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	char
count	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000
mean	39.207025	0.505232	30.663397	1.094918	0.204783	1.515695	13270.422
std	14.049960	0.500160	6.098187	1.205493	0.403694	1.104885	12110.011
min	18.000000	0.000000	15.960000	0.000000	0.000000	0.000000	1121.873
25%	27.000000	0.000000	26.296250	0.000000	0.000000	1.000000	4740.287
50%	39.000000	1.000000	30.400000	1.000000	0.000000	2.000000	9382.033
75%	51.000000	1.000000	34.693750	2.000000	0.000000	2.000000	16639.912
max	64.000000	1.000000	53.130000	5.000000	1.000000	3.000000	63770.428

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]:
```python
# checking if any null value is present or not
df.isnull().any()
```

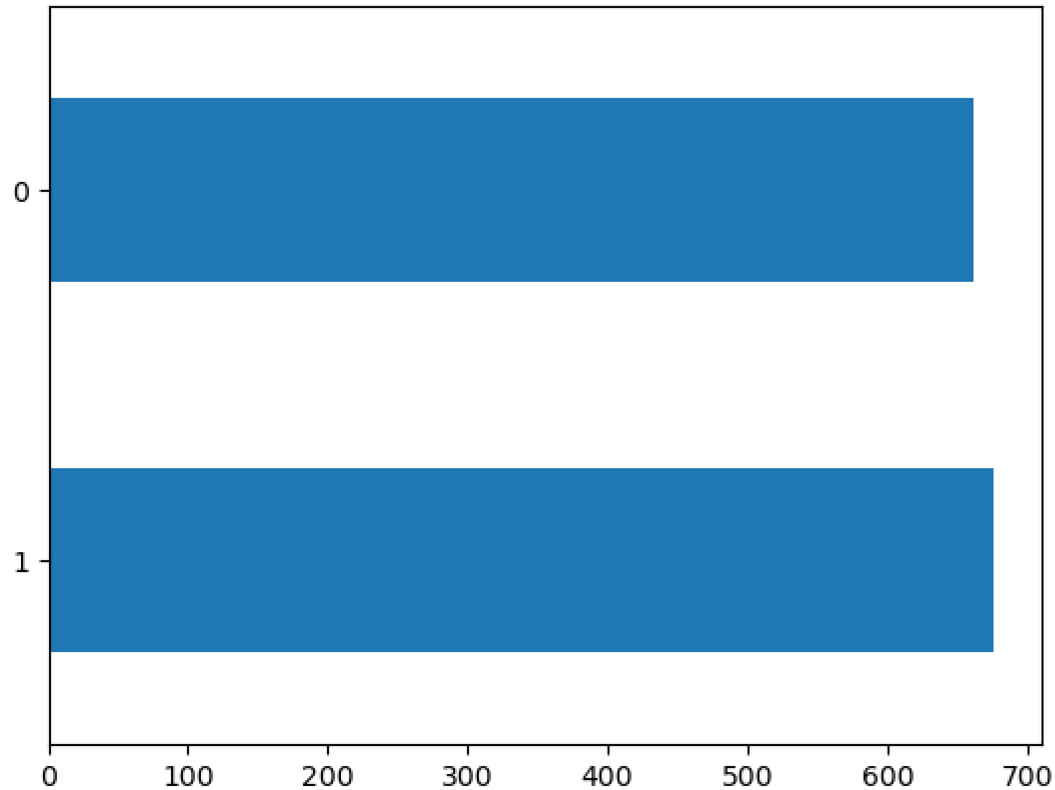
Out[8]:
```
age               False
sex               False
bmi               False
children          False
smoker            False
region            False
charges           False
insuranceclaim    False
dtype: bool
```

In [9]:
```python
# 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
```

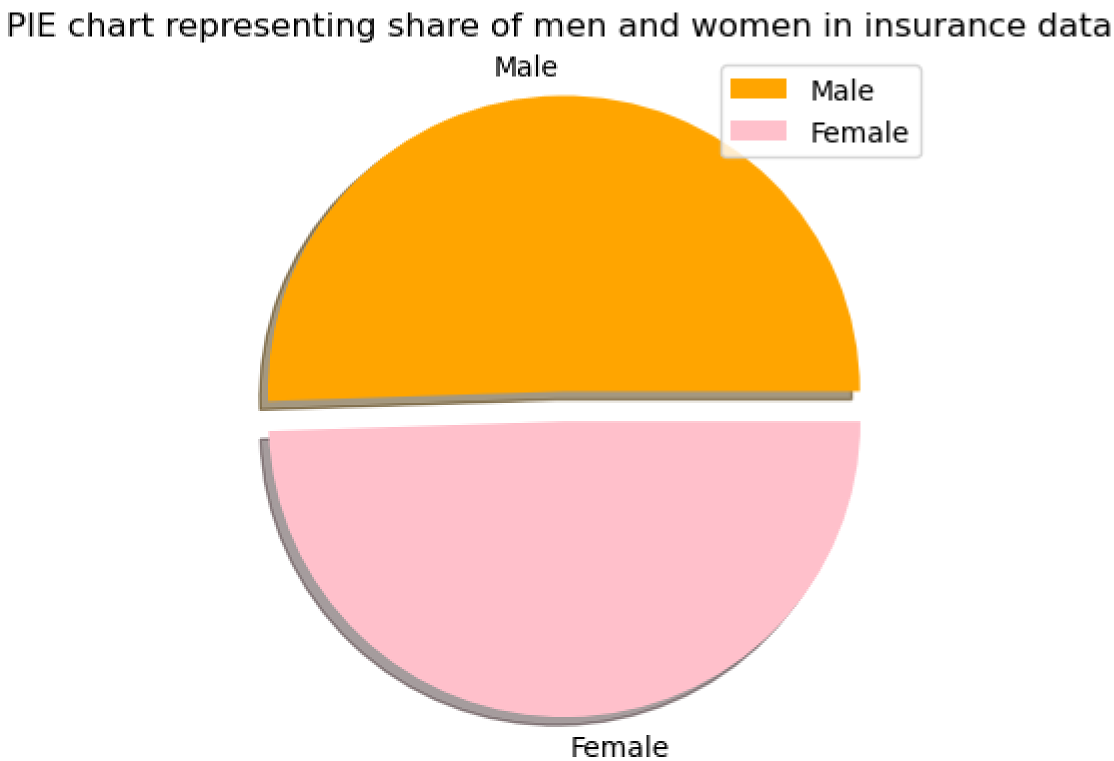
In [10]:
```python
# checking value count of male and female in data
df['sex'].value_counts()
```

