Medical Cost Personal Insurance Project

Project Goal: The goal of this project is to develop a predictive model that accurately estimates health insurance costs for individuals based on a set of input features including age, gender, body mass index (BMI), number of dependents, smoking status, and residential region in the United States. The primary objective is to create a robust and reliable algorithm that can assist individuals, insurance companies, and healthcare providers in predicting insurance costs with a high degree of accuracy.

In [1]:
```
#Importing Libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#Loading DataSet

df=pd.read_csv("https://raw.githubusercontent.com/dsrscientist/dataset4/main/me
df
```

Out[2]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

Variables in Dataset:

- **age**: age of primary beneficiary
- **sex:** insurance contractor gender, female, male

- **bmi:** Body mass index, providing an understanding of body, weights that are relatively high or low relative to height, objective index of body weight (kg / m ^ 2) using the ratio of height to weight, ideally 18.5 to 24.9.
- **children:** Number of children covered by health insurance / Number of dependents
- **smoker:** Smoking
- **region:** the beneficiary's residential area in the US, northeast, southeast, southwest, northwest.
- **charges:** Individual medical costs billed by health insurance

In [3]:
```
df.shape
```

Out[3]:
```
(1338, 7)
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype
='object')
```

In [5]:
```
df.dtypes
```

Out[5]:
```
age           int64
sex          object
bmi         float64
children      int64
smoker       object
region       object
charges     float64
dtype: object
```

Observation :

- we can see there are 1338 rows and 7 columns in the dataset.
- Among 7 column, one column 'charges' is our target variable, remaining 6 columns are independent variables.

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

Observations:

- We can see in the dataset there are 3 types of datatype are present in the dataset which are integar, float & object
- We have 2 columns holding integar value, 2 columns contain float values and rest 3 columns has object values.

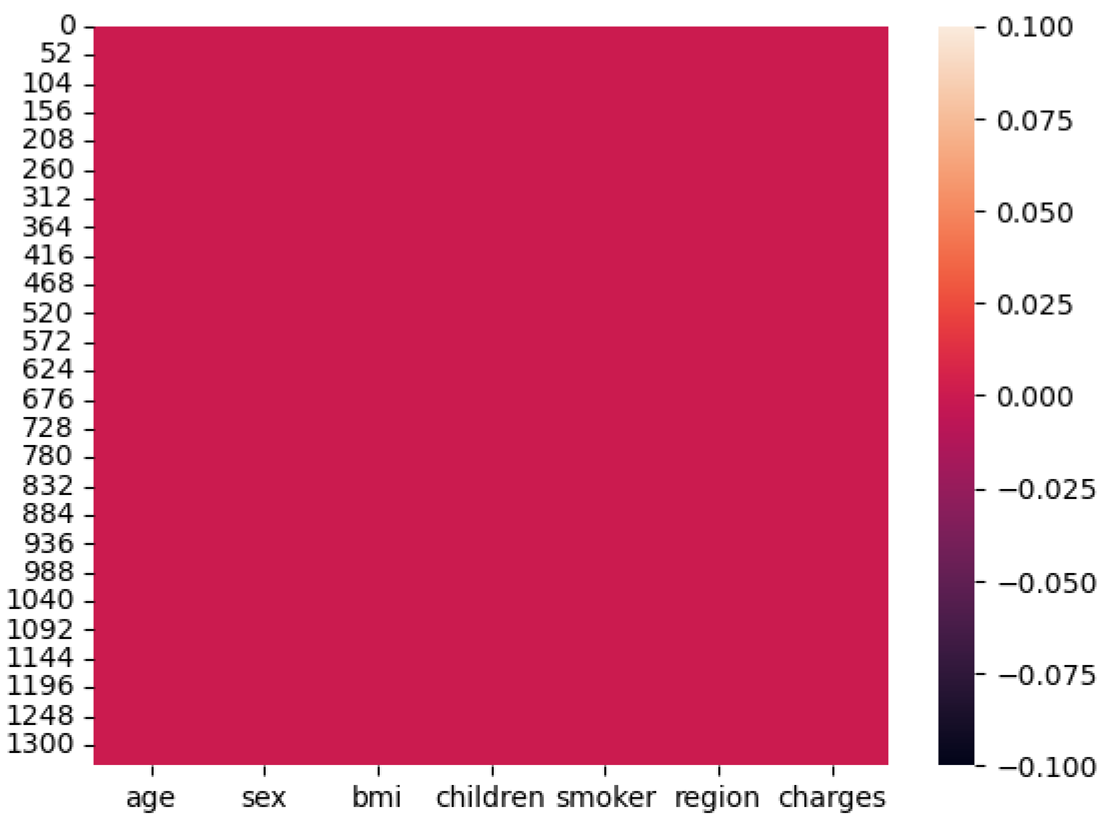
Missing Data

In [7]:
```
#Checking for missing values

df.isnull().sum()
```

Out[7]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

In [8]:
```
#lets visualise it
sns.heatmap(df.isnull())
```

Out[8]:
```
<Axes: >
```



Observations :

- we can clearly visualize there is no missing values in the dataset.

Duplicate Values

In [9]:
```
#Checking duplicate values
df.duplicated().sum()
```

Out[9]: 1

In [10]:
```
#dropping duplicated Values
df.drop_duplicates(inplace=True)

#checking Duplicated Values again
df.duplicated().sum()
```

Out[10]: 0

In [11]:
```
#checking dimension of data after removing duplicate values
df.shape
```

Out[11]: (1337, 7)

Observation :

- 1 duplicated values found in the dataset, so we have droped the duplicated row.
- Now our dataframe contains 1337 rows and 7 columns

In [12]:
```
df.nunique().to_frame("No. of Unique Values in Each column")
```

Out[12]:

	No. of Unique Values in Each column
age	47
sex	2
bmi	548
children	6
smoker	2
region	4
charges	1337

Observations :

- This are the unique values in each column in the dataframe.

In [13]:
```
#checking unique values in target Column
df["charges"].value_counts()
```

Out[13]:
```
16884.92400    1
2117.33885     1
2221.56445     1
19798.05455    1
13063.88300    1
              ..
7345.08400     1
26109.32905    1
28287.89766    1
1149.39590     1
29141.36030    1
Name: charges, Length: 1337, dtype: int64
```

Observation : We can clearly visialize that our target variable has 1337 unique values which equal to the number of rows in our dataframe, it will be termed as "Regression problem" where we need to predict the health insurance costs using the regression model.

Column Types:

In [14]:
```
num_colmn = df.select_dtypes(include=['int64', 'float64']).columns
print("Numerical columns in our dataset :\n", num_colmn,"\n")

cat_column = df.select_dtypes(include=['object']).columns
print("categorical columns in our dataset :\n", cat_column)
```

```
Numerical columns in our dataset :
 Index(['age', 'bmi', 'children', 'charges'], dtype='object')

categorical columns in our dataset :
 Index(['sex', 'smoker', 'region'], dtype='object')
```

Observations: These are the numerical and categorical columns in our set.

In [15]:
```
#description of data in categorical columns:
df.describe()
```

Out[15]:

	age	bmi	children	charges
count	1337.000000	1337.000000	1337.000000	1337.000000
mean	39.222139	30.663452	1.095737	13279.121487
std	14.044333	6.100468	1.205571	12110.359656
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.290000	0.000000	4746.344000
50%	39.000000	30.400000	1.000000	9386.161300
75%	51.000000	34.700000	2.000000	16657.717450
max	64.000000	53.130000	5.000000	63770.428010

Observation:

- 'age' column does not appear to be significantly skewed. It is approximately symmetric or very close to being normally distributed.
- mean of 'BMI' (30.66) is somewhat closer to the 75th percentile (34.70) than the 25th percentile (26.29). This suggests a slightly right-skewed distribution, where the tail of the distribution extends more to the right.
- mean number of 'children'(1.10) is closer to the 75th percentile (2.00) than the 25th percentile (0.00). This also indicates a right-skewed distribution, where more individuals have a higher number of dependents.

