24]
 [ 25 189]]
Where ; True Positive: is 104; False Positive: is 40; True Negative : is 24; False Negative is 189
```

In [ ]:
```

```

In [55]:
```
# 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 :  0.8442622950819673
Testing Data Accuracy by Logistics Regression is :  0.8646408839779005
```

In [56]:
```
# 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 :  0.8646408839779005
```

In [57]:
```
# 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 : 0.13535911602209943
RMSE : 0.3679118318593457
```

In [ ]:
```

```

Importing and Using Random Forest Method with same test size

In [58]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [59]:
```
# model = RF Random Forest
rf = RandomForestClassifier(n_estimators=1000,random_state=45)

#fitting model
rf.fit(X_train,y_train)
```

Out[59]:
```
RandomForestClassifier(n_estimators=1000, random_state=45)
```

In [60]:
```
#predicting

y_pred=rf.predict(X_test)

y_pred
```

Out[60]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 0, 0, 1], dtype=int64)
```

In [61]:
```
# compute accuracy on training set

rf_train= rf.score(X_train,y_train)
print("Training Data Accuracy by Random Forest Algorithm is : " , rf_train)
# compute accuracy on testing set
rf_test= rf.score(X_test,y_test)
print("Testing Data Accuracy by Random Forest Algorithm is : " , rf_test)
```

```
Training Data Accuracy by Random Forest Algorithm is :  1.0
Testing Data Accuracy by Random Forest Algorithm is :  0.9613259668508287
```

In [ ]:
```

```

In [62]:
```
# calculating the mean squared error
mse = np.mean((y_test - y_pred)**2, axis = None)
print("MSE :", mse)
```

```
MSE : 0.03867403314917127
```

In [63]:
```
# Calculating the root mean squared error
rmse = np.sqrt(mse)
print("RMSE :", rmse)
```

```
RMSE : 0.19665714619400756
```

Importing and Using Decision Tree (Supervised Learning) Algorithm

In [64]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [65]:
```
# model
dtc = DecisionTreeClassifier()
#fitting
dtc.fit(X_train,y_train)
```

Out[65]:
```
DecisionTreeClassifier()
```

In [66]:
```
#predicting via Decision Tree Algorithm
y_pred=dtc.predict(X_test)
y_pred
```

Out[66]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 0, 0, 1], dtype=int64)
```

In [67]:
```
#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 =  0.23505024736113422
```

In [68]:

```
y_pred_df=pd.DataFrame(y_pred)
y_pred_df
```

Out[68]:

	0
0	0
1	0
2	0
3	0
4	0
...	...
357	1
358	1
359	0
360	0
361	1

362 rows × 1 columns

In [69]:

```
y_pred_df["Actual"]=y_test
y_pred_df
```

Out[69]:

	0	Actual
0	0	0
1	0	0
2	0	0
3	0	1
4	0	0
...	...	...
357	1	1
358	1	1
359	0	0
360	0	0
361	1	1

362 rows × 2 columns

In [70]:

```

y_pred_df.columns=["Predicated","Actual"]
y_pred_df
```

Out[70]:

	Predicated	Actual
0	0	0
1	0	0
2	0	0
3	0	1
4	0	0
...	...	...
357	1	1
358	1	1
359	0	0
360	0	0
361	1	1

362 rows × 2 columns

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

In [71]:
```
# 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 :  0.9447513812154696
```

In [72]:
```
# 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 : 0.055248618784530384
RMSE : 0.23505024736113422
```

Importing and Using Naive Bayes Method

In [73]:
```
from sklearn.naive_bayes import GaussianNB
```

In [74]:
```
#model naive Bayes
nb = GaussianNB()
nb.fit(X_train,y_train)

```

Out[74]:
```
GaussianNB()
```

In [75]:
```
print("Naive Bayes Score : ",nb.score(X_test,y_test))
```

```
Naive Bayes Score :  0.8011049723756906
```

In [76]:
```

#prediction
y_pred= nb.predict(X_test)
print(y_pred)
```

```
[0 0 0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 0
 1 0 1 1 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 1 0 0 1 0 0 0 0 0
 1 0 0 0 1 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1
 0 0 0 1 0 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 1 1 1 1
 0 0 0 1 0 0 1 1 1 0 0 1 0 0 0 1 0 1 1 1 0 1 0 1 0 0 0 1 1 0 0 1 0 0 0 0 1
 0 1 0 1 0 0 1 0 1 0 0 1 0 1 0 1 1 0 1 0 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 1 0
 0 0 1 1 0 1 1 0 0 1 0 1 0 0 0 0 1 1 1 1 1 0 0 1 0 0 1 0 0 1 0 1 1 0 1 0 0
 1 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 0 1 1 1 0 0 0 0 1 1
 0 1 1 0 0 1 0 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 0 0 0 1 0 1 0 0 0 1 0 0
 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 0 1 1 1 1 0 0 1 0 0 1]
```

In [77]:
```
y_pred.size
```

Out[77]:
```
362
```

In [78]:
```
# compute accuracy on training set
nb_train= nb.score(X_train,y_train)
print("Training Data Accuracy by Random Forest Algorithm is : " ,nb_train)
# compute accuracy on testing set
nb_test= nb.score(X_test,y_test)
print("Testing Data Accuracy by Random Forest Algorithm is : " ,nb_test)
```

```
Training Data Accuracy by Random Forest Algorithm is :  0.7838114754098361
Testing Data Accuracy by Random Forest Algorithm is :  0.8011049723756906
```

In [79]:
```
# 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 : 0.19889502762430938
RMSE : 0.4459764877482998
```

Deciding a Model

In [80]:
```
# Genrally we select only those model which has highest accuracy among all the prediction result

```

In [101]:
```
if dtc_test > rf_test:
 print (" For this Data Highest Accuracy belong to Decision Tree, out of 4 model ")
elif rf_test > nb_test:
 print (" For this Highest Accuracy belong Data Random Forest, out of 4 model ")
elif nb_test> logreg_test:
 print(" For this Highest Accuracy belong Data Naive Bayes classifier, out of 4 model ")
elif logreg_test>dtc_test:
 print(" For this Highest Accuracy belong Data Logistics Regression, out of 4 model ")
```

```
 For this Highest Accuracy belong Data Random Forest, out of 4 model
```

In [ ]:
```

```