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 = Tr
plt.title('PIE chart representing share of men and women in insurance data ')
plt.legend()
plt.show()
```

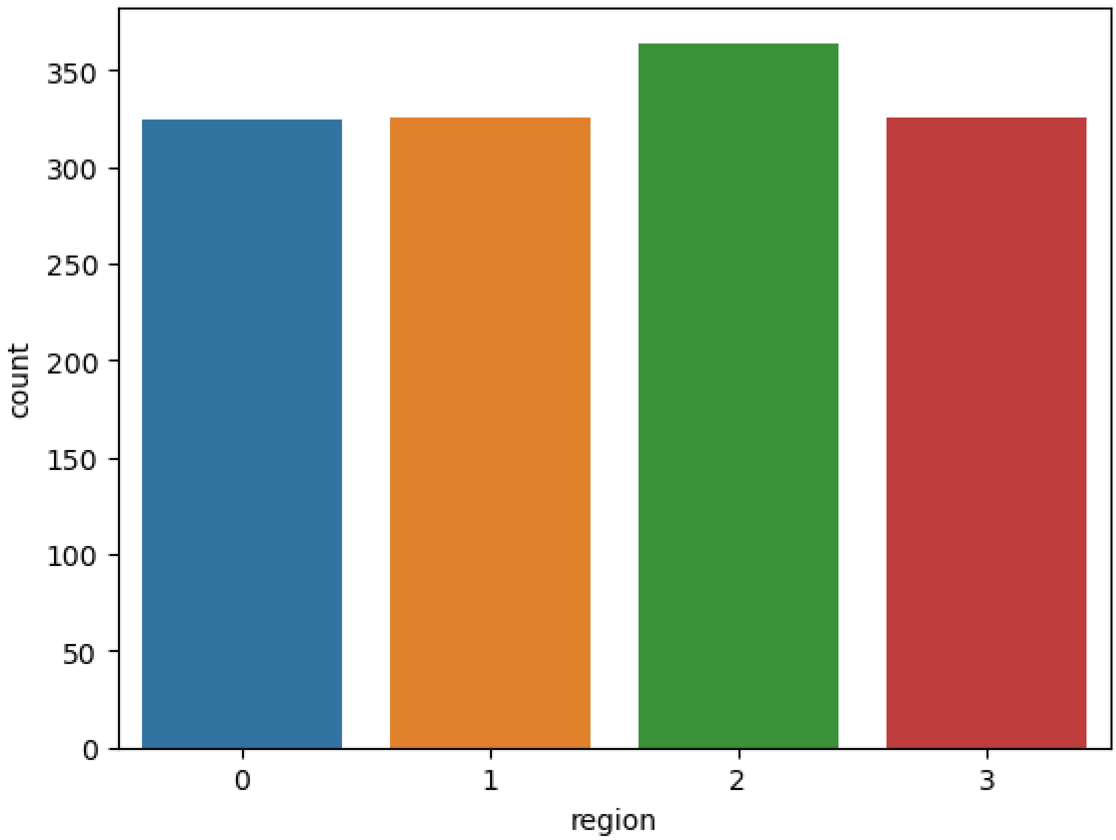


In [13]:
```
# checking customer belonging
df['region'].value_counts()
```

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

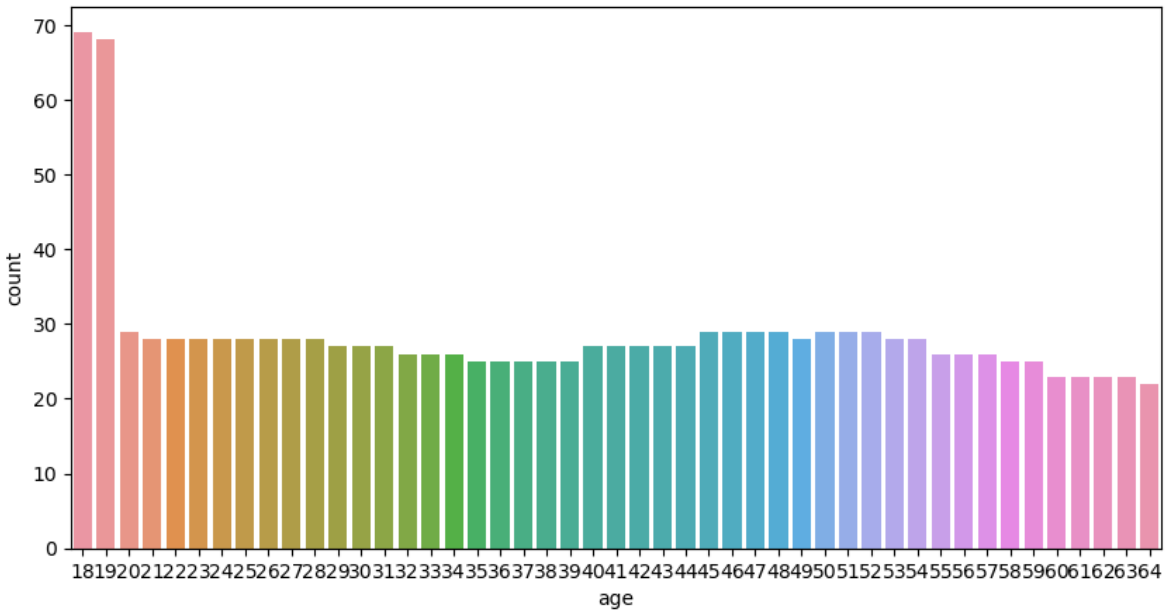
In [14]:
```
#ploting a Countplot showing region
sns.countplot("region",data = df)
plt.show()
```

```
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 argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

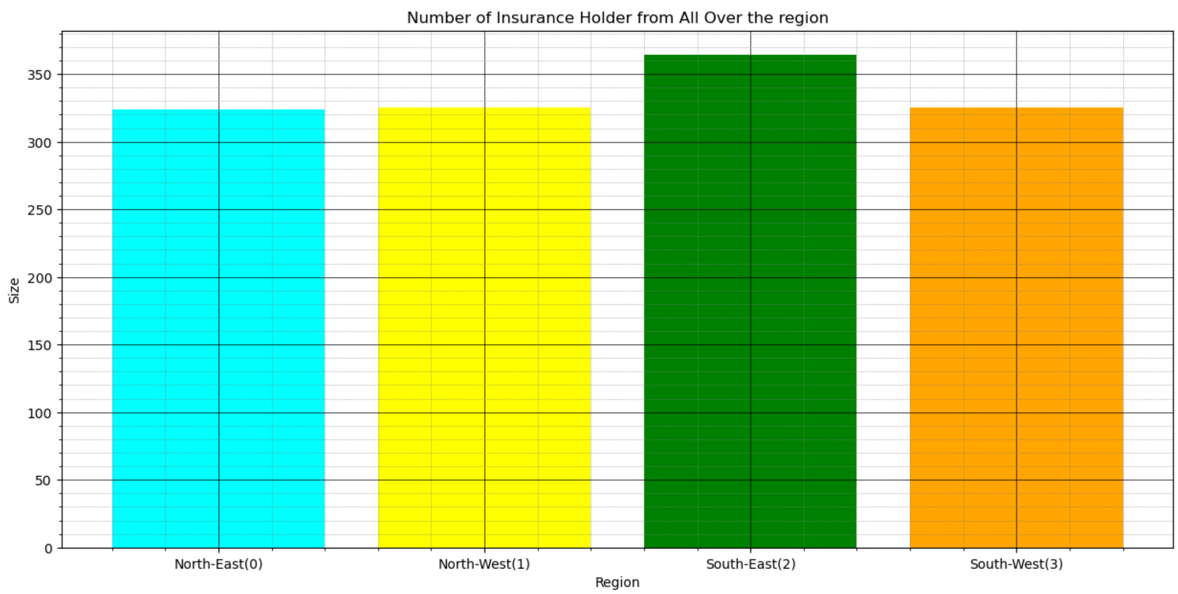


In [15]:
```
#ploting a Countplot showing age
plt.figure(figsize = (10,5))
sns.countplot("age",data = df)
plt.show()
```

```
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 argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



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

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()
```