In [16]:
```
#description of data in categorical columns:
df.describe(include=['object'])
```

Out[16]:

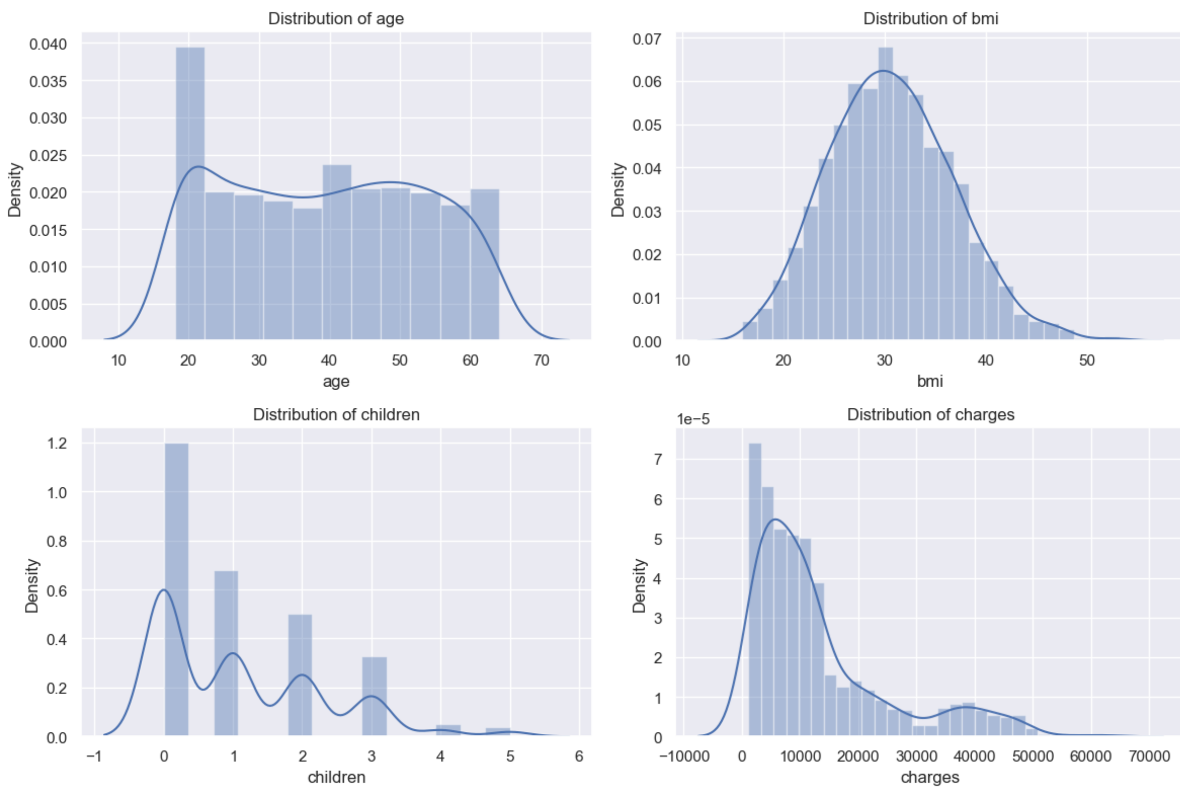
	sex	smoker	region
count	1337	1337	1337
unique	2	2	4
top	male	no	southeast
freq	675	1063	364

Observations:

- Unique Values in Sex, Smoker & Region are 2,2 and 4 Respectively.
- Based on the results from describe, we also see these variables are at the top and more frequent in the dataset: male, non-smoker, southeast region.

Lets visualize the distribution of data in thecolumns

In [17]:
```
# Numerical Columns
plt.figure(figsize=(12, 8))
p=1
for column in num_colmn:
    sns.set()
    # Adjust the subplot size within the figure
    plt.subplot(2,2,p)
    sns.distplot(df[column])
    plt.title(f'Distribution of {column}')
    p += 1
plt.tight_layout()
plt.show()
```

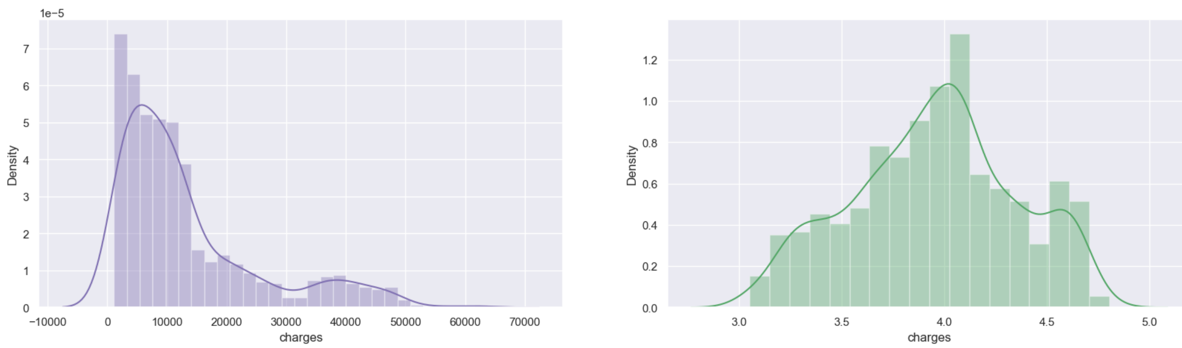


Observations :

- Age: As we know we have 47 unique values in our dataset and Maximum people are in age between 20-25
- BMI : Normal BMi is ideally 18.5 to 24.9 however, in our dataset we can see maximum occurances in between 24.9 to 35 approx
- Children: In out dataset Maximum number of people don't have children. We can visualize that data is showing skewness. However, we wont remove it at is belongs to categorical data
- Charges : It is our target column, it contains 1337 unique values, and maximum people are getting 1000-17000

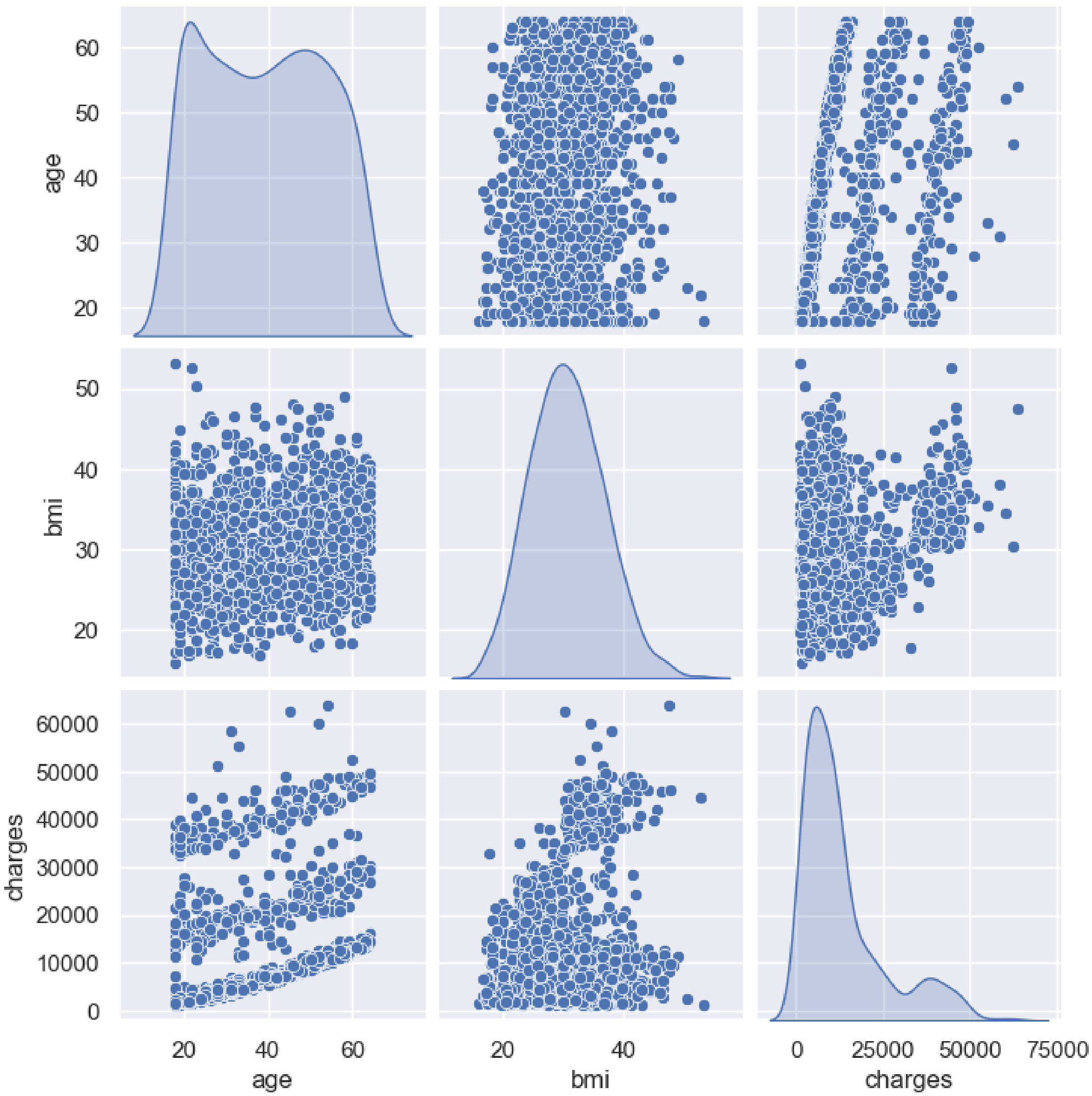
In [18]:

```
#Handling Distribution of Charges

# Distribution of the charges
plt.figure(figsize = (20,5))
plt.subplot(1,2,1)
sns.distplot(df.charges, color = 'm')

# Natural Log for approximately normal distribution
plt.subplot(1,2,2)
sns.distplot(np.log10(df.charges), color = 'g')
plt.show()
```



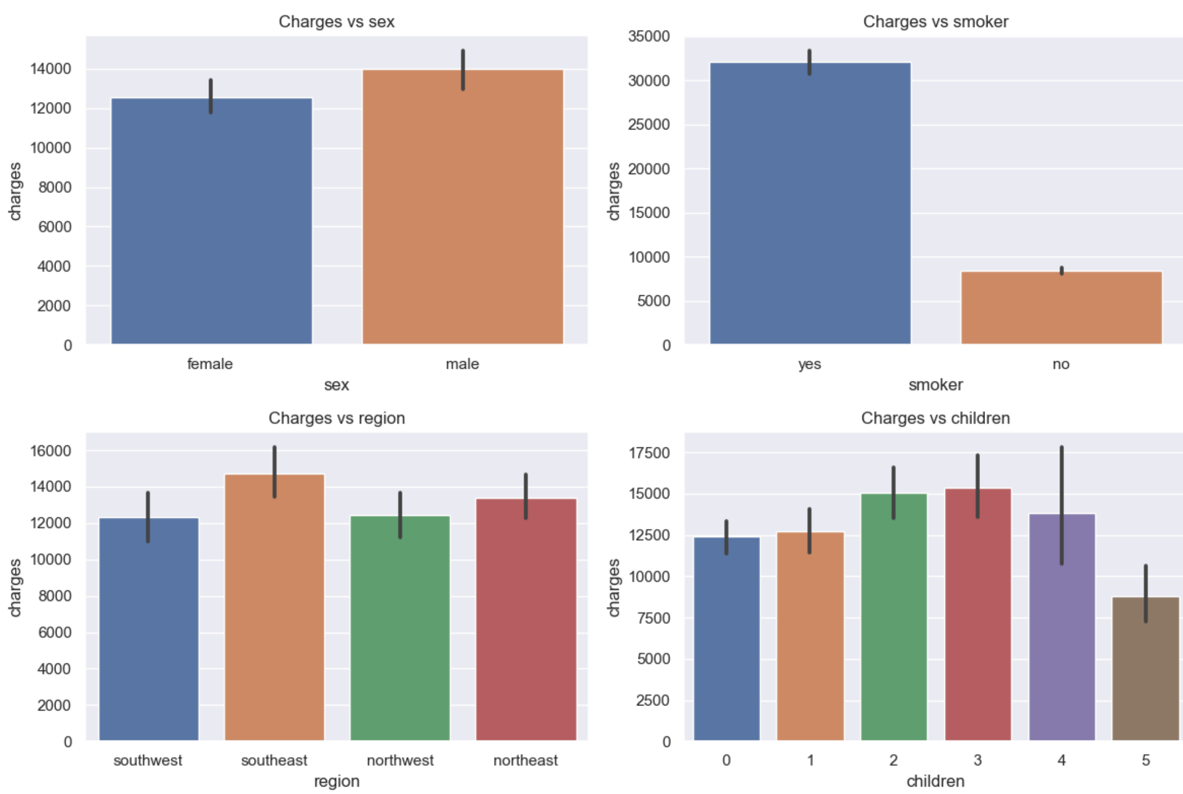
Observations : The distribution of charges exhibits right-skewness, prompting us to apply the natural logarithm transformation to approximate a more normal distribution.

In [19]:
```
# Pairplot for Age, BMI, Charges
numerical_df = df[num_colmn]

# Remove the 'children' column from numerical_df
numerical_df.drop('children', axis=1, inplace=True)

sns.pairplot(numerical_df, diag_kind = 'kde')
plt.show()
```



Observations: The pairplot indicates the presence of a linear relationship between age and BMI with respect to charges.

In [20]:
```
#Adding children variable in categorical column
cat_column = cat_column.append(pd.Index(['children']))

#Create a DataFrame containing the selected categorical columns
cat_df = df[cat_column]

p = 1
plt.figure(figsize=(12, 8))

for column in cat_df:
    sns.set()
    if p <= 4:
        plt.subplot(2, 2, p)
        sns.barplot(x=column, y='charges', data=df)
        plt.title(f'Charges vs {column}')
        p += 1

plt.tight_layout()
plt.show()
```

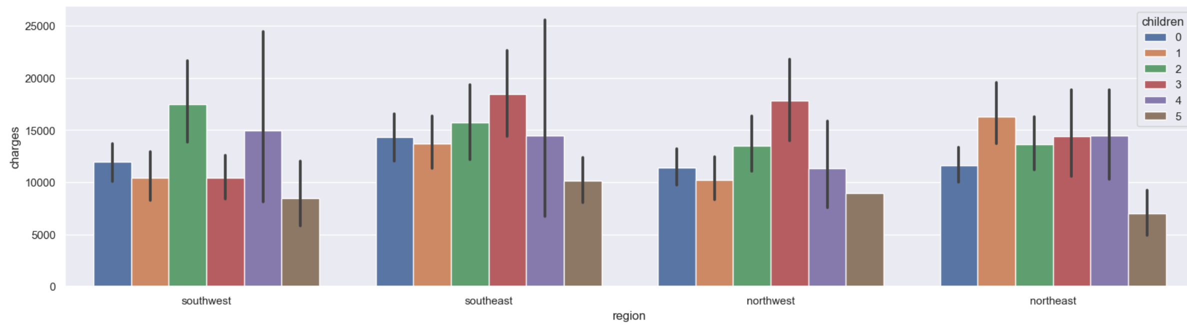


Observations :

- Sex: The data is bit baised towards male than women in the dataset.
- Smoker: Ration of smoker is higher than non smoker in our dataset.And we can clearly visualise smokers medical charges is higher than non smokers.
- Region: We can notice southeast region are paying higher than other 3 regions.
- Children: Here cam see people with 2 and 3 children than the people with 0,1,4 and 5 respectively.