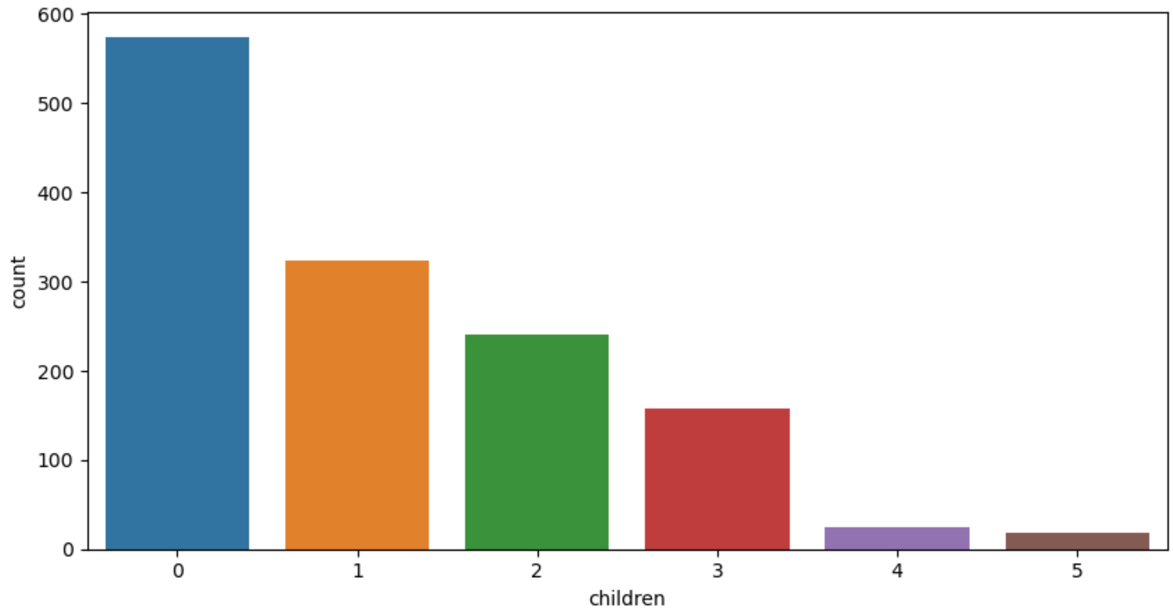
In [17]:
```
# 6.people belonging residential area from  northeast(0) are 324 person ; nort
```

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

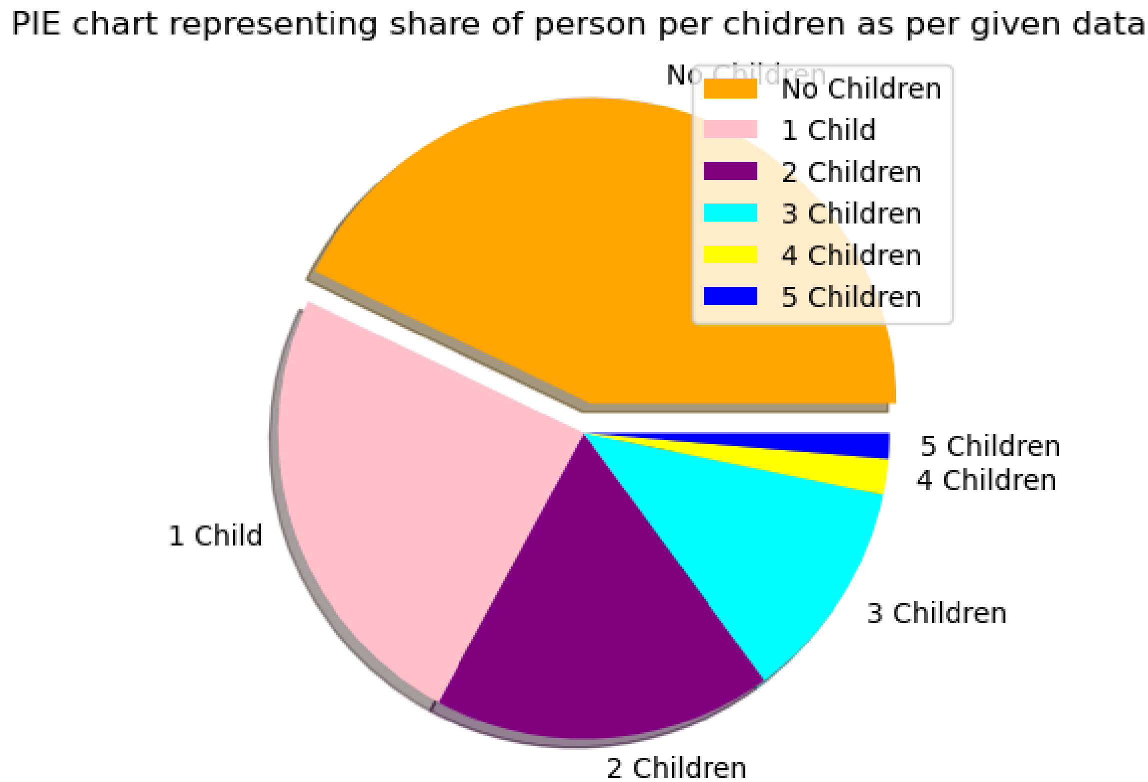
Out[18]:
```
0    574
1    324
2    240
3    157
4     25
5     18
Name: children, dtype: int64
```