Bivariate Analysis

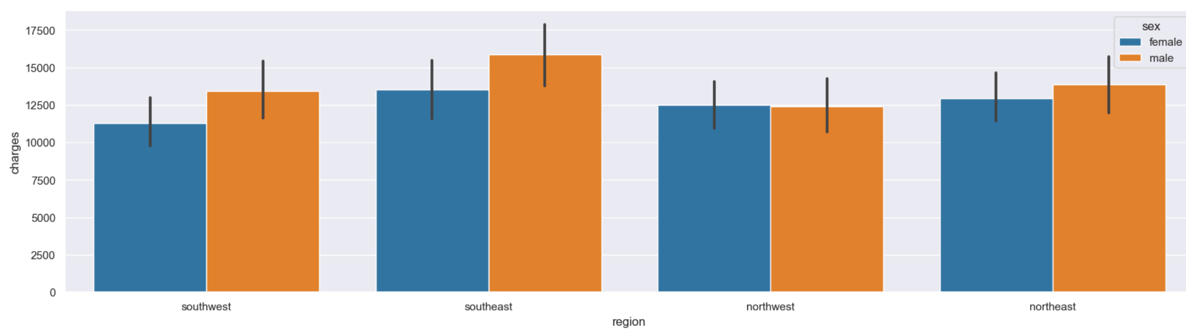
In [21]:
```
# Setting children as hue parameter
plt.figure(figsize=(20,5))
ax = sns.barplot(x = 'region', y = 'charges', hue = 'children', data = df)
```



Observations :

- It appears that there is no statistically significant difference in how the number of children in a specific region impacts medical charges.

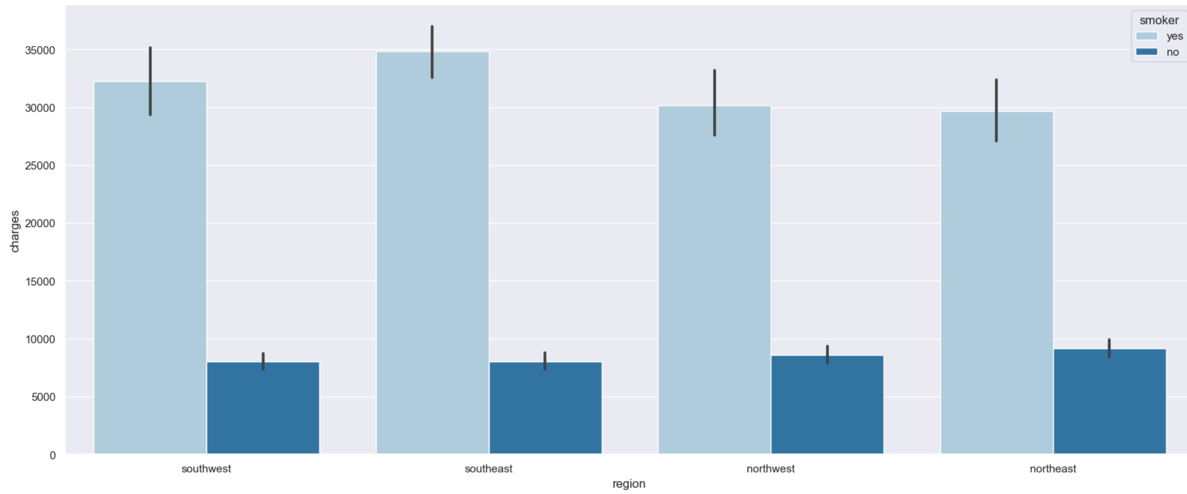
In [22]:
```
# setting Sex as Hue parameter
plt.figure(figsize=(20,5))
sns.barplot(x = 'region', y = 'charges', hue = 'sex', data = df, palette = 'tab
plt.show()
```



Observations :

- Once again, similar to the previous graph, there doesn't appear to be a statistically significant difference in how gender within a specific region affects medical charges.

In [23]:
```
plt.subplots(1, 1, figsize = (20, 8))
sns.barplot(x = 'region', y = 'charges', hue = 'smoker', data = df, palette = '
plt.show()
```