In [19]:
```
#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 argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [20]:
```
#pie chart: with Label and explode
plt.figure(figsize = (10,5))
mylables=["No Children","1 Child","2 Children","3 Children","4 Children","5 Ch
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 = Tr
plt.title('PIE chart representing share of person per chidren as per given dat
plt.legend()
plt.show()
```

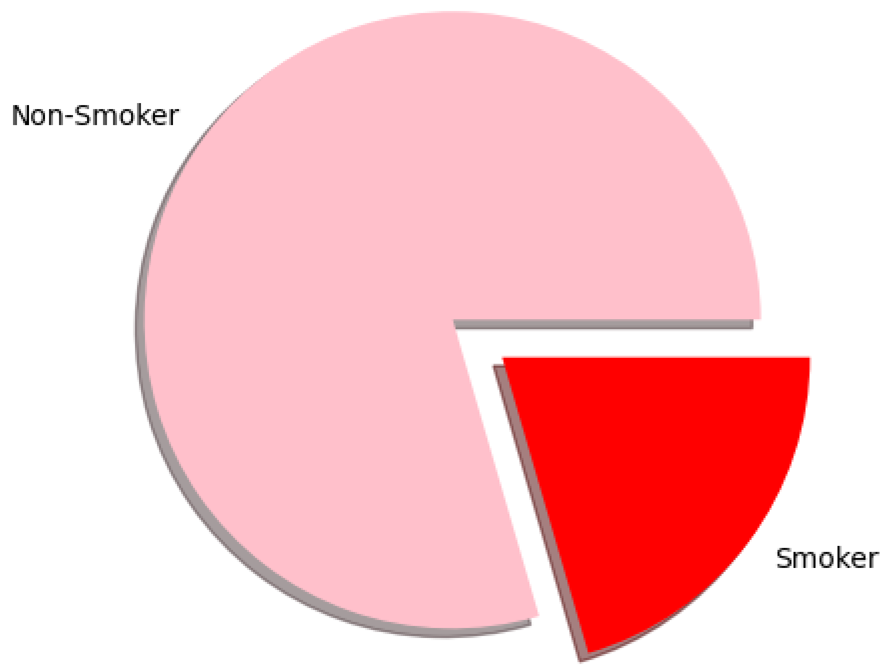


In [21]:
```
# 7. count of people having no children are 574 ; with 1 children are 324 ; wi
```

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

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

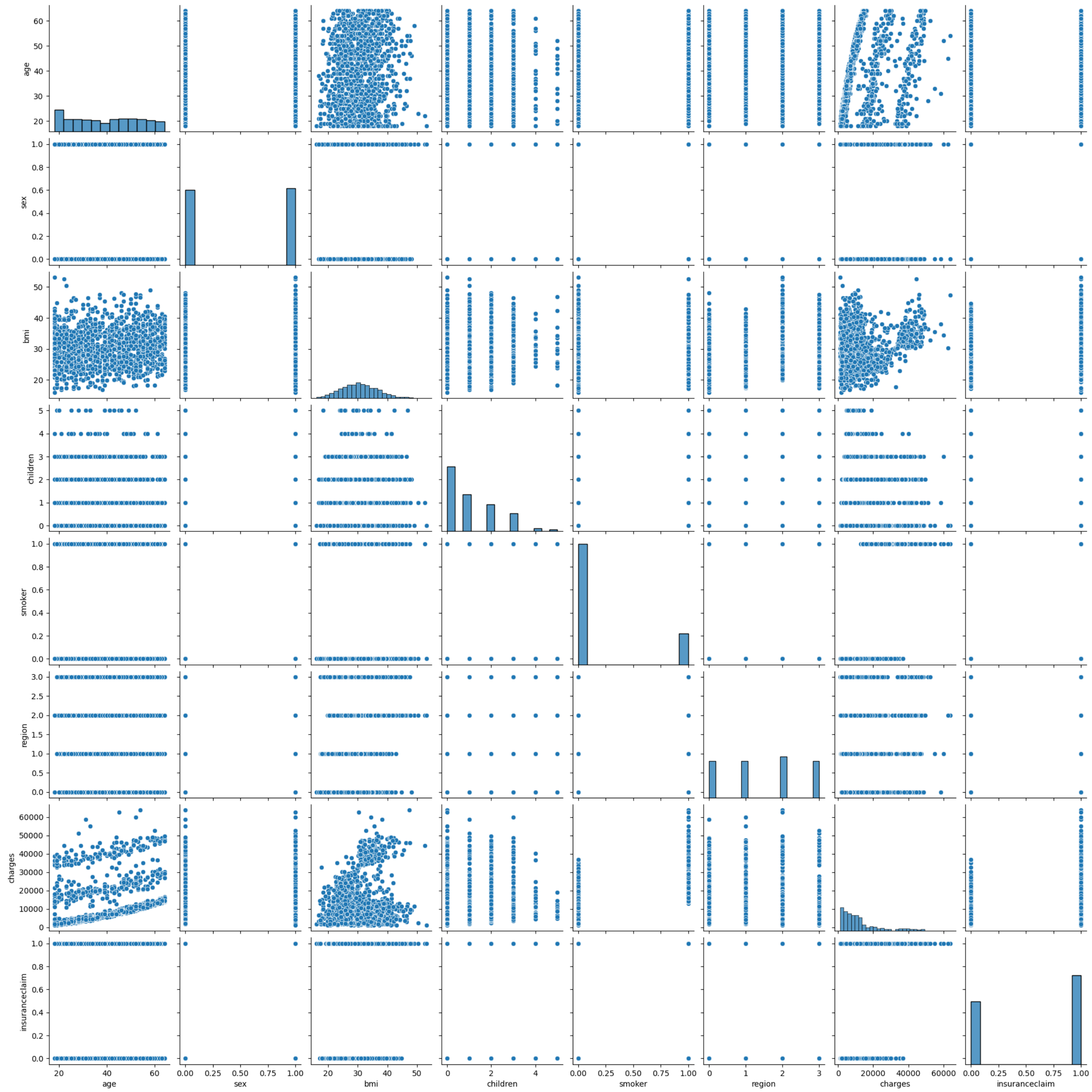
In [23]:
```
#ploting a bar grap showing number of smoker
plt.figure(figsize = (10,5))
mylables=['Non-Smoker','Smoker']
colors = ['pink','Red']
myexplode=[0.10,0.10]
size = [1064,274]
plt.pie(size,colors = colors,labels =mylables,explode = myexplode, shadow = Tr

plt.show()
```



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

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

Out[25]: <seaborn.axisgrid.PairGrid at 0x248ea07f640>



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
```