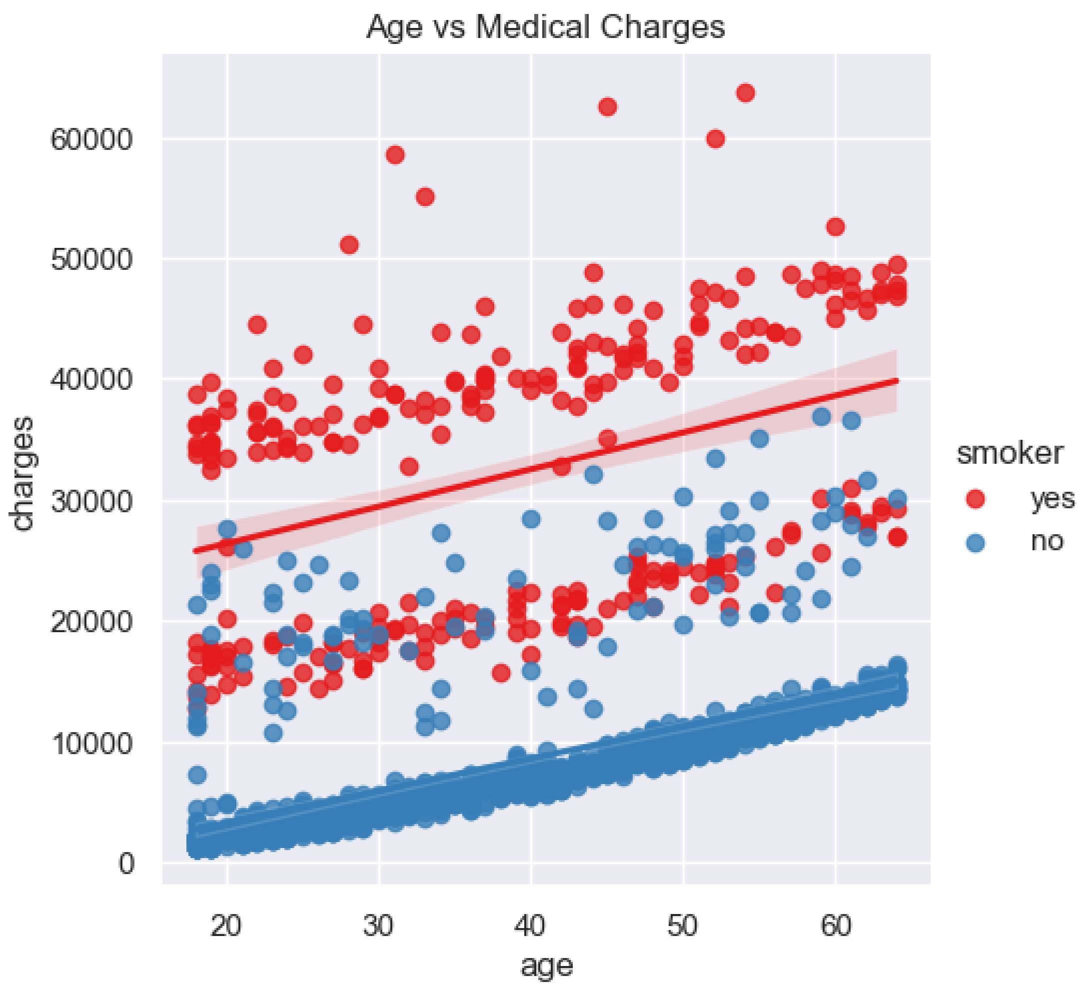


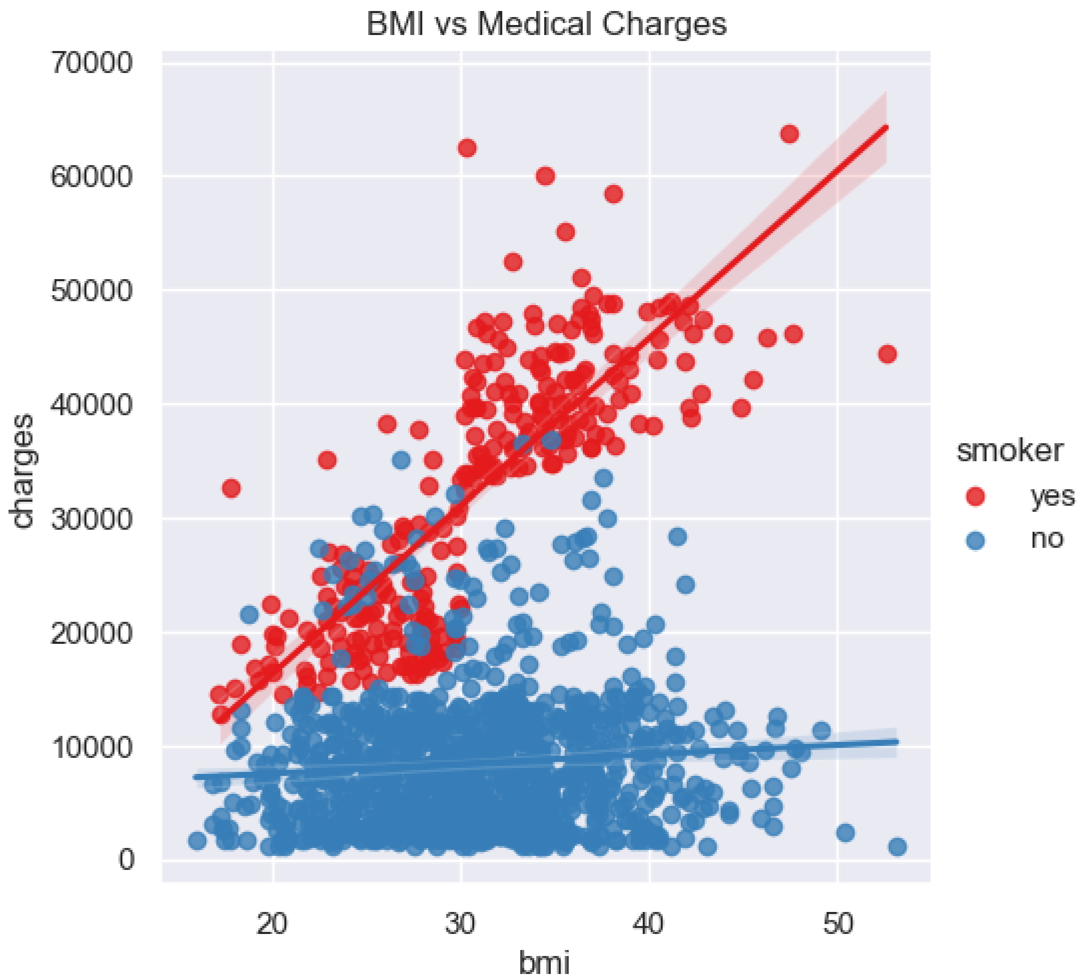
Observations: In contrast to the other factors, it appears that there is a statistically significant difference in how smoking behavior within a region impacts medical charges.

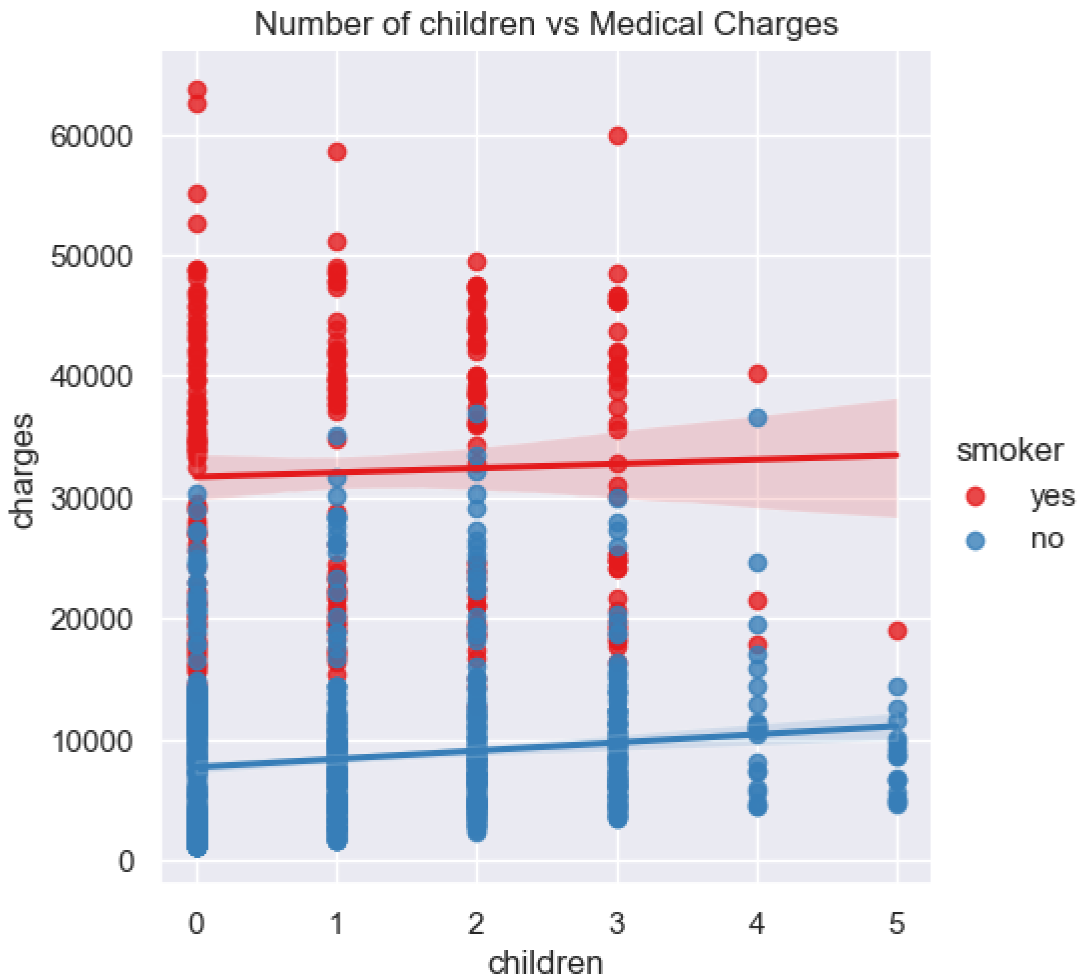
From these bar charts, it is evident that smoking behavior by region significantly affects medical charges. Now, let's examine the correlation between non-categorical variables (age, BMI, and children) and medical charges, considering smoking behavior as a factor.

In [24]:
```
#Charges by age, bmi, and children based on smoking behavior

ax = sns.lmplot(x = 'age', y = 'charges', data = df, hue = 'smoker', palette =
plt.title('Age vs Medical Charges')

ax = sns.lmplot(x = 'bmi', y = 'charges', data = df, hue = 'smoker', palette =
plt.title('BMI vs Medical Charges')

ax = sns.lmplot(x = 'children', y = 'charges', data = df, hue = 'smoker', palet
plt.title('Number of children vs Medical Charges')
```

Out[24]:
```
Text(0.5, 1.0, 'Number of children vs Medical Charges')
```




BMI vs Medical Charges
70000
60000
50000
40000
30000
20000
10000
0
charges
20
30
40
50
bmi
smoker
yes
no



Observations: Based on the analysis, it's clear that smoking has a significant impact on medical costs. When combined with other factors such as age, BMI, and the number of children, the effect of smoking on medical costs becomes even more pronounced and leads to higher medical expenses. In other words, smoking, when considered alongside these factors, contributes to increased medical costs.