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'
plt.show()
```

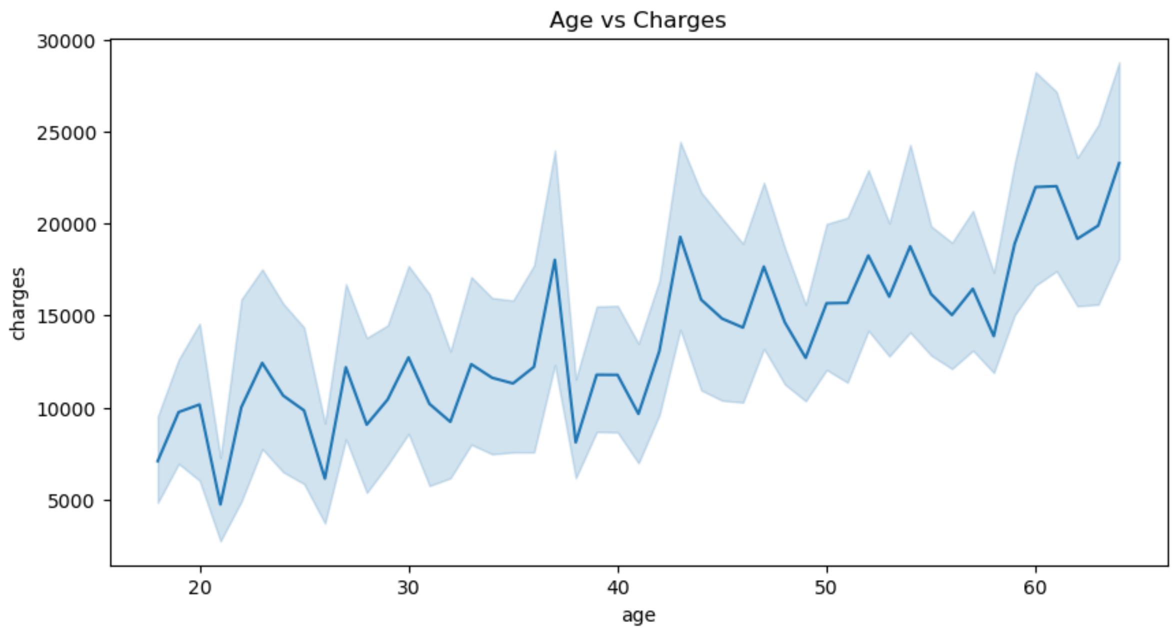
In [29]:
```
df.columns
```

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

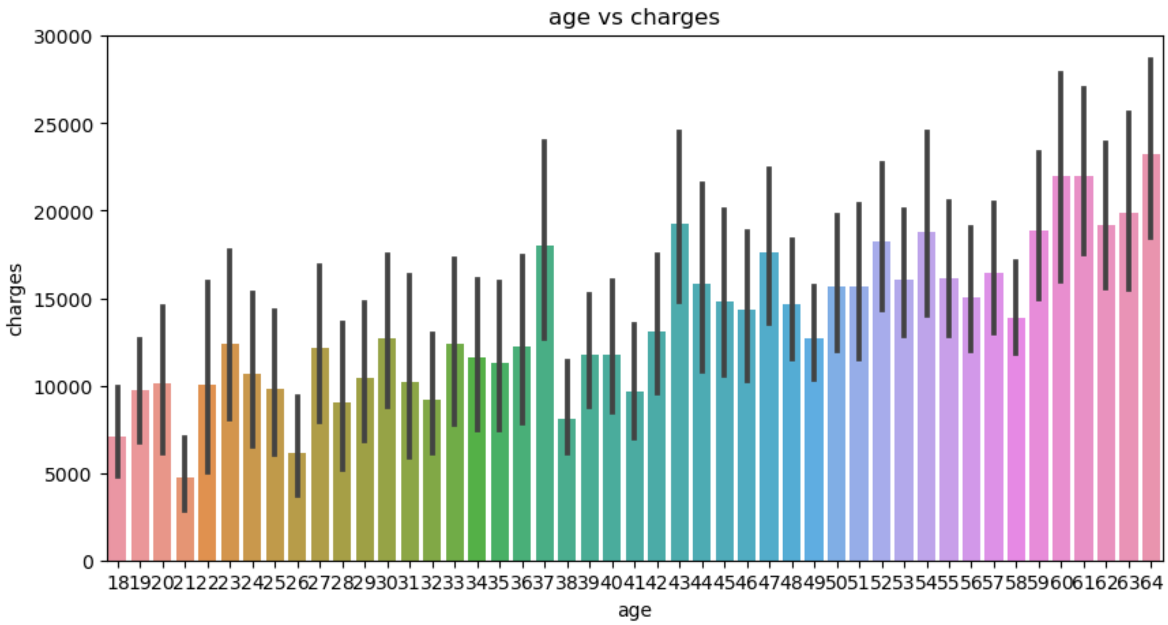
In [30]:
```
# Insites From this Heat Map :- 1. Smoker Tends to Pay More Insurance Charges;
```

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()
```

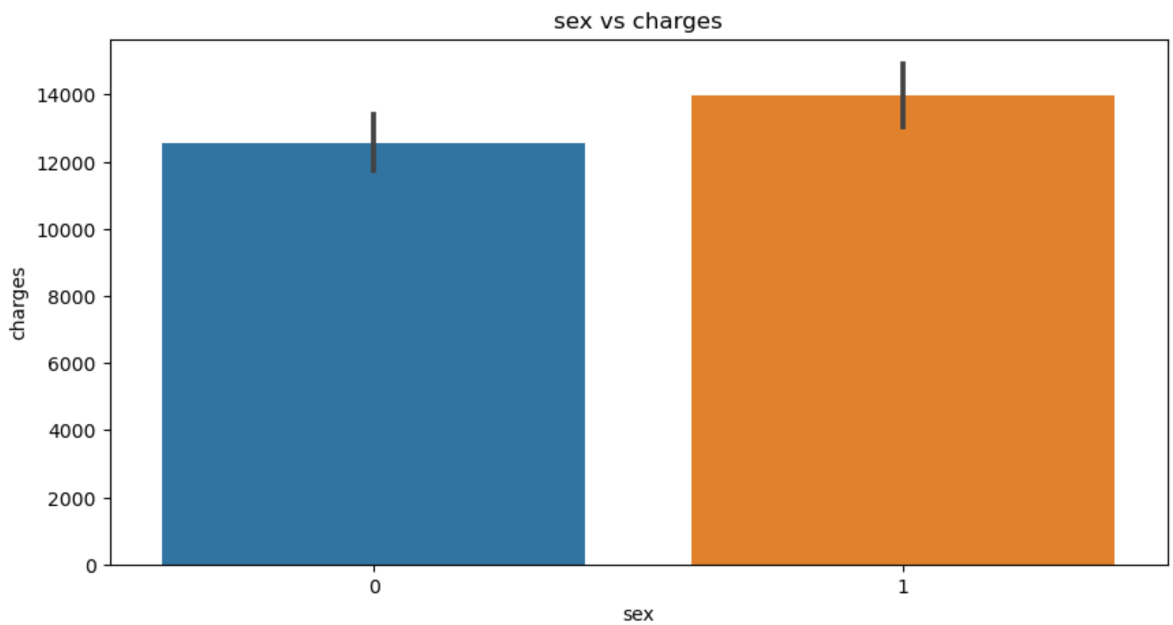


In [32]:
```
#bax plot for age vs charge

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

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



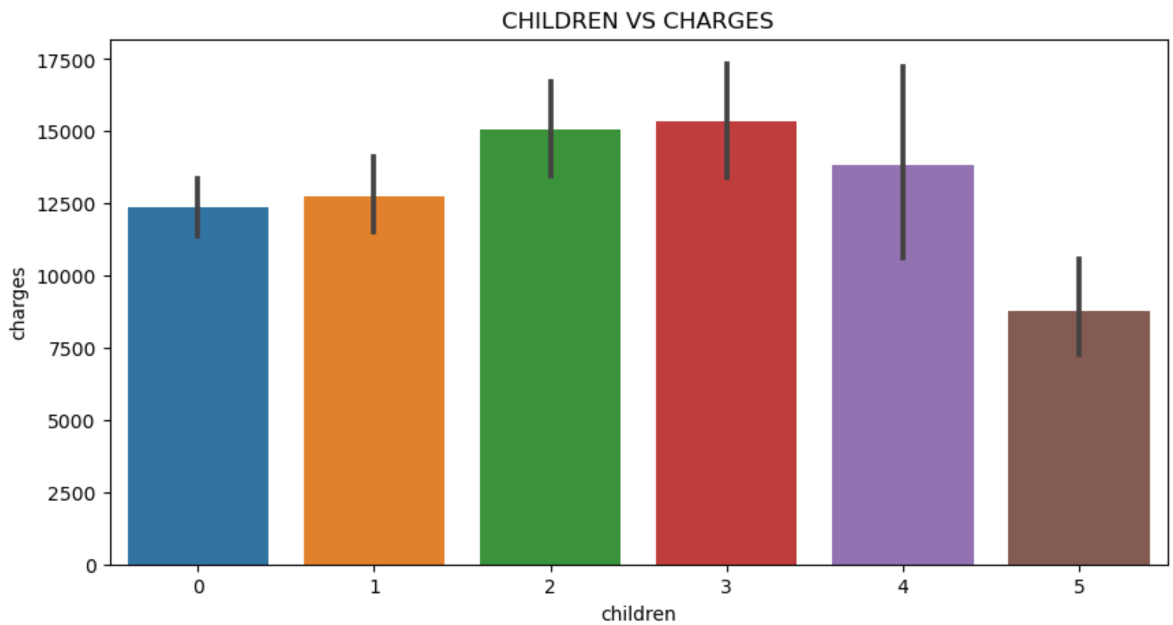
In [33]:

```
#plot the box plot of sex and charges
# as 1 belongs to men : it shows that men are paying more insurance charges th
#bar plot

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

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

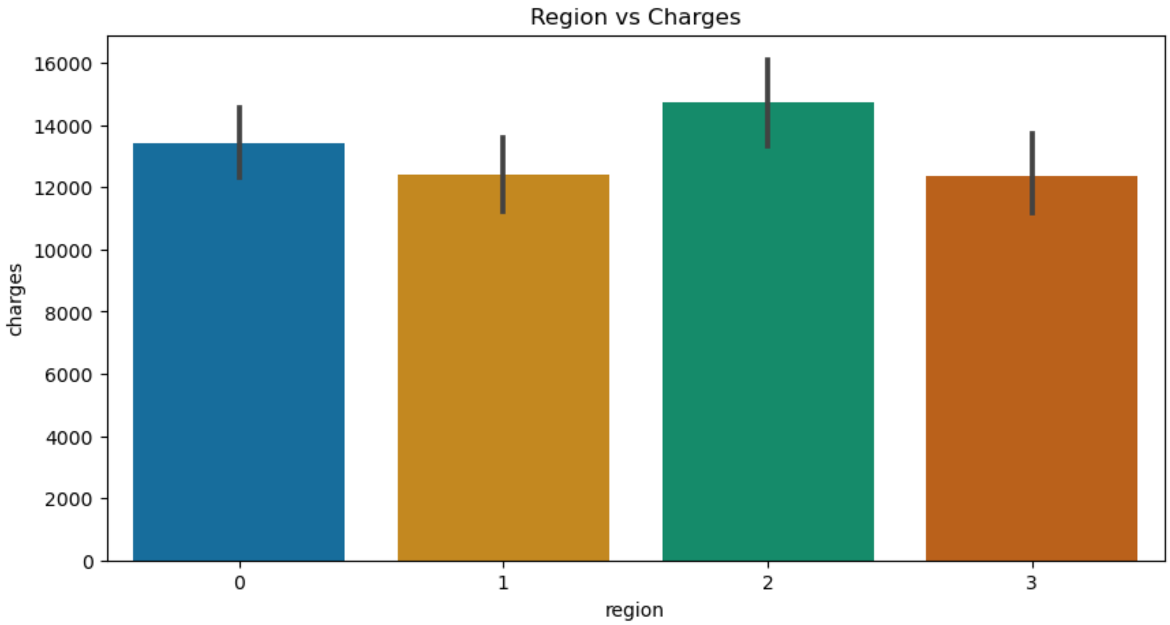


In [34]:

```
# children vs charges
# no. of childrens of a person has a weird dependency on insurance charge. i.e

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

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

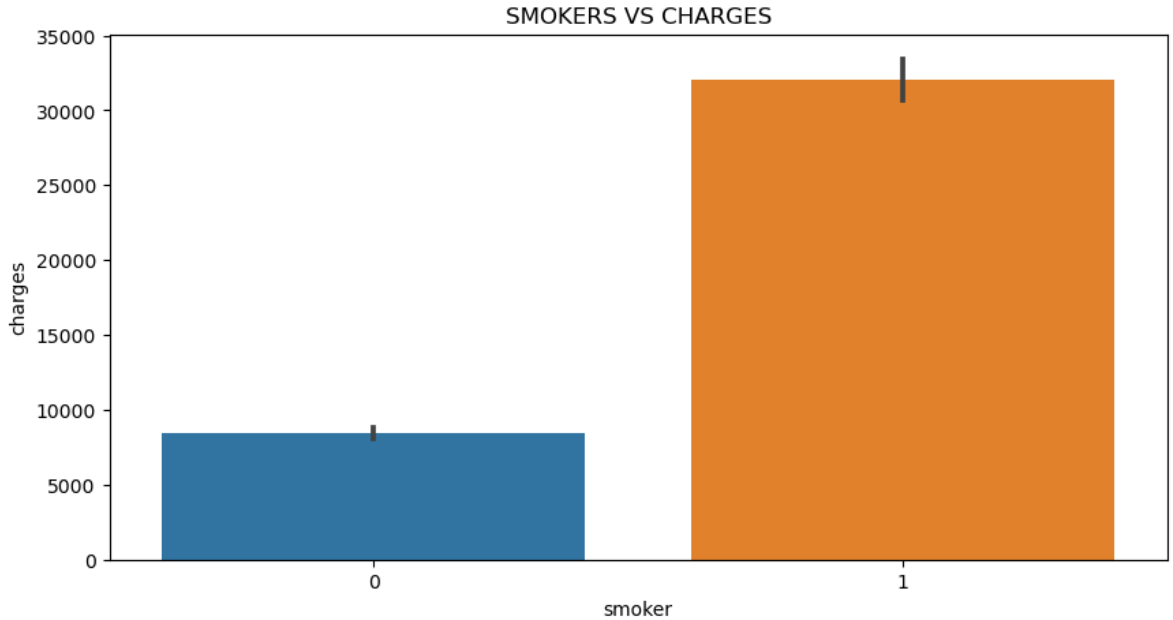


In [35]:
```
# region vs charges BAR GRAPh