Encoding the categorical features

In [25]:
```
# Converting Data to numerical type using LabelEncoder
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()

label.fit(df.region)
df.region = label.transform(df.region)

label.fit(df.sex)
df.sex = label.transform(df.sex)

label.fit(df.smoker)
df.smoker = label.transform(df.smoker)

df.dtypes
```

Out[25]:
```
age           int64
sex           int32
bmi         float64
children      int64
smoker        int32
region        int32
charges     float64
dtype: object
```

In [26]:
```
df.head()
```

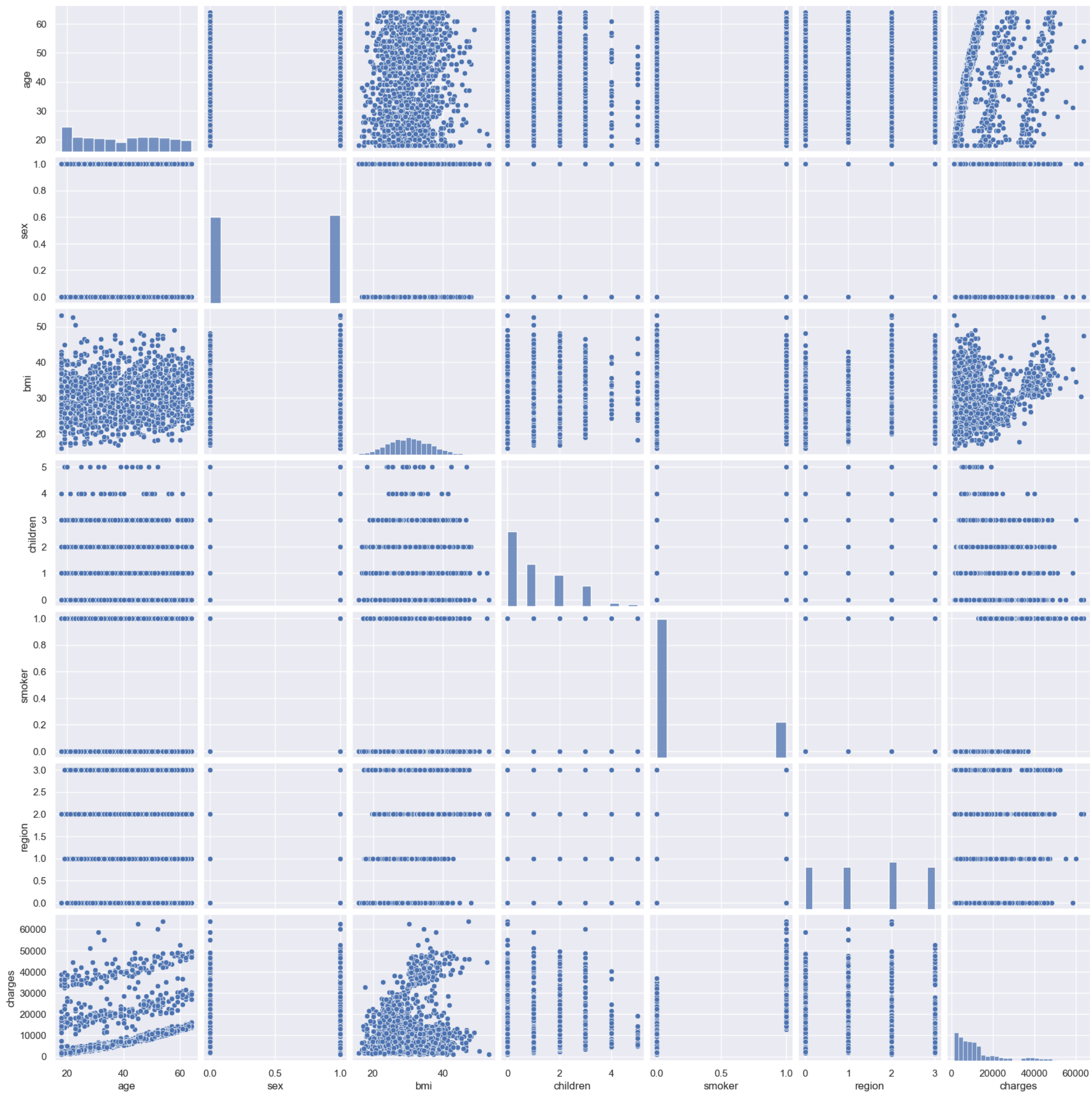
Out[26]:

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

Observations : We have converted all the categorical columns in numeric

Multivariate Analysis

In [27]:
```
sns.pairplot(df, palette ='hot_r')
plt.show()
```



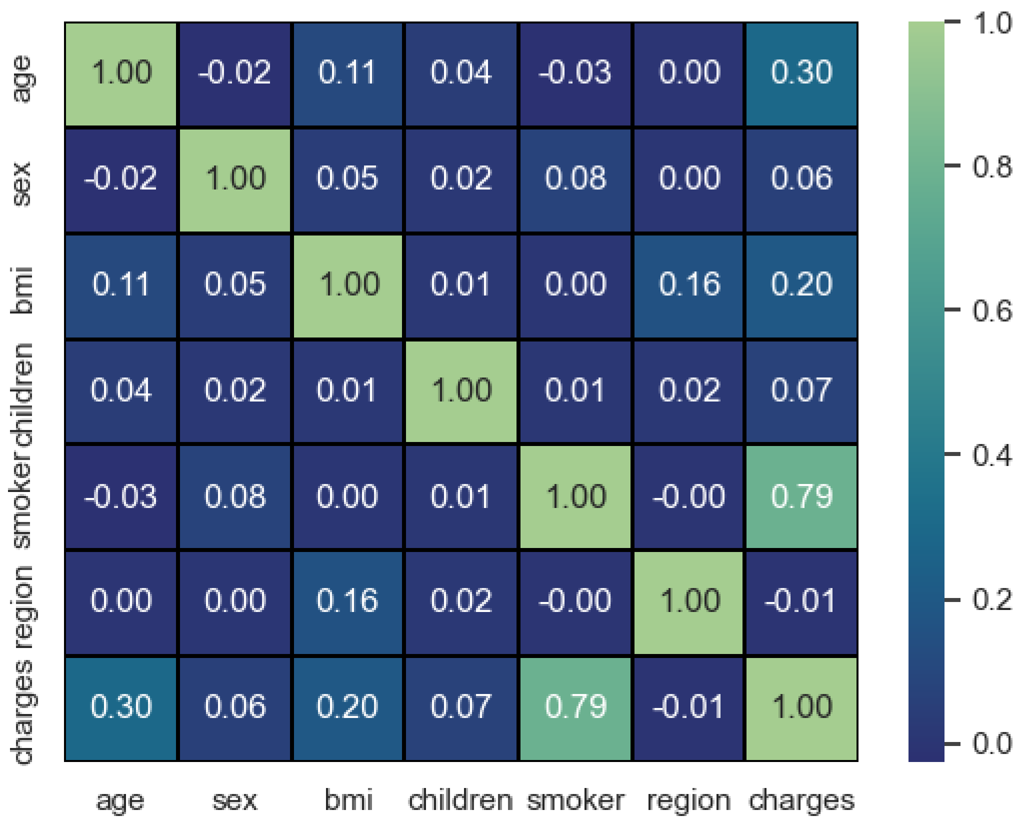
Correlation between target variables & independent variables

In [28]:
```
df.corr()
```

Out[28]:

	age	sex	bmi	children	smoker	region	charges
age	1.000000	-0.019814	0.109344	0.041536	-0.025587	0.001626	0.298308
sex	-0.019814	1.000000	0.046397	0.017848	0.076596	0.004936	0.058044
bmi	0.109344	0.046397	1.000000	0.012755	0.003746	0.157574	0.198401
children	0.041536	0.017848	0.012755	1.000000	0.007331	0.016258	0.067389
smoker	-0.025587	0.076596	0.003746	0.007331	1.000000	-0.002358	0.787234
region	0.001626	0.004936	0.157574	0.016258	-0.002358	1.000000	-0.006547
charges	0.298308	0.058044	0.198401	0.067389	0.787234	-0.006547	1.000000