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

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



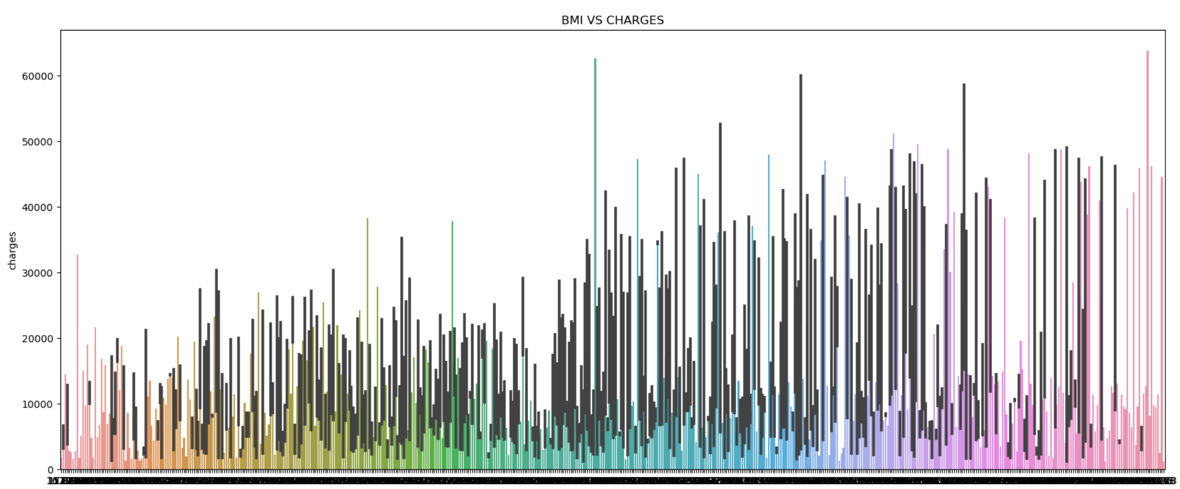
In [36]:
```
# from the graph we can clearly state that region dont play any role in charge
```

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 cl
```

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 Premiu
```

Data Cleaning

In [70]:
```
# removing un required columns from the insurance data
# As from the above grph we can clearly state that region dont play any role i

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

In [71]:
```
df.shape
```

Out[71]: (1338, 7)

In [72]:
```
#as earlier there was 10704 data point the new one has 9366 data point after r
df.size
```

Out[72]: 9366

In [73]:
```
# seperate out features and target value from dataset

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

In [74]:
```
X.shape
```

Out[74]: (1338, 6)

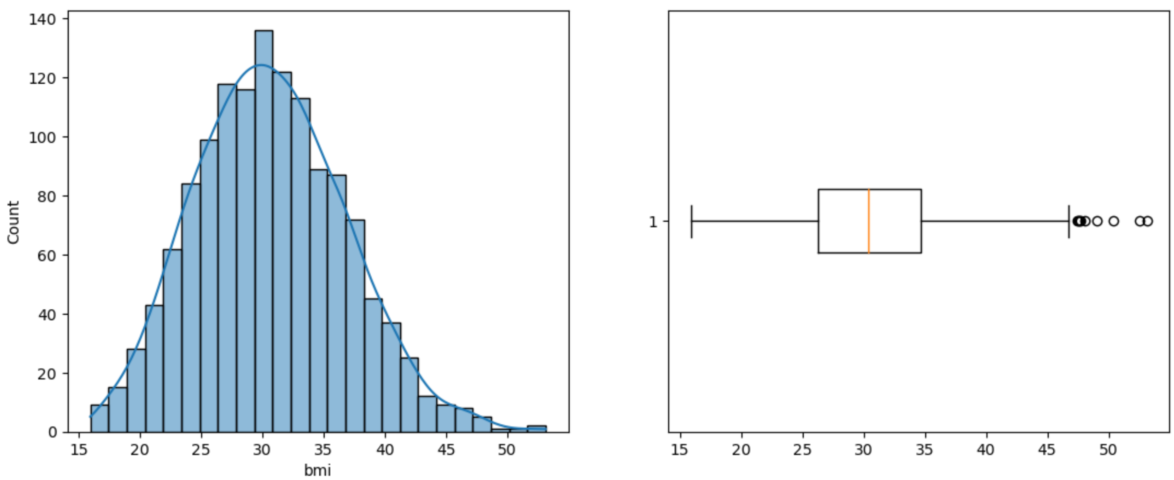
In [75]:
```
y.shape
```

Out[75]: (1338,)

Finding an Outlier

In [76]:
```
#bmi outlier

# For BMI feature
print("BMI: ")
print("Skewness : ",round(df['bmi'].skew(),3))
plt.figure(figsize=(13,5))
plt.subplot(1,2,1)
sns.histplot(data=df['bmi'],kde=True)
plt.subplot(1,2,2)
plt.boxplot(x=df['bmi'],vert=False)
plt.show()
```

```
BMI: 
Skewness :  0.284
```



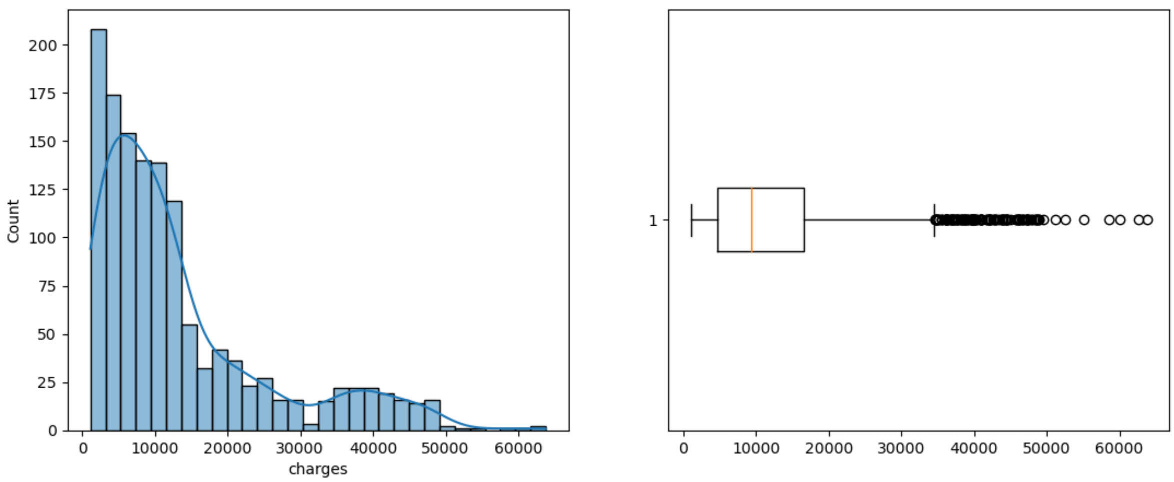
In [77]:
```
# Finding Position of Outlier
#position plot of outlier

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 [78]:
```
# bmi can be more or less as per medical condition of person so no need to tre
```

In [79]:
```
#Charges outlier
print("charges: ")
print("Skewness : ",round(df['charges'].skew(),3))
plt.figure(figsize=(13,5))
plt.subplot(1,2,1)
sns.histplot(data=df['charges'],kde=True)
plt.subplot(1,2,2)
plt.boxplot(x=df['charges'],vert=False)
plt.show()
```

```
charges:
Skewness :  1.516
```



In [80]:
```
# Charges can be More or less as per required by insurance company
```

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

In [81]:
```
#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 = 0.3, random_s
```

Scaling by Standardization

In [82]:
```
from sklearn.preprocessing import StandardScaler
```

In [83]:
```
sd = StandardScaler()
X=sd.fit_transform(X)
```

In [84]:
```
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 :  (936, 6)
X_test shape :  (402, 6)
y_train shape :  (936,)
y_test shape :  (402,)
```

Importing and Using Decision Tree (Supervised Learning) Algorithm

In [85]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [86]:
```
# model

dtc = DecisionTreeClassifier(max_depth=4)

#fitting

dtc.fit(X_train,y_train)
```

Out[86]:
```
DecisionTreeClassifier(max_depth=4)
```

In [87]:
```
#predicting via Decision Tree Algorithm

y_pred=dtc.predict(X_test)

y_pred
```

Out[87]:
```
array([1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 0, 0, 1, 1, 0], dtype=int64)
```

In [88]:
```
#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 =  0.34554737023254406
```

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

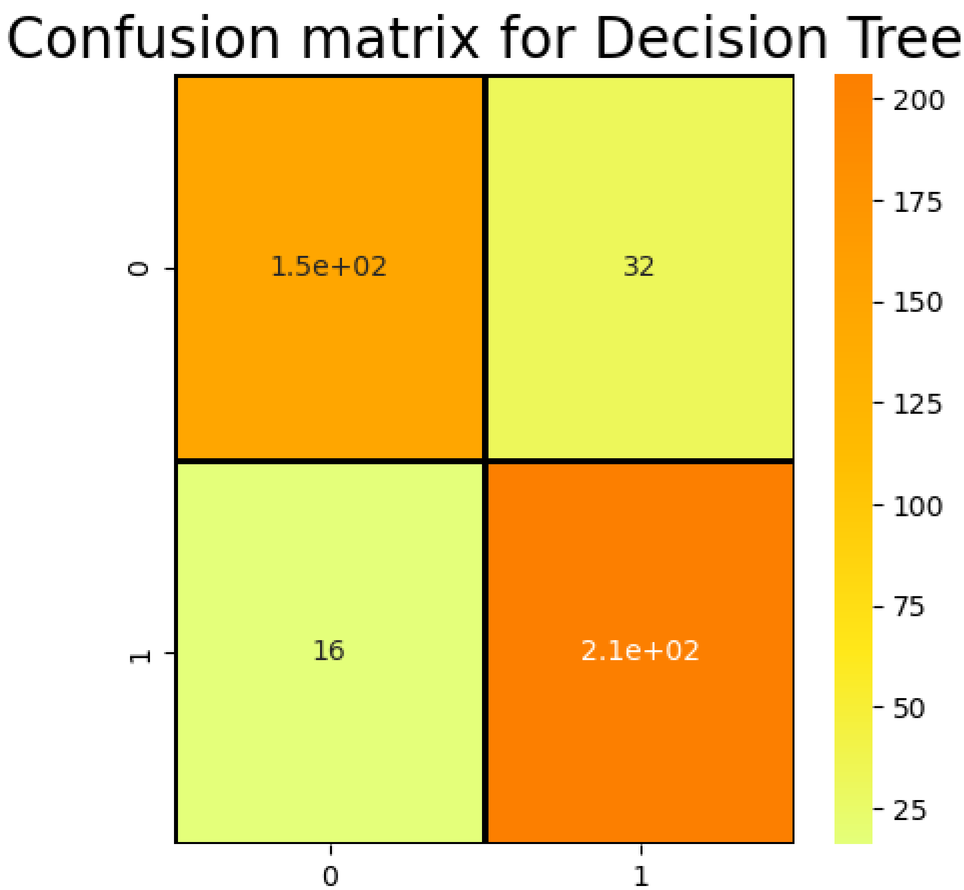
In [89]:
```
# compute accuracy on training set

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 :  0.8707264957264957
Testing Data Accuracy by Decision Tree Algorithm is :  0.8805970149253731
```

In [90]:
```
# 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 : 0.11940298507462686
RMSE : 0.34554737023254406
```

In [91]:
```
# lets evaluate the model performance
from sklearn.metrics import confusion_matrix
#Lets print the confusion metrix first
plt.rcParams['figure.figsize']=(5,5)
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True,cmap='Wistia',linewidths=1,linecolor='black')
plt.title("Confusion matrix for Decision Tree",fontsize=20)
plt.show()

from sklearn.metrics import confusion_matrix
# calculate the confusion matrix
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
# print the confusion matrix
print("Confusion Matrix:")
print("True Negative:",tn,"False Positive:",fp)
print("False Negative:",fn,"True Positive:",tp)
```




```
Confusion Matrix:
True Negative: 148 False Positive: 32
False Negative: 16 True Positive: 206
```

In [92]:
```
df.head()
```
Out[92]:

	age	sex	bmi	children	smoker	charges	insuranceclaim
0	19	0	27.900	0	1	16884.92400	1
1	18	1	33.770	1	0	1725.55230	1
2	28	1	33.000	3	0	4449.46200	0
3	33	1	22.705	0	0	21984.47061	0
4	32	1	28.880	0	0	3866.85520	1

In [95]:
```
# Lets view how model predicts the result
new_check = dtc.predict((np.array([[32,0,44,0,1,3500]])))
print("Provide Wether insurance claim for given condition (should claim be giv
```
```
Provide Wether insurance claim for given condition (should claim be given or
not) :  [1]
```
In [ ]:
```
# here 1 means insurance can be given where as 0 mean not to give
```
In [97]:
```
# Lets view how model predicts the result
new_check = dtc.predict((np.array([[25,1,66,3,0,1000]])))
print("Provide Wether insurance claim for given condition (should claim be giv
```
```
Provide Wether insurance claim for given condition (should claim be given or
not) :  [0]
```
Deciding a Model

In [66]:
```
# Usually We Select Only Those Model Whisch Has Highest Accuracy Among All tho
```
In [67]:
```
# Note
# We applied many models to the data, but decision tree gave the best accuracy
# Decision Tree Classifier - It gives 88% on testing and 91% on training data
# Logistic Regression - It gives 88% accuracy on testing and 88% on training a
# SVC - It gives 87% accuracy
# Naive Bayes - It gives 75-76% Accuracy
```

In [68]:
```
# Lets print the classifiction report also
from sklearn.metrics import classification_report
cr =classification_report(y_test,y_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.90      0.82      0.86       180
           1       0.87      0.93      0.90       222

    accuracy                           0.88       402
   macro avg       0.88      0.88      0.88       402
weighted avg       0.88      0.88      0.88       402
```

In [69]:
```
# High precision indicates that the model is making less but some false positi
# As all the values of precision recall and f1-score is near 1
# While high recall indicates that the model is making fewer false negative pr
# We can say our model gives good precision, recall and f1-score
```

In [ ]:
```
## Created by Bharat Bhushan Kulmani
```