In [29]:
```
sns.heatmap(df.corr(),annot=True, fmt='0.2f',linewidth=0.2, linecolor='black',c
plt.show()
```

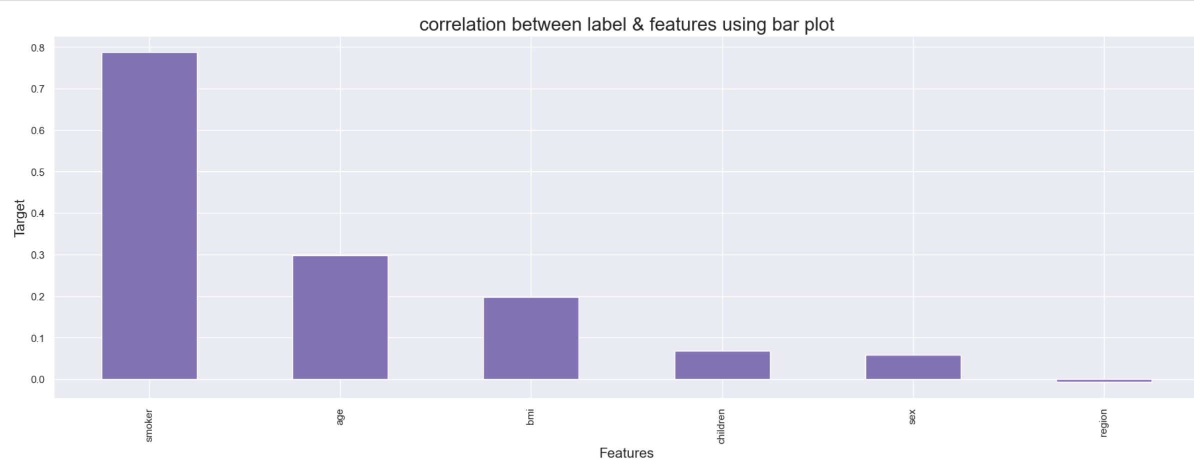


In [30]:
```
df.corr().charges.sort_values()
```

Out[30]:
```
region      -0.006547
sex          0.058044
children     0.067389
bmi          0.198401
age          0.298308
smoker       0.787234
charges      1.000000
Name: charges, dtype: float64
```

Visualizing the correlation between label & features using bar plot

In [31]:
```
plt.figure(figsize=(22,7))
df.corr()['charges'].sort_values(ascending=False).drop(['charges']).plot(kind='
plt.xlabel('Features', fontsize=15)
plt.ylabel('Target', fontsize=15)
plt.title('correlation between label & features using bar plot', fontsize=20)
plt.show()
```



Observation : From the above barplot we can notice the positive correlationship between the features and the target. Here 'smoker' is positive correlation with our target , gradually it decrease to age than bmi and child, than sex and less correlation with region.

Separating Features & Labels

In [32]:
```
#separating independent and target variables into x and y
x=df.drop('charges', axis=1)
y=df['charges']

print("Feature Dimension ", x.shape)
print("Label Dimension", y.shape)
```

```
Feature Dimension  (1337, 6)
Label Dimension (1337,)
```

We have scaled the data using Standard Scalarized method to overcome the issue of biasness.

In [33]:
```
#finding the best random state

#importing necessary libraries:
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
```

In [34]:
```
#finding the best random state
maxAccu=0
maxRS=0
for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size= 0.25, random_
    lr=LinearRegression()
    lr.fit(x_train,y_train)
    pred=lr.predict(x_test)
    acc=r2_score(y_test,pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print('Maimum r2 score is ', maxAccu, "Random_State ",maxRS)
```

```
Maimum r2 score is  0.8095751620376272 Random_State  11
```

In [35]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size= 0.25, random_st
```

In [36]:
```
print("x_train :", x_train.shape)
print("x_test :", x_test.shape)
print("y_train :", y_train.shape)
print("y_test :", y_test.shape)
```

```
x_train : (1002, 6)
x_test : (335, 6)
y_train : (1002,)
y_test : (335,)
```

In [37]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.linear_model import Lasso, Ridge
```

In [38]:
```
LR= LinearRegression()
LR.fit(x_train,y_train)
pred_LR=LR.predict(x_test)
pred_train=LR.predict(x_train)
print('R2 Score: ', r2_score(y_test,pred_LR))
print('R2 on training Data: ', r2_score(y_train,pred_train)*100)
print('Mean Absolute Error: ', mean_absolute_error(y_test,pred_LR))
print('Mean Squared Error: ', mean_squared_error(y_test,pred_LR))
print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y_test,pred_LR)))
```

```
R2 Score:  0.8095751620376272
R2 on training Data:  73.14466579170875
Mean Absolute Error:  3758.9463608385913
Mean Squared Error:  25834546.641032364
Root Mean Squared Error:  5082.769583704574
```

In [39]:
```
RFR= RandomForestRegressor()
RFR.fit(x_train,y_train)
pred_RFR=RFR.predict(x_test)
pred_train=RFR.predict(x_train)
print('R2 Score: ', r2_score(y_test,pred_RFR))
print('R2 on training Data: ', r2_score(y_train,pred_train)*100)
print('Mean Absolute Error: ', mean_absolute_error(y_test,pred_RFR))
print('Mean Squared Error: ', mean_squared_error(y_test,pred_RFR))
print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y_test,pred_RFR)))
```

```
R2 Score:  0.8708940780634452
R2 on training Data:  97.38019125984236
Mean Absolute Error:  2426.149850421517
Mean Squared Error:  17515535.250514265
Root Mean Squared Error:  4185.15653835245
```

In [40]:
```
KNN= KNN()
KNN.fit(x_train,y_train)
pred_KNN=KNN.predict(x_test)
pred_train=KNN.predict(x_train)
print('R2 Score: ', r2_score(y_test,pred_KNN))
print('R2 on training Data: ', r2_score(y_train,pred_train)*100)
print('Mean Absolute Error: ', mean_absolute_error(y_test,pred_KNN))
print('Mean Squared Error: ', mean_squared_error(y_test,pred_KNN))
print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y_test,pred_KNN)))
```

```
R2 Score:  0.1082367965286255
R2 on training Data:  40.762357520748694
Mean Absolute Error:  7828.773399776121
Mean Squared Error:  120983682.1675014
Root Mean Squared Error:  10999.258255332557
```

In [41]:
```
GBR= GradientBoostingRegressor()
GBR.fit(x_train,y_train)
pred_GBR=GBR.predict(x_test)
pred_train=GBR.predict(x_train)
print('R2 Score: ', r2_score(y_test,pred_GBR))
print('R2 on training Data: ', r2_score(y_train,pred_train)*100)
print('Mean Absolute Error: ', mean_absolute_error(y_test,pred_GBR))
print('Mean Squared Error: ', mean_squared_error(y_test,pred_GBR))
print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y_test,pred_GBR))
```

```
R2 Score:  0.9021119726679765
R2 on training Data:  89.6512123733554
Mean Absolute Error:  2231.998710617618
Mean Squared Error:  13280267.609877203
Root Mean Squared Error:  3644.210148972916
```

In [42]:
```
lasso= Lasso()
lasso.fit(x_train,y_train)
pred_lasso=lasso.predict(x_test)
pred_train=lasso.predict(x_train)
print('R2 Score: ', r2_score(y_test,pred_lasso))
print('R2 on training Data: ', r2_score(y_train,pred_train)*100)
print('Mean Absolute Error: ', mean_absolute_error(y_test,pred_lasso))
print('Mean Squared Error: ', mean_squared_error(y_test,pred_lasso))
print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y_test,pred_lassc
```

```
R2 Score:  0.8095755153106238
R2 on training Data:  73.14465723752146
Mean Absolute Error:  3759.389643549799
Mean Squared Error:  25834498.71321053
Root Mean Squared Error:  5082.764868967532
```

In [43]:
```
ridge= Ridge()
ridge.fit(x_train,y_train)
pred_ridge=ridge.predict(x_test)
pred_train=ridge.predict(x_train)
print('R2 Score: ', r2_score(y_test,pred_ridge))
print('R2 on training Data: ', r2_score(y_train,pred_train)*100)
print('Mean Absolute Error: ', mean_absolute_error(y_test,pred_ridge))
print('Mean Squared Error: ', mean_squared_error(y_test,pred_ridge))
print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y_test,pred_ridge
```

```
R2 Score:  0.8091990653200128
R2 on training Data:  73.14242689385839
Mean Absolute Error:  3772.7274885336424
Mean Squared Error:  25885570.910222866
Root Mean Squared Error:  5087.7864450292
```

In [44]:
```
DTR= DecisionTreeRegressor()
DTR.fit(x_train,y_train)
pred_DTR=DTR.predict(x_test)
pred_train=DTR.predict(x_train)
print('R2 Score: ', r2_score(y_test,pred_DTR))
print('R2 on training Data: ', r2_score(y_train,pred_train)*100)
print('Mean Absolute Error: ', mean_absolute_error(y_test,pred_DTR))
print('Mean Squared Error: ', mean_squared_error(y_test,pred_DTR))
print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y_test,pred_DTR)))
```

```
R2 Score:  0.692535328676606
R2 on training Data:  99.87795868900035
Mean Absolute Error:  2980.210751191045
Mean Squared Error:  41713100.43778772
Root Mean Squared Error:  6458.567986619612
```

In [45]:
```
from sklearn.svm import SVR
svr=SVR()
svr.fit(x_train,y_train)
pred_svr=svr.predict(x_test)
pred_train=svr.predict(x_train)
print('R2 Score: ', r2_score(y_test,pred_svr))
print('R2 on training Data: ', r2_score(y_train,pred_train)*100)
print('Mean Absolute Error: ', mean_absolute_error(y_test,pred_svr))
print('Mean Squared Error: ', mean_squared_error(y_test,pred_svr))
print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y_test,pred_svr)))
```

```
R2 Score:  -0.0558537065410194
R2 on training Data:  -10.663314922990551
Mean Absolute Error:  7897.660732821093
Mean Squared Error:  143245503.68335247
Root Mean Squared Error:  11968.521365789196
```

In [46]:
```
from sklearn.ensemble import ExtraTreesRegressor
etr = ExtraTreesRegressor()
etr.fit(x_train,y_train)
pred_etr=etr.predict(x_test)
pred_train=etr.predict(x_train)
print('R2 Score: ', r2_score(y_test,pred_etr))
print('R2 on training Data: ', r2_score(y_train,pred_train)*100)
print('Mean Absolute Error: ', mean_absolute_error(y_test,pred_etr))
print('Mean Squared Error: ', mean_squared_error(y_test,pred_etr))
print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y_test,pred_etr)))
```

```
R2 Score:  0.8596862524790267
R2 on training Data:  99.87795868900035
Mean Absolute Error:  2260.234043433134
Mean Squared Error:  19036077.927107897
Root Mean Squared Error:  4363.03540291709
```

Cross Validation Score

In [47]:
```
from sklearn.model_selection import cross_val_score
```

In [48]:
```
score =cross_val_score(LR,x,y, cv=5, scoring='r2')
print(score)
print(score.mean())
print("Difference between R2 Score & Cross Validation Score: ", (r2_score(y_tes
```

```
[0.76123487 0.70840689 0.77720769 0.73365562 0.7551376 ]
0.7471285330088384
Difference between R2 Score & Cross Validation Score:  6.244662902878884
```

In [49]:
```
score =cross_val_score(RFR,x,y, cv=5, scoring='r2')
print(score)
print(score.mean())
print("Difference between R2 Score & Cross Validation Score: ", (r2_score(y_tes
```

```
[0.85258371 0.77660455 0.87105448 0.83096793 0.85074724]
0.8363915821612735
Difference between R2 Score & Cross Validation Score:  3.4502495902171693
```

In [50]:
```
score =cross_val_score(KNN,x,y, cv=5, scoring='r2')
print(score)
print(score.mean())
print("Difference between R2 Score & Cross Validation Score: ", (r2_score(y_tes
```

```
[0.14668553 0.03658099 0.03172295 0.13957399 0.16316057]
0.10354480529055962
Difference between R2 Score & Cross Validation Score:  0.4691991238065879
```

In [51]:
```
score =cross_val_score(GBR,x,y, cv=5, scoring='r2')
print(score)
print(score.mean())
print("Difference between R2 Score & Cross Validation Score: ", (r2_score(y_tes
```

```
[0.87437676 0.79922506 0.8938781  0.85052066 0.86054906]
0.8557099265578053
Difference between R2 Score & Cross Validation Score:  4.640204611017185
```

In [52]:
```
score =cross_val_score(lasso,x,y, cv=5, scoring='r2')
print(score)
print(score.mean())
print("Difference between R2 Score & Cross Validation Score: ", (r2_score(y_tes
```

```
[0.76125678 0.70843568 0.77718717 0.73366564 0.7551367 ]
0.7471363933595827
Difference between R2 Score & Cross Validation Score:  6.243912195104107
```

In [53]:
```
score =cross_val_score(ridge,x,y, cv=5, scoring='r2')
print(score)
print(score.mean())
print("Difference between R2 Score & Cross Validation Score: ", (r2_score(y_tes
```

```
[0.76139416 0.70872916 0.77656529 0.7340199  0.75488218]
0.7471181375337135
Difference between R2 Score & Cross Validation Score:  6.208092778629936
```

In [54]:
```
score =cross_val_score(DTR,x,y, cv=5, scoring='r2')
print(score)
print(score.mean())
print("Difference between R2 Score & Cross Validation Score: ", (r2_score(y_tes
```

```
[0.75566105 0.7002571  0.74743149 0.72306662 0.67121963]
0.7195271796070898
Difference between R2 Score & Cross Validation Score:  -2.699185093048373
```

In [55]:
```
score =cross_val_score(svr,x,y, cv=5, scoring='r2')
print(score)
print(score.mean())
print("Difference between R2 Score & Cross Validation Score: ", (r2_score(y_tes
```

```
[-0.11521827 -0.10975599 -0.08194453 -0.10398097 -0.10781598]
-0.10374314816465749
Difference between R2 Score & Cross Validation Score:  4.788944162363809
```

In [56]:
```
score =cross_val_score(etr,x,y, cv=5, scoring='r2')
print(score)
print(score.mean())
print("Difference between R2 Score & Cross Validation Score: ", (r2_score(y_tes
```

```
[0.83541517 0.74166917 0.84596636 0.81276096 0.8436369 ]
0.8158897127779184
Difference between R2 Score & Cross Validation Score:  4.37965397011083
```

Observation: From the difference of both R2 & Cross validation score computed on R2 score we can conclude that Gradient Boosting Regressor is our best fitting & best performing model

Saving The best model

In [59]:
```
import pickle
filename ="Medical_cost_insurance.pkl"
pickle.dump(GBR, open(filename, 'wb'))
```

In [60]:
```
#Loading Model
load_model=pickle.load(open('Medical_cost_insurance.pkl','rb'))
result=load_model.score(x_test,y_test)
print(result*100)
```

```
90.21119726679765
```

In [61]:
```
conclusion=pd.DataFrame([load_model.predict(x_test)[:],y_test[:]],index=['Predi
conclusion
```

Out[61]:

	0	1	2	3	4	5	
Predicted	35631.827377	5997.99314	8205.048588	5952.071914	10005.733549	12387.810493	6132
Original	36397.576000	4415.15880	7639.417450	2304.002200	9563.029000	11454.021500	5012

2 rows × 335 columns

In [ ]: