VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CSE4036

MACHINE LEARNING

MEDICAL INSURANCE COST PREDICTION

GROUP-9 TEAM MEMBERS

P B S S JASWANTH – 19MIA1039

S. VAMSIDHAR – 19MIA1074

MANSOOR KHAN LODI – 19MIA1094

SHASHANK RAVOOR – 19MIA1105

FACULTY

NAME: BHARGAVI R

SLOT: F1

TABLE OF CONTENTS

ABSTRACT

- Covid-19 pandemic has been difficult in various aspects. It has made individuals realize the significance of health and taught people how to take sound care of their health and the well-being of their loved ones.

- Since India is one of the top countries that has worst hit by the pandemic, the health insurance sector has witnessed a huge surge due to increased demand. Individuals have now started getting worried about the effect of a medical emergency on their wallet, and this is the reason why they are spending to buy comprehensive health insurance.

- The main purpose of medical insurance is to receive the best medical care without any strain on your finances. Health insurance plans offer protection against high medical costs. It covers hospitalization expenses, day care procedures and ambulance charges; besides many others. You may, therefore, focus on your speedy recovery instead of worrying about such high costs.

OBJECTIVE

1) Many factors that affect how much we pay for health insurance are not within our control. Here will be focusing on the factors like age, gender, BMI, children, smoking status, and region that affect how much health insurance premiums cost and we will be finding out the "*feature importance ranking*".

2) Now with more number of people opting for health insurance, we will be evaluating the performance of different regression models and we will try to find out the ''*predicted charges*'' for the insurance and compare it with the actual output charges.

DATASET

The dataset we had referred with the help of Kaggle platform is “Medical Insurance Cost Prediction Dataset”.

*Here’s the dataset **g-drive link**:*

https://drive.google.com/file/d/17FXzyymGAvcnybg67cnal2HKh7mnSNgL/view?usp=sharing

METHODOLOGY

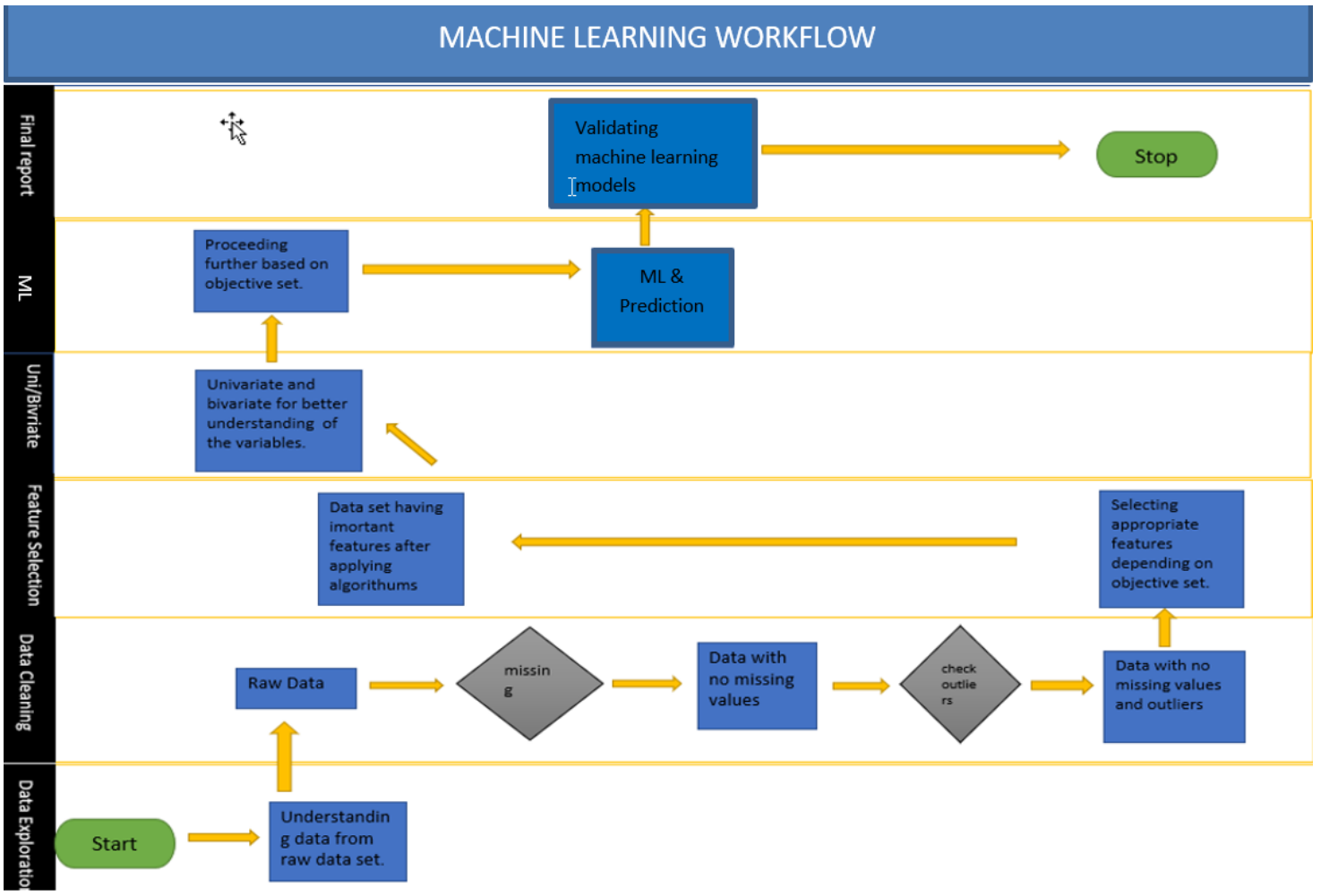
- Initially we will be doing EDA for our dataset. This includes analyzing the features and selecting important features based on statistical methods.

- Next, we will be dealing with the presence of missing values and outliers which will have a direct & adverse impact in machine learning models. So, we will be removing them as well.

- We will be performing univariate analysis, plots, feature importance ranking.

- We will be using linear regression models to estimate the medical insurance cost.

MACHINE LEARNING WORKFLOW



CODE

- The coding part starts with the Data – Preprocessing part, where we first understand the data and cleanse the data by removing outliers.

- First, we import the libraries required for this project.

In [1]:
```
# Basic Imports
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import *
from statistics import mean
from scipy.stats import bootstrap
```

- Then we understand the data by getting an overall summary of the data, getting the unique values and checking whether there are null values in the data or not.

- We used Label Encoder to convert textual data into numerical data in order to facilitate outlier removal as outlier removal requires data to be in numerical format only.

In [5]:
```
le1 = LabelEncoder()
le2 = LabelEncoder()
le3 = LabelEncoder()
```

In [6]:
```
data['sex'] = le1.fit_transform(data['sex'])
data['smoker'] = le2.fit_transform(data['smoker'])
data['region'] = le3.fit_transform(data['region'])
```

In [7]:
```
data.head(10)
```

Out[7]:

	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
5	31	0	25.740	0	0	2	3756.62160
6	46	0	33.440	1	0	2	8240.58960
7	37	0	27.740	3	0	1	7281.50560
8	37	1	29.830	2	0	0	6406.41070
9	60	0	25.840	0	0	1	28923.13692

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

In [11]:
```
data.describe()
```

Out[11]:

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

In [12]:
```
data.nunique()
```

Out[12]:
```
age           47
sex            2
bmi          548
children       6
smoker         2
region         4
charges     1337
dtype: int64
```

In [13]:
```
data['age'].unique()
```

Out[13]:
```
array([19, 18, 28, 33, 32, 31, 46, 37, 60, 25, 62, 23, 56, 27, 52, 30, 34,
       59, 63, 55, 22, 26, 35, 24, 41, 38, 36, 21, 48, 40, 58, 53, 43, 64,
       20, 61, 44, 57, 29, 45, 54, 49, 47, 51, 42, 50, 39], dtype=int64)
```

In [14]:
```
data['sex'].unique()
```

Out[14]:
```
array([0, 1])
```

In [15]:
```
data['bmi'].unique()
```

Out[15]:
```
array([27.9  , 33.77 , 33.   , 22.705, 28.88 , 25.74 , 33.44 , 27.74 ,
       29.83 , 25.84 , 26.22 , 26.29 , 34.4  , 39.82 , 42.13 , 24.6  ,
       30.78 , 23.845, 40.3  , 35.3  , 36.005, 32.4  , 34.1  , 31.92 ,
       28.025, 27.72 , 23.085, 32.775, 17.385, 36.3  , 35.6  , 26.315,
       28.6  , 28.31 , 36.4  , 20.425, 32.965, 20.8  , 36.67 , 39.9  ,
       26.6  , 36.63 , 21.78 , 30.8  , 37.05 , 37.3  , 38.665, 34.77 ,
       24.53 , 35.2  , 35.625, 33.63 , 28.   , 34.43 , 28.69 , 36.955,
       31.825, 31.68 , 22.88 , 37.335, 27.36 , 33.66 , 24.7  , 25.935,
       22.42 , 28.9  , 39.1  , 36.19 , 23.98 , 24.75 , 28.5  , 28.1  ,
       32.01 , 27.4  , 34.01 , 29.59 , 35.53 , 39.805, 26.885, 38.285,
```

In [16]:
```
data['children'].unique()
```

Out[16]:
```
array([0, 1, 3, 2, 5, 4], dtype=int64)
```

In [17]:
```
data['smoker'].unique()
```

Out[17]:
```
array([1, 0])
```

In [18]:
```
data['region'].unique()
```

Out[18]:
```
array([3, 2, 1, 0])
```

In [19]:
```
data['charges'].unique()
```

Out[19]:
```
array([16884.924 ,  1725.5523,  4449.462 , ...,  1629.8335,  2007.945 ,
       29141.3603])
```

- Then we remove the outliers using the Local Outlier Factor (LOF) method.

In [20]:
```
# Treatment of Outliers
```

In [8]:
```
lof = LocalOutlierFactor()
yhat = lof.fit_predict(data)
mask = yhat != -1
data = data[mask]
```

In [9]:
```
data.shape
```

Out[9]:
```
(1318, 7)
```

- Then we convert the previously converted textual data back into textual form for making data visualizations and analysis easily interpretable.

In [23]:
```
# Now we will perform inverse_transform for visual plots to convert them back to string.
```

In [10]:
```
data['sex'] = le1.inverse_transform(data['sex'])
```

In [11]:
```
data
```

Out[11]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	1	3	16884.92400
1	18	male	33.770	1	0	2	1725.55230
2	28	male	33.000	3	0	2	4449.46200
3	33	male	22.705	0	0	1	21984.47061
4	32	male	28.880	0	0	1	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	0	1	10600.54830
1334	18	female	31.920	0	0	0	2205.98080
1335	18	female	36.850	0	0	2	1629.83350
1336	21	female	25.800	0	0	3	2007.94500
1337	61	female	29.070	0	1	1	29141.36030

1318 rows × 7 columns

In [12]:
```
data['smoker'] = le2.inverse_transform(data['smoker'])
```

In [13]:
```
data
```

Out[13]:

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

1318 rows × 7 columns

In [14]:
```
data['region'] = le3.inverse_transform(data['region'])
```

In [15]:
```
data
```

Out[15]:

	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

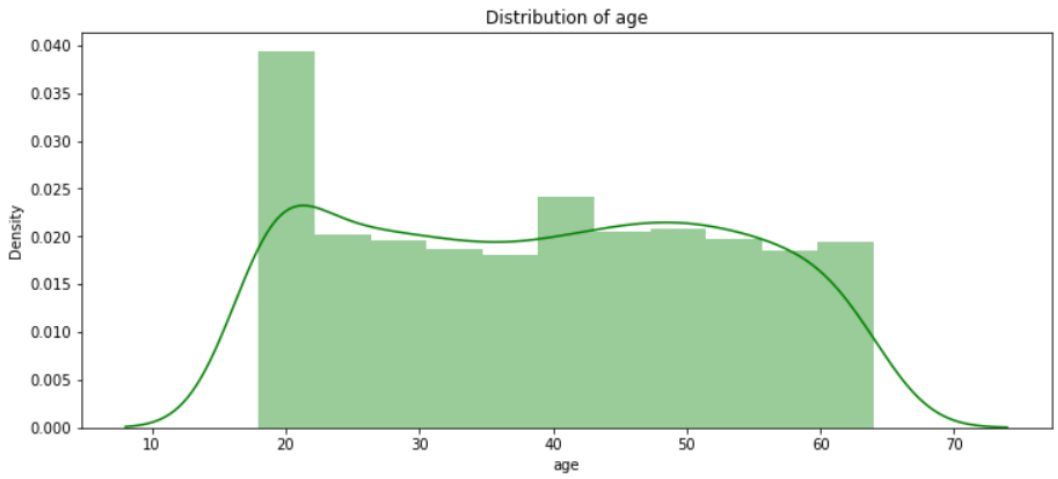
1318 rows × 7 columns

- Now, we perform Univariate Analysis on the data to understand the distribution of each and every continuous variable.

In [30]:
```
# Univariate Analysis
```

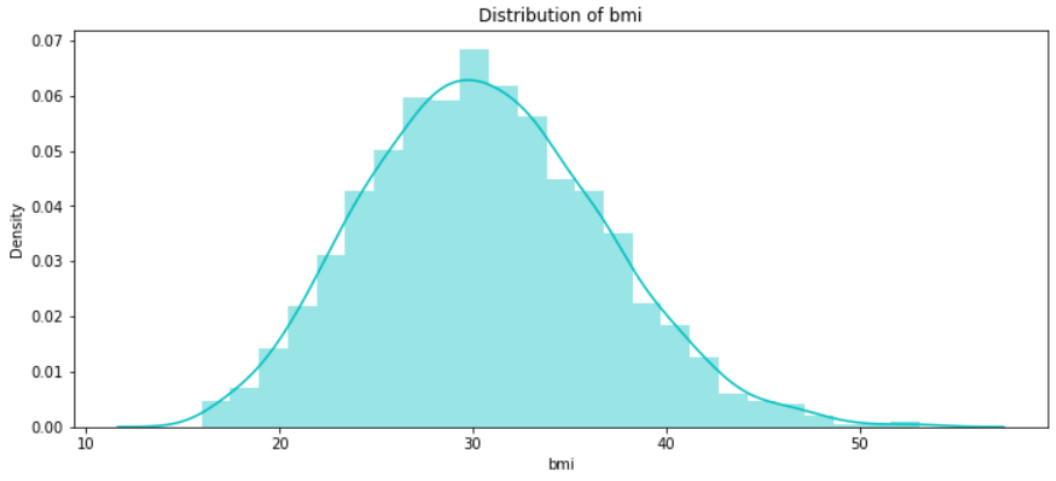
In [31]:
```
plt.figure(figsize=(12,5))
plt.title("Distribution of age")
ax = sns.distplot(data["age"], color = 'g')
```

```
D:\Annaconda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be re
moved in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or
`histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



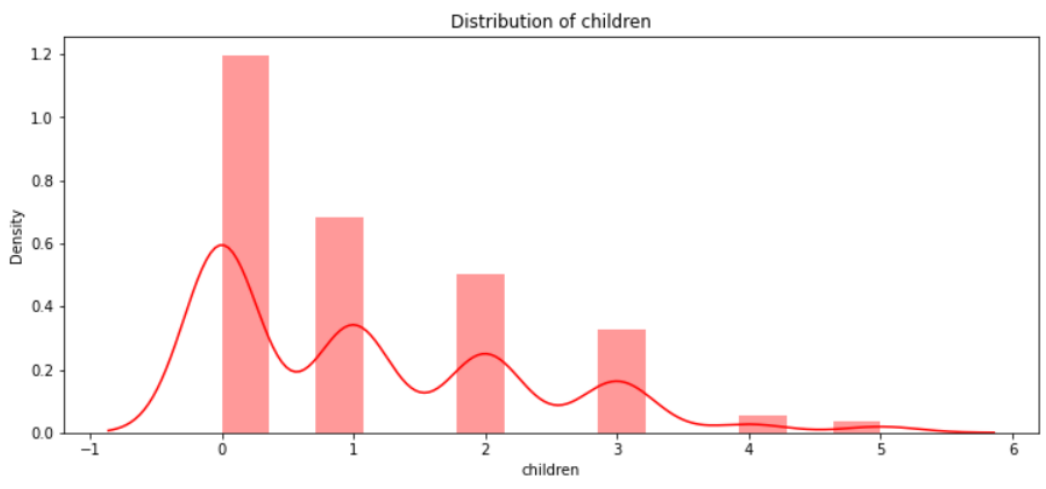
In [32]:
```
plt.figure(figsize=(12,5))
plt.title("Distribution of bmi")
ax = sns.distplot(data["bmi"], color = 'c')
```

```
D:\Annaconda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be re
moved in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or
`histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



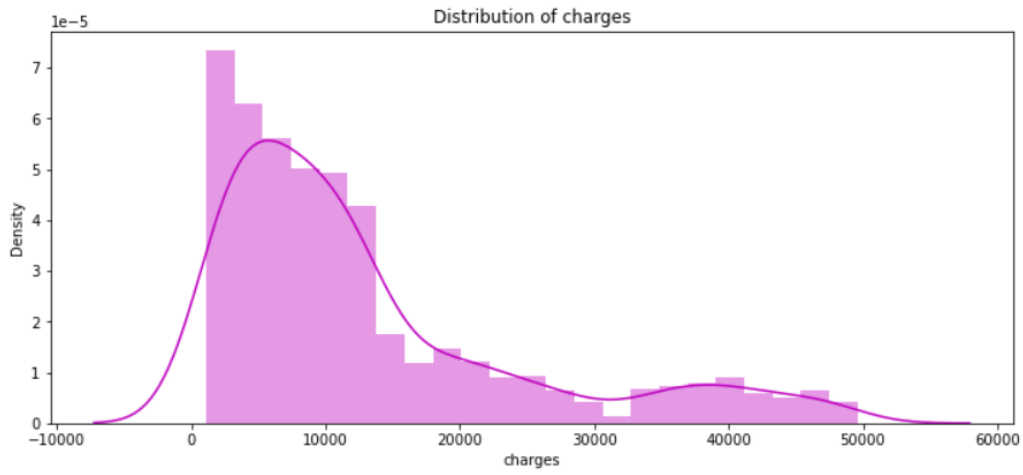
In [33]:
```
plt.figure(figsize=(12,5))
plt.title("Distribution of children")
ax = sns.distplot(data["children"], color = 'r')
```

```
D:\Annaconda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be re
moved in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or
`histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



```
In [34]: plt.figure(figsize=(12,5))
         plt.title("Distribution of charges")
         ax = sns.distplot(data["charges"], color = 'm')
```

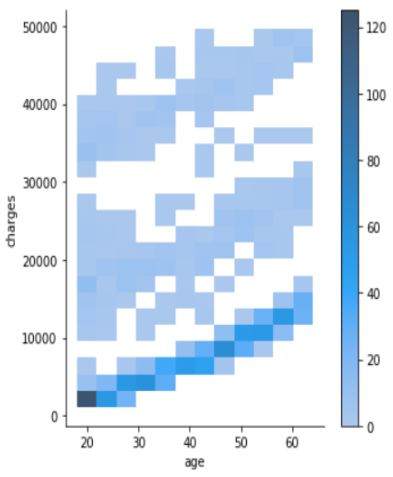
```
D:\Annaconda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be re
moved in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or
`histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



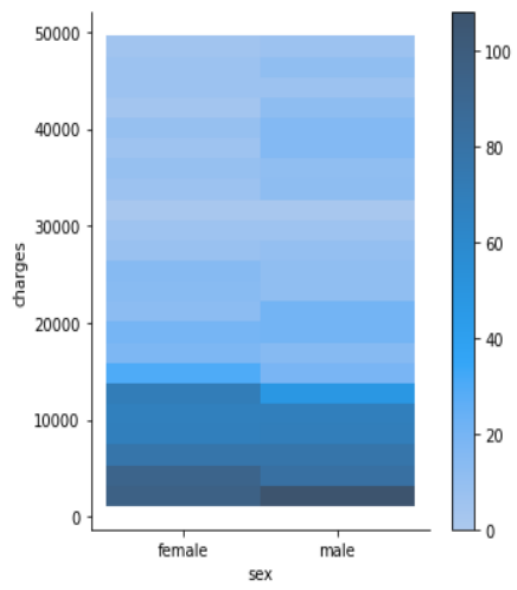
- After Univariate analysis we are going to perform Bivariate analysis using different plots on the data to understand the relation between twovariables in the data where charges is the target variable and remaining variables are feature variables.

1. Heat Map:

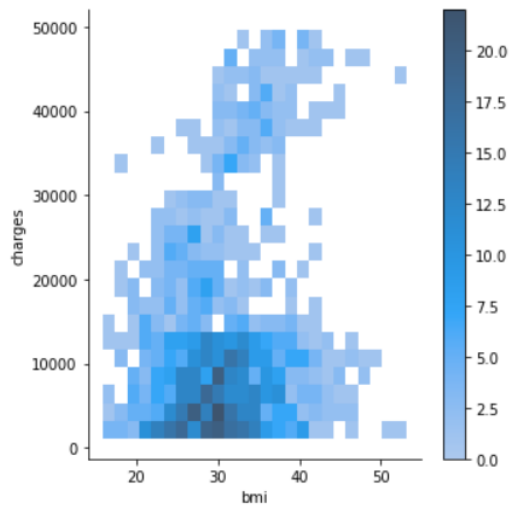
```
In [35]: #Bivariate Analysis
         #Taking charges as target variable (y) and remaining all as feature variables (x)

In [36]: # 1. Heat Map

In [37]: sns.displot(x="age", y="charges",data=data,cbar=True)

Out[37]: <seaborn.axisgrid.FacetGrid at 0x2138ddfe1f0>
```



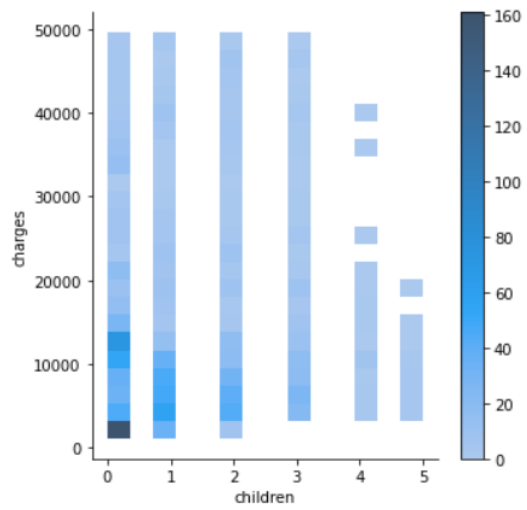

```
In [38]: sns.displot(x="sex", y="charges",data=data,cbar=True)

Out[38]: <seaborn.axisgrid.FacetGrid at 0x2138dc93730>
```



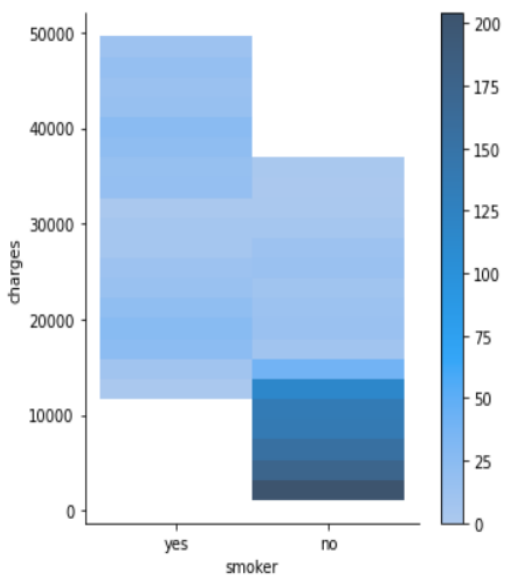
```
In [39]: sns.displot(x="bmi", y="charges",data=data,cbar=True)
Out[39]: <seaborn.axisgrid.FacetGrid at 0x2138b43d2b0>
```



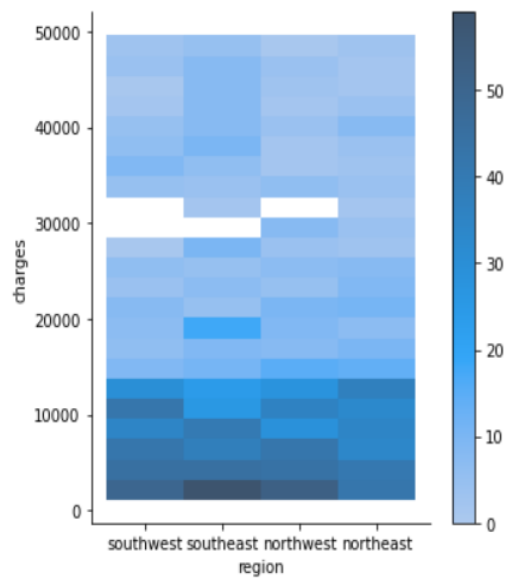

```
In [40]: sns.displot(x="children", y="charges",data=data,cbar=True)
Out[40]: <seaborn.axisgrid.FacetGrid at 0x2138de0d460>
```




```
In [41]: sns.displot(x="smoker", y="charges",data=data,cbar=True)
Out[41]: <seaborn.axisgrid.FacetGrid at 0x2138f0d2b20>
```



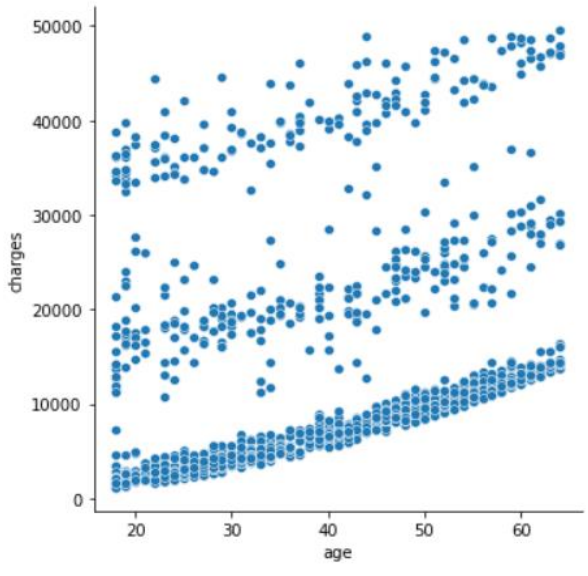

```
In [42]: sns.displot(x="region", y="charges",data=data,cbar=True)
Out[42]: <seaborn.axisgrid.FacetGrid at 0x2138efd9910>
```



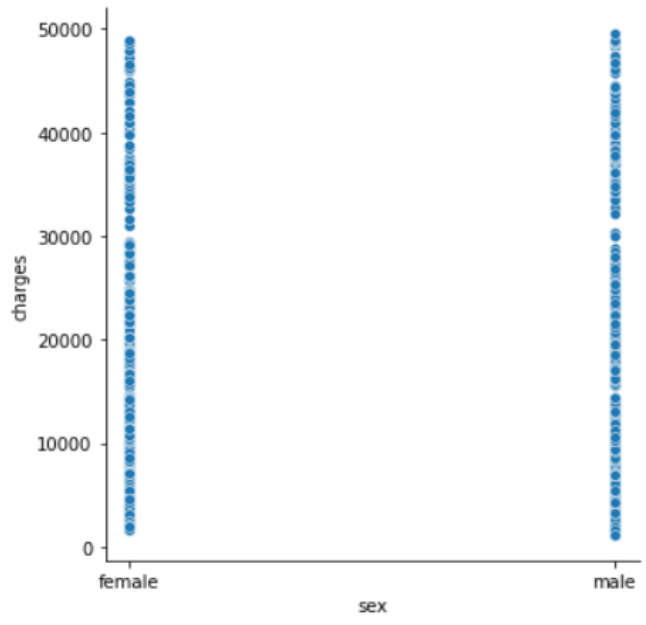
2. Scatter Plot:

```
In [43]: # 2.Scatter Plot
```

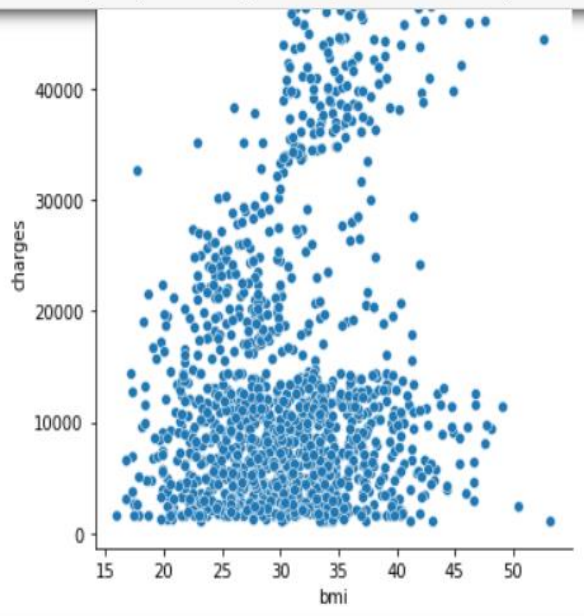
```
In [44]: sns.relplot(x='age',y='charges',data=data)
Out[44]: <seaborn.axisgrid.FacetGrid at 0x2138f1a2220>
```



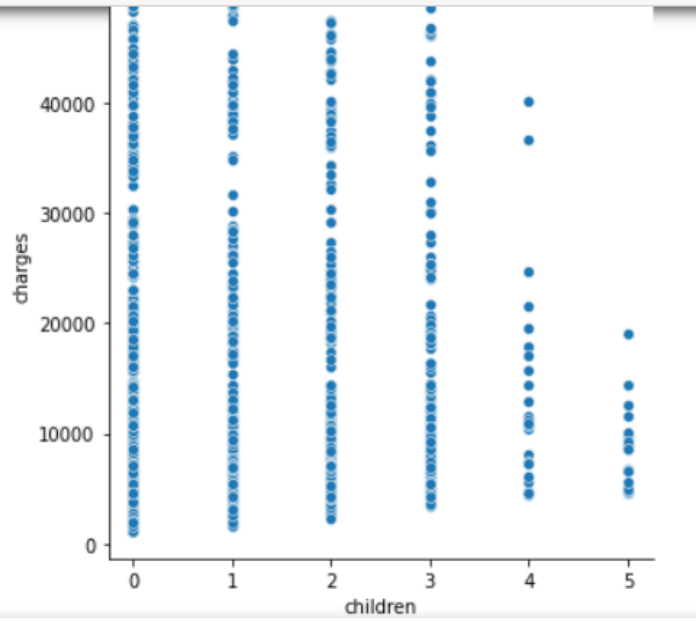

```
In [45]: sns.relplot(x='sex',y='charges',data=data)
Out[45]: <seaborn.axisgrid.FacetGrid at 0x2138f2aabb0>
```



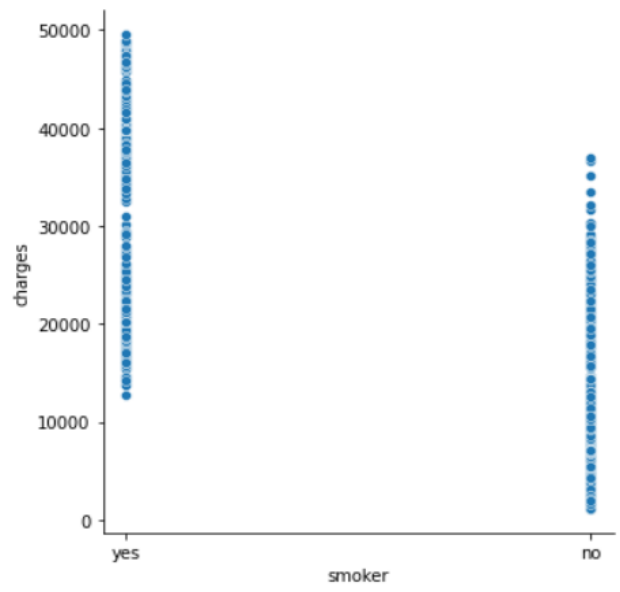
```
In [46]: sns.relplot(x='bmi',y='charges',data=data)
```



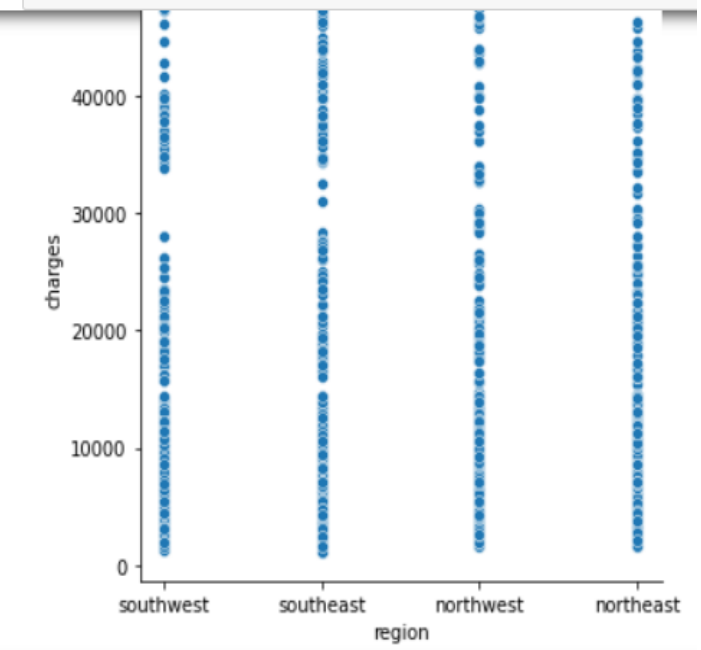

```
In [47]: sns.relplot(x='children',y='charges',data=data)
```




```
In [48]: sns.relplot(x='smoker',y='charges',data=data)
Out[48]: <seaborn.axisgrid.FacetGrid at 0x2138dc66760>
```



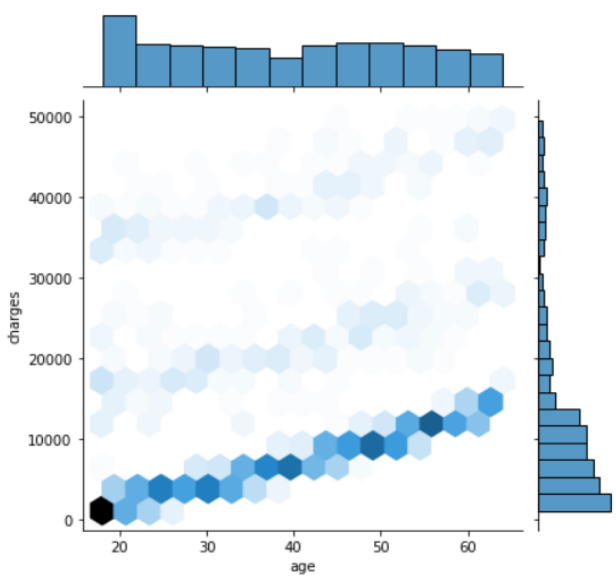

```
In [49]: sns.relplot(x='region',y='charges',data=data)
```



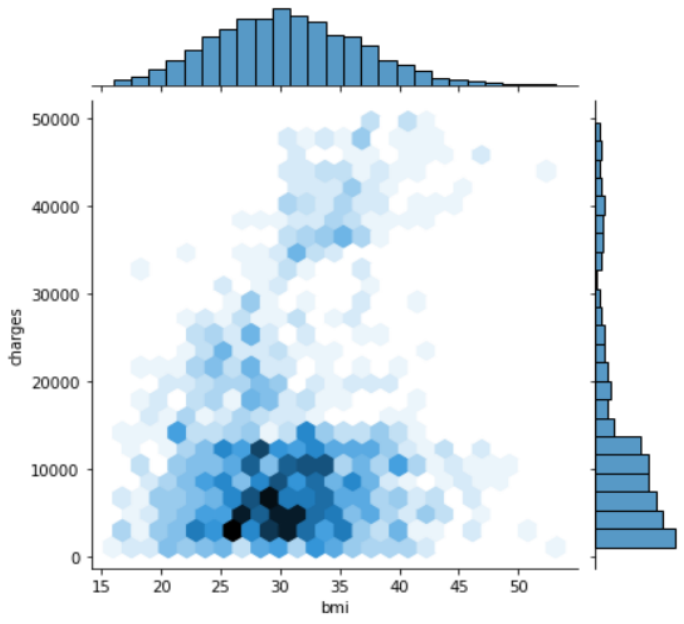
2. Hex Plot:

```
In [50]: # 3.Hex plot
```

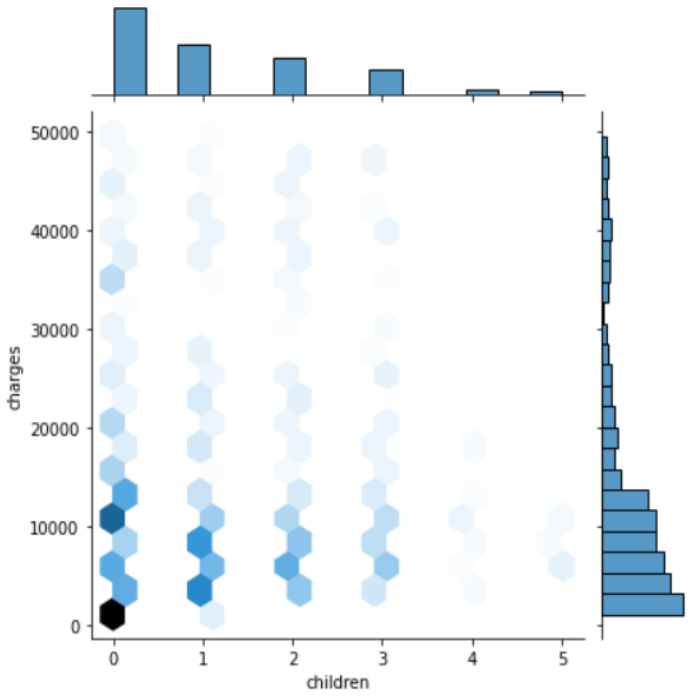
```
In [51]: sns.jointplot(x='age', y='charges', kind="hex",data=data)
Out[51]: <seaborn.axisgrid.JointGrid at 0x2138b47f970>
```




```
In [52]: sns.jointplot(x='bmi', y='charges', kind="hex",data=data)
Out[52]: <seaborn.axisgrid.JointGrid at 0x2138b43d370>
```

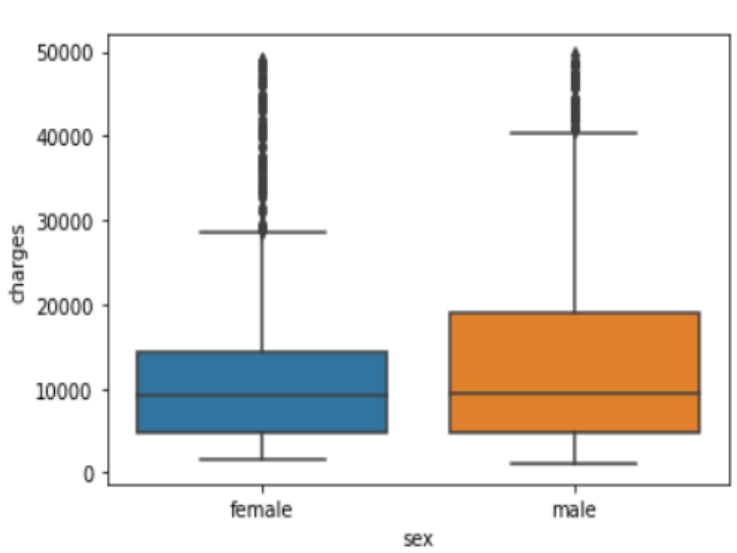


```
In [53]: sns.jointplot(x='children', y='charges', kind="hex",data=data)
Out[53]: <seaborn.axisgrid.JointGrid at 0x213905a4dc0>
```

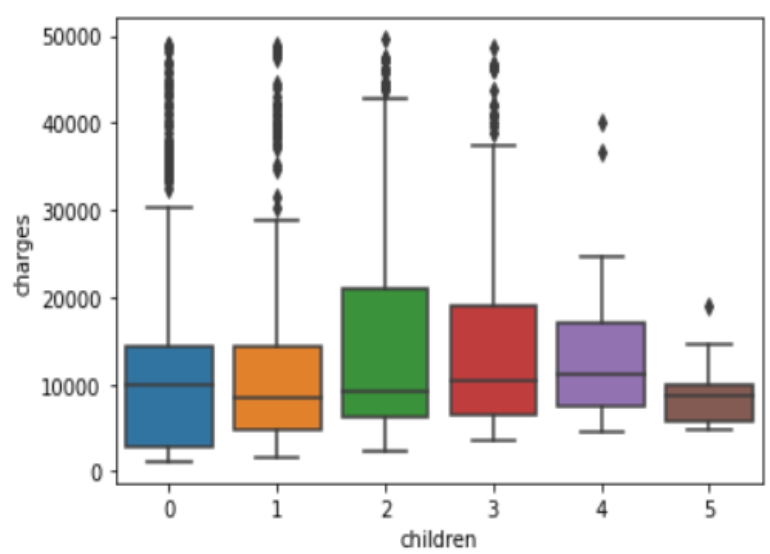


4. Box plot:

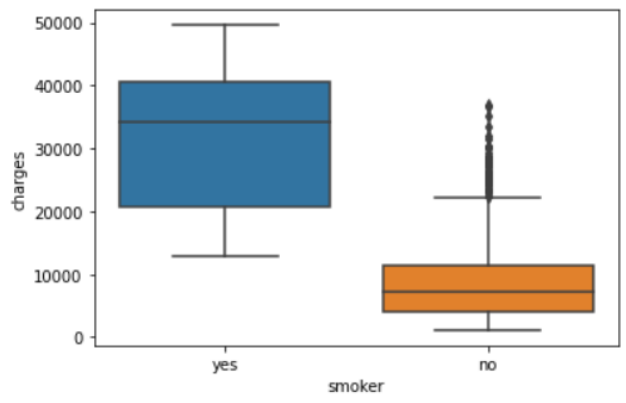
```
In [56]: sns.boxplot(x="sex", y="charges", data=data)
Out[56]: <AxesSubplot:xlabel='sex', ylabel='charges'>
```



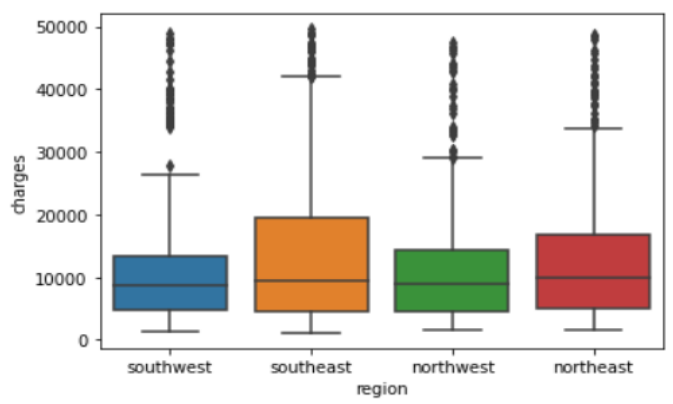

```
In [58]: sns.boxplot(x="children", y="charges", data=data)
Out[58]: <AxesSubplot:xlabel='children', ylabel='charges'>
```



```
In [59]: sns.boxplot(x="smoker", y="charges", data=data)
Out[59]: <AxesSubplot:xlabel='smoker', ylabel='charges'>
```

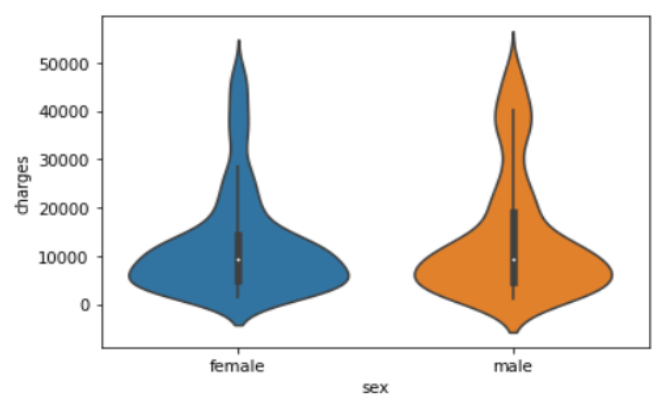



```
In [60]: sns.boxplot(x="region", y="charges", data=data)
Out[60]: <AxesSubplot:xlabel='region', ylabel='charges'>
```

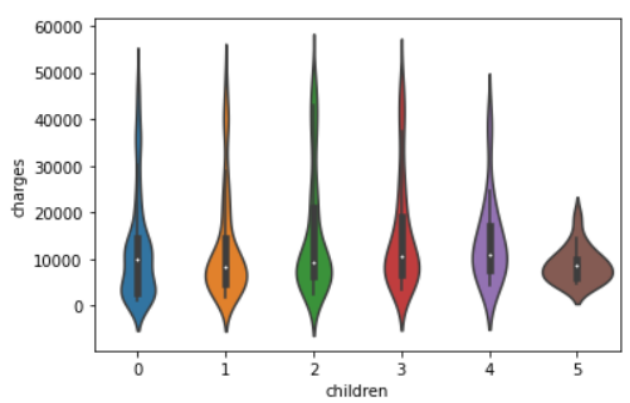


5. Violin Plot:

```
In [63]: sns.violinplot(x="sex", y="charges", data=data)
Out[63]: <AxesSubplot:xlabel='sex', ylabel='charges'>
```

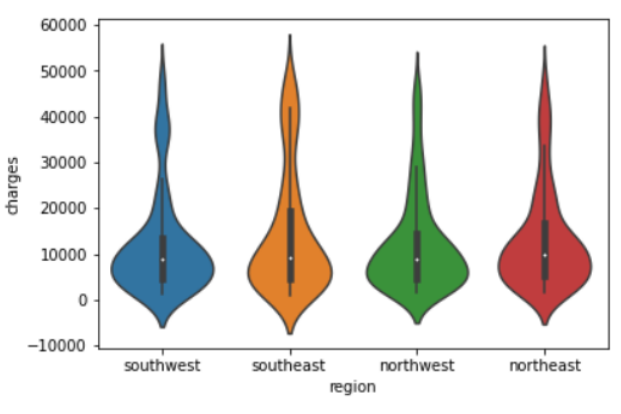



```
In [65]: sns.violinplot(x="children", y="charges", data=data)
Out[65]: <AxesSubplot:xlabel='children', ylabel='charges'>
```




```
In [66]: sns.violinplot(x="smoker", y="charges", data=data)
Out[66]: <AxesSubplot:xlabel='smoker', ylabel='charges'>
```

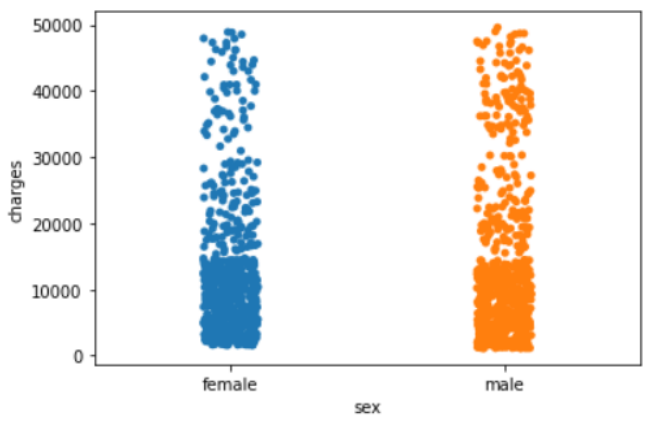

```
In [67]: sns.violinplot(x="region", y="charges", data=data)
Out[67]: <AxesSubplot:xlabel='region', ylabel='charges'>
```



6. Stripp plot:

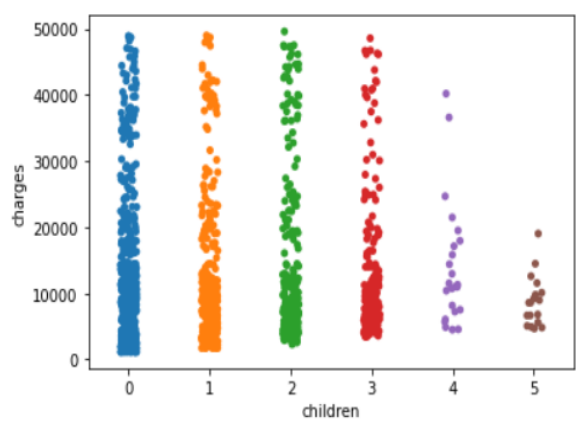
```
In [70]: sns.stripplot(x = 'sex', y = 'charges', data = data)
```

```
Out[70]: <AxesSubplot:xlabel='sex', ylabel='charges'>
```



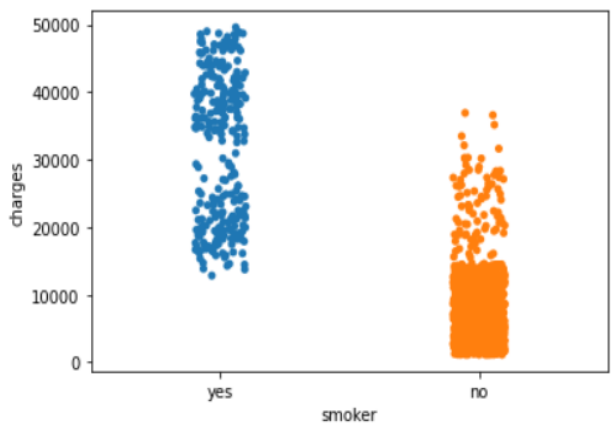

```
In [72]: sns.stripplot(x = 'children', y = 'charges', data = data)
```

```
Out[72]: <AxesSubplot:xlabel='children', ylabel='charges'>
```



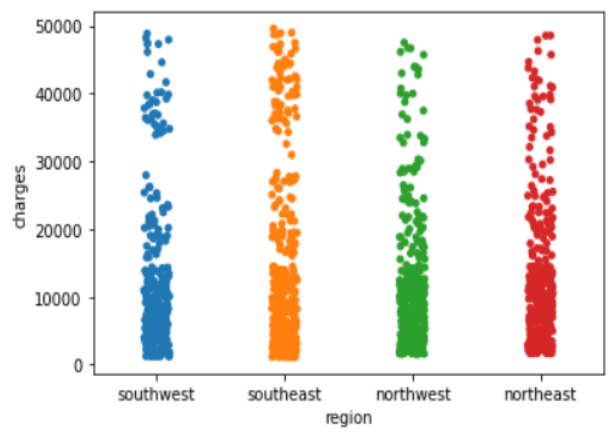

```
In [73]: sns.stripplot(x = 'smoker', y = 'charges', data = data)
```

```
Out[73]: <AxesSubplot:xlabel='smoker', ylabel='charges'>
```




```
In [74]: sns.stripplot(x = 'region', y = 'charges', data = data)
```

```
Out[74]: <AxesSubplot:xlabel='region', ylabel='charges'>
```



After Bivariate analysis we are going to perform Multivariate analysis using different plots on the data to understand the relation between different in the data.

1. Correlation Plot:

```
In [75]: #Multivariate analysis
```

```
In [76]: # 1.Correlation heat plot
```

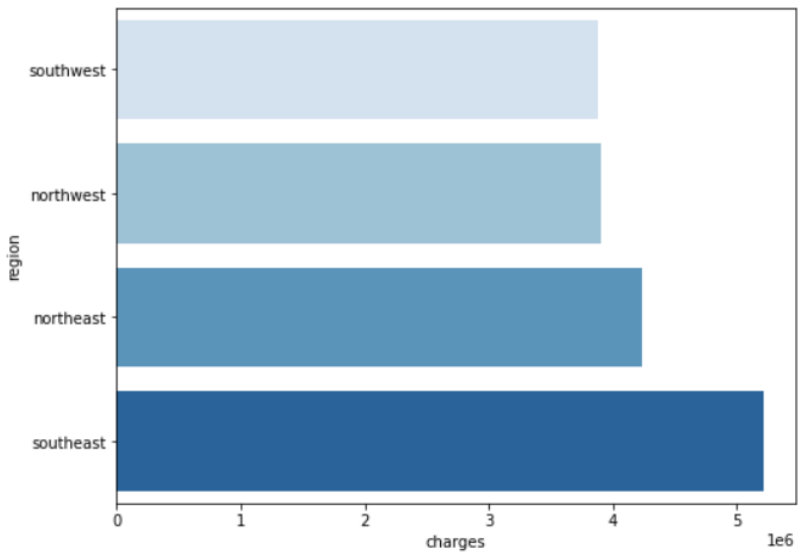
```
In [77]: corelation=data.corr()
         sns.heatmap(corelation,xticklabels=corelation.columns,yticklabels=corelation.columns,annot=True)
```

```
Out[77]: <AxesSubplot:>
```

charges by region: -

```
charges = data['charges'].groupby(data.region).sum().sort_values(ascending = True)
f, ax = plt.subplots(1, 1, figsize=(8, 6))
ax = sns.barplot(charges.head(), charges.head().index, palette='Blues')
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following v
be `data`, and passing other arguments without an explicit keyword will result in an error or misinte
  FutureWarning
```

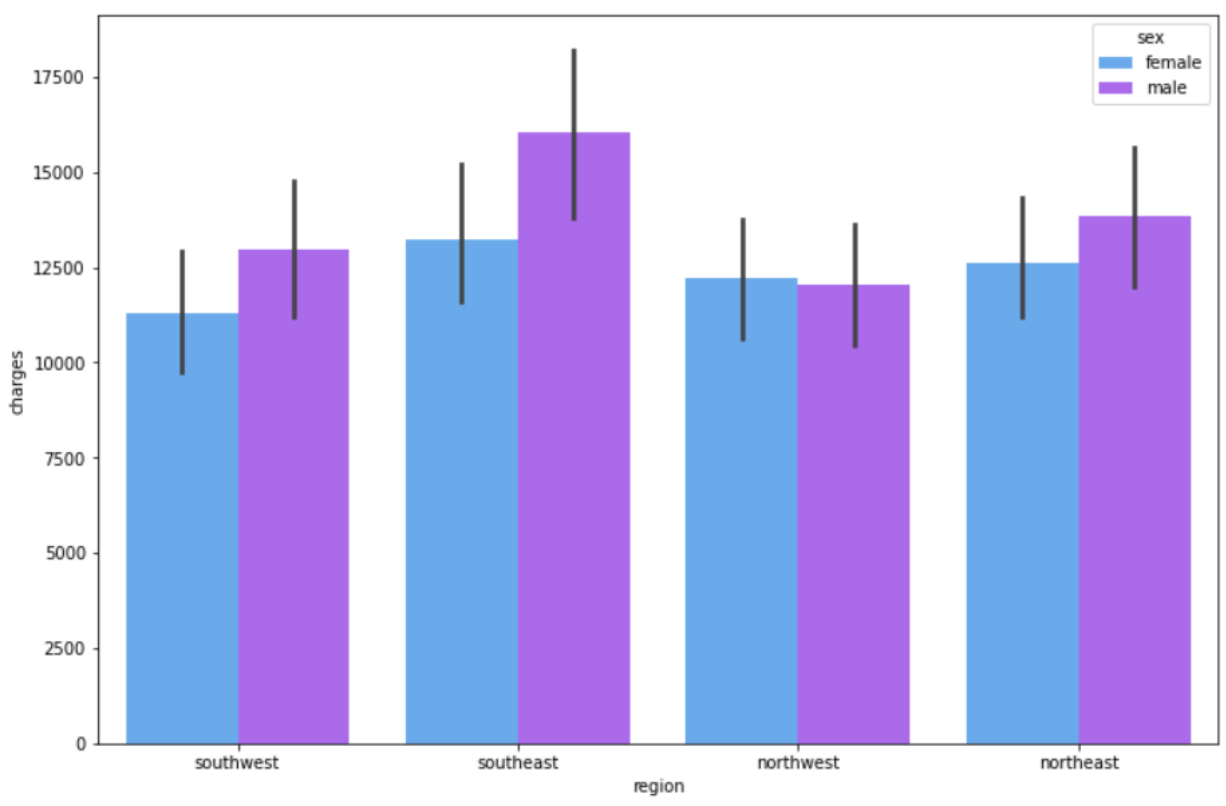


- So overall the highest medical charges are in the Southeast and the lowest are in the Southwest.

Taking into account certain factors (sex, smoking, having children) let's see how it changes by region -

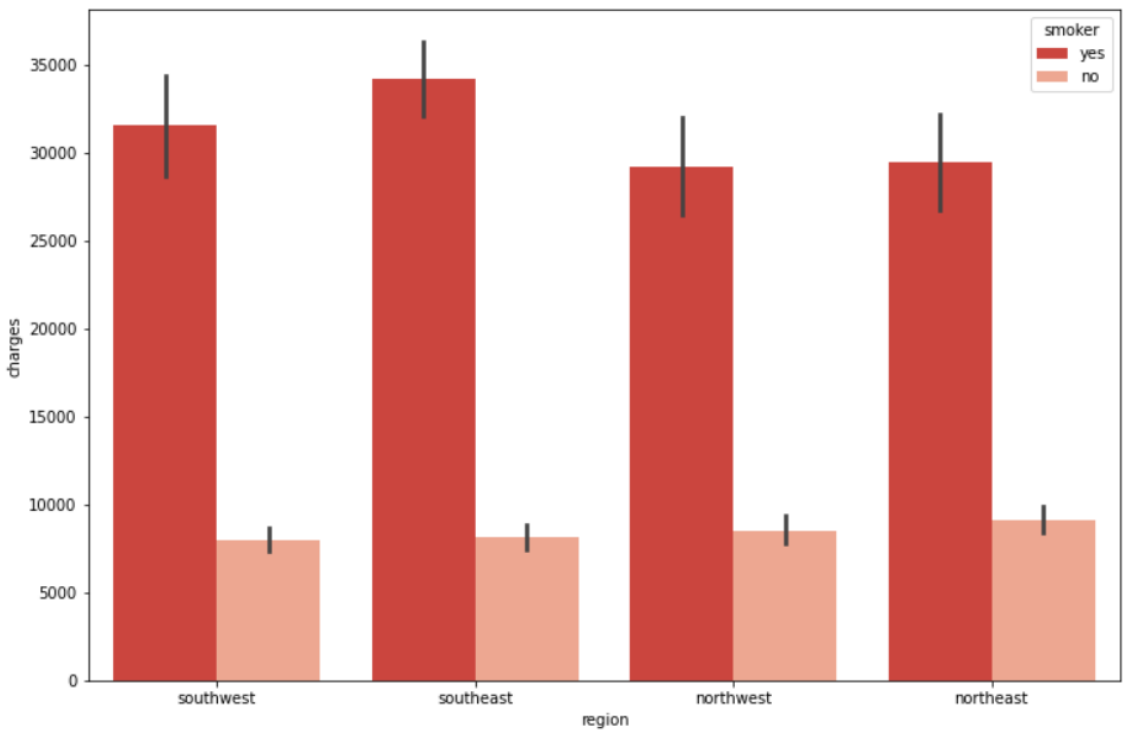
- **Factor = 'sex'**

```
f, ax = plt.subplots(1, 1, figsize=(12, 8))
ax = sns.barplot(x='region', y='charges', hue='sex', data=data, palette='cool')
```



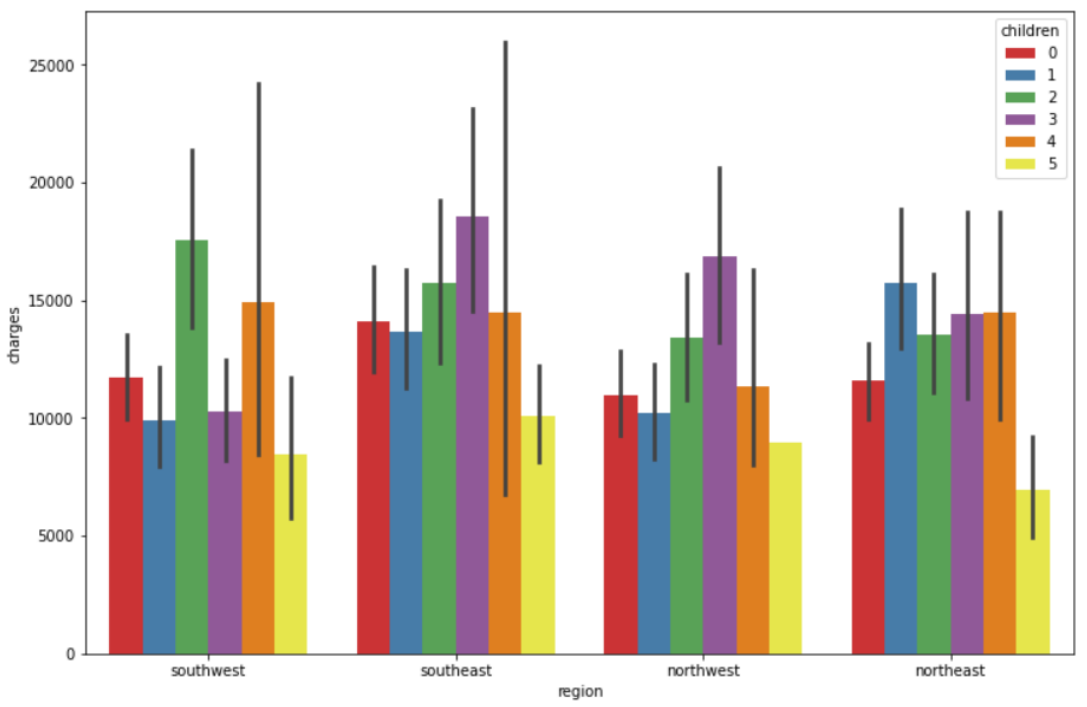
- **Factor = 'smoker'**

```
f, ax = plt.subplots(1,1, figsize=(12,8))
ax = sns.barplot(x = 'region', y = 'charges',
                 hue='smoker', data=data, palette='Reds_r')
```



- **Factor = 'children'**

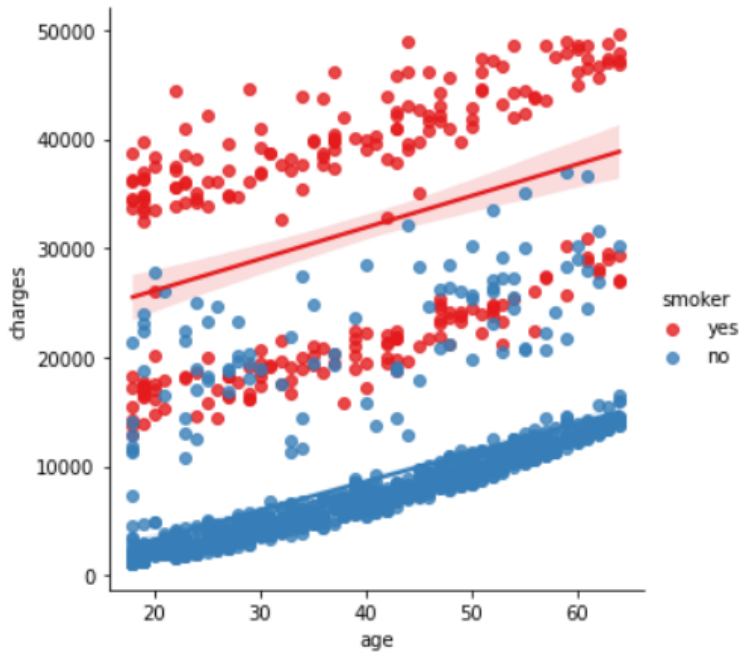
```
f, ax = plt.subplots(1, 1, figsize=(12, 8))
ax = sns.barplot(x='region', y='charges', hue='children', data=data, palette='Set1')
```

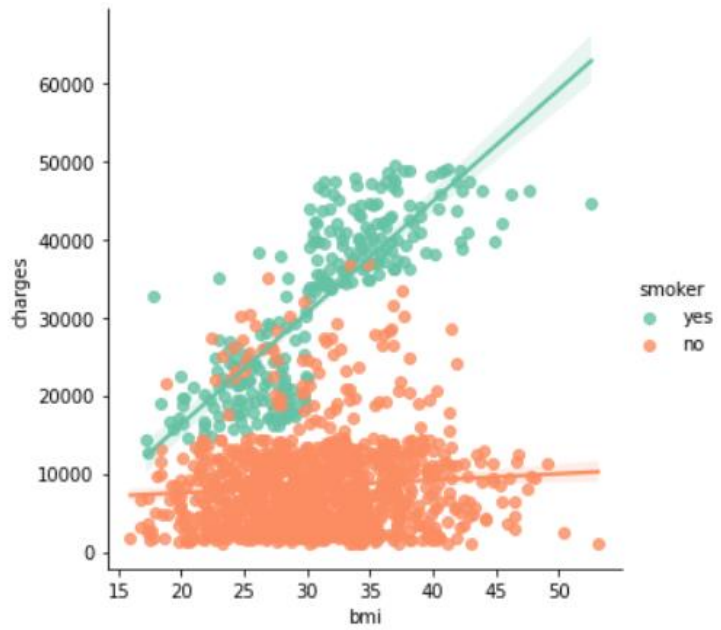


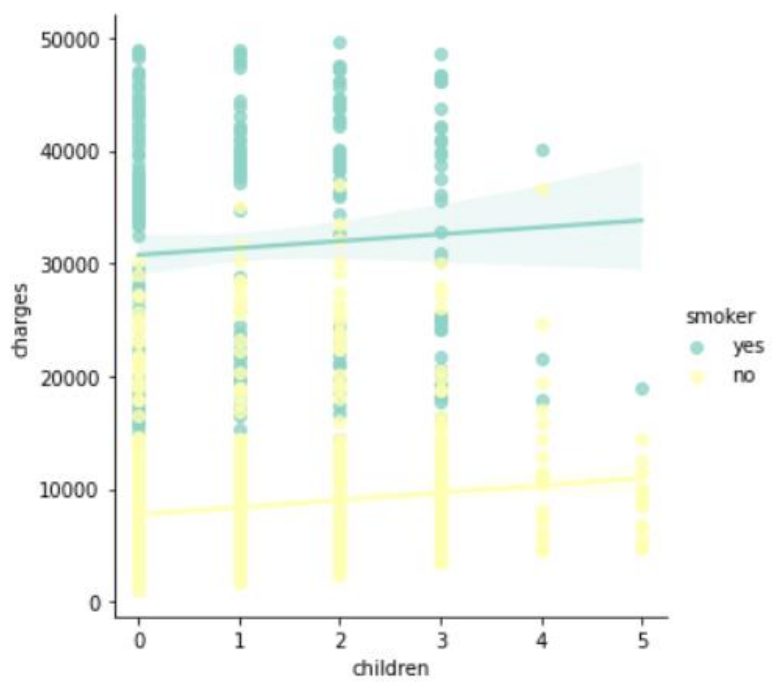
- As we can see from these bar plots the highest charges due to smoking are still in the Southeast but the lowest are in the Northeast.
- People in the Southwest generally smoke more than people in the Northeast, but people in the Northeast have higher charges by gender than in the Southwest and Northwest overall. And people with children tend to have higher medical costs overall as well.

Now let's analyze the medical charges by age, bmi and children according to the smoking factor –

```
ax = sns.lmplot(x = 'age', y = 'charges', data=data, hue='smoker', palette='Set1')
ax = sns.lmplot(x = 'bmi', y = 'charges', data=data, hue='smoker', palette='Set2')
ax = sns.lmplot(x = 'children', y = 'charges', data=data, hue='smoker', palette='Set3')
```







- Smoking has the highest impact on medical costs, even though the costs are growing with age, bmi and children. Also, people who have children generally smoke less.

Again, we will be using LabelEncoder() inorder to convert them back to numerics

```
data['sex'] = le1.fit_transform(data['sex'])
data['smoker'] = le2.fit_transform(data['smoker'])
data['region'] = le3.fit_transform(data['region'])
```

Now after applying LabelEncoder, we will now do the train_test_split on the data

```
from sklearn.model_selection import train_test_split as holdout
from sklearn import metrics

x = data.drop(['charges'], axis = 1)
y = data['charges']
x_train, x_test, y_train, y_test = holdout(x, y, test_size=0.2, random_state=0)
```

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor as rfr
x = data.drop(['charges'], axis=1)
y = data.charges
Rfr = rfr(n_estimators = 100, criterion = 'mse',
                              random_state = 1,
                              n_jobs = -1, oob_score=True)
Rfr.fit(x_train,y_train)
x_train_pred = Rfr.predict(x_train)
x_test_pred = Rfr.predict(x_test)

print('MSE train data: %.3f, MSE test data: %.3f' %
      (metrics.mean_squared_error(x_train_pred, y_train),
       metrics.mean_squared_error(x_test_pred, y_test)))
print('R2 train data: %.3f, R2 test data: %.3f' %
      (metrics.r2_score(y_train,x_train_pred, y_train),
       metrics.r2_score(y_test,x_test_pred, y_test)))

print('OOB score:', Rfr.oob_score_)
```

```
MSE train data: 3236350.821, MSE test data: 23001389.717
R2 train data: 0.975, R2 test data: 0.812
OOB score: 0.8376630757921818
```

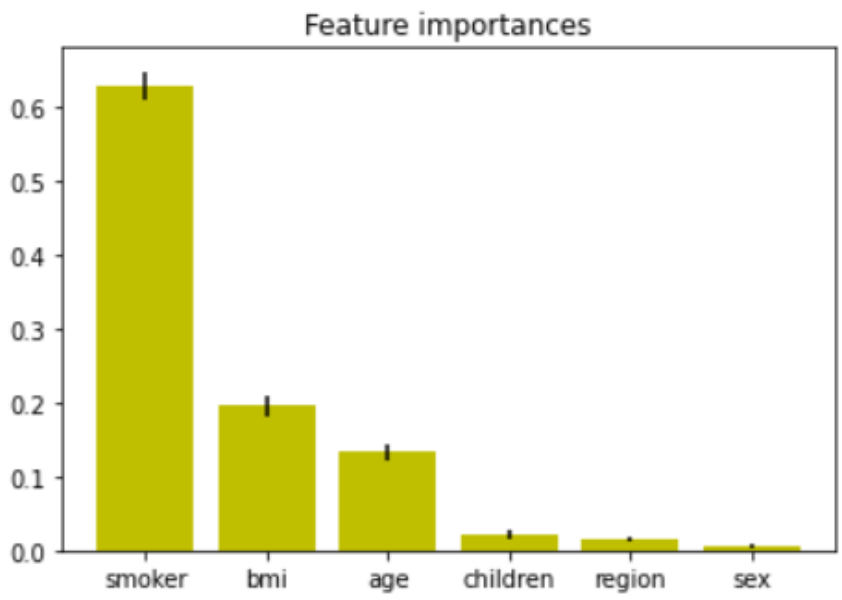
- First, we will import the random forest regressor as rfr
- Next, we will be dropping the ‘charges’ column from the dataset and store the remaining data in ‘x’ variable and in the ‘y’ variable we will store only the ‘charges’ column from the data.
- Here in random forest regressor function the arguments used are -
- **n_estimator** is the number of trees in the forest., by default we will take ‘100’.
- **Criterion:** MSE (mean squared error) to measure the quality of split
- **Random state:** it controls both the randomness of the bootstrapping of the samples used when building trees.
- **N jobs:** The number of jobs to run in parallel and -1 means using all processors.
- **OOB score:** True, (Out Of Bag score) which is used as a replacement for cross validation in Random Forest Regressor).
- Then, we will fit the x train and y train using Rfr.
- Next, we will be performing random forest prediction on both x_train and x_test and store them in variables ‘x_train_pred’ & ‘x_test_pred’.
- Finally, we will print the Mean squared error, R-square values for both the test and train data& OOB score.
- If we see the scores obtained after performing Random Forest Regression on the data, the R-square & OOB scores obtained are high so it is considered good for the model whereas the MSE (mean squared error) values are also high for both train data and test data, from which we can say that there is a large deviation b/w both the observed values and predicted values.

Feature Importance Ranking:

```
print('Feature importance ranking\n\n')
importances = Rfr.feature_importances_
std = np.std([tree.feature_importances_ for tree in Rfr.estimators_],axis=0)
indices = np.argsort(importances)[::-1]
variables = ['age', 'sex', 'bmi', 'children','smoker', 'region']
importance_list = []
for f in range(x.shape[1]):
    variable = variables[indices[f]]
    importance_list.append(variable)
    print("%d.%s(%f)" % (f + 1, variable, importances[indices[f]]))

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(importance_list, importances[indices],
       color="y", yerr=std[indices], align="center")
```

```
Feature importance ranking


1.smoker(0.628962)
2.bmi(0.195371)
3.age(0.133038)
4.children(0.021136)
5.region(0.014632)
6.sex(0.006862)
<BarContainer object of 6 artists>
```

```
<BarContainer object of 6 artists>
```



- Feature importances are provided by the fitted attribute “feature_importances_” and they are computed as the mean and standard deviation of accumulation of the impurity decrease within each tree.

- Next, we will be sorting out the “importances” except the last column and store it in the variable ‘indices’.
- Here we will be declaring the ‘variables’ and an empty ‘importance_list’.
- Then in the “for” loop we will be storing the variables of the indices of ‘f’ in a higher order.
- The next step we will be appending the empty list with the ‘variables’ of higher order in ranking.
- Finally, we will be plotting the bar plot based on the ranking.
- From the result values & the Bar plot obtained we can conclude that the feature importance ranking is high for “smoker” factor & least for “sex” factor compared to all the other factors in the dataset.

Next, we will be using machine learning models for predicting charges. We will be using different regression models to get accurate predictions.

1. Linear Regression:

```
In [103]: # 1 Linear Regression

In [104]: from sklearn.linear_model import LinearRegression
          Lin_reg = LinearRegression()
          Lin_reg.fit(x_train, y_train)
          print(Lin_reg.intercept_)
          print(Lin_reg.coef_)
          print(Lin_reg.score(x_test, y_test))

          -10737.349581180844
          [  248.93819365    81.64928972   304.75124515   515.98877006
           23353.0665509   -419.94573573]
          0.729611397171023
```

- First, we will import the Linear regression from sklearn.linear_model and assign it to Lin_reg.
- Next, we will be fitting the Lin_reg to x_train and y_train.
- Then we will be printing intercept, coefficients of linear regression model fitted before.

- At last, we will be calculating the score by fitting the model trained to x_test and y_test.

```
In [105]: # Cross validation for linear regression
          scores=cross_val_score(Lin_reg,x,y,cv=10,scoring='r2')
          print('R_square score for Cross Validation:',mean(scores))

          R_square score for Cross Validation: 0.7481927885694074
```

```
In [106]: # Kfold cross validation for Linear regression
          cv=KFold(n_splits=10,random_state=1,shuffle=True)
          scores=cross_val_score(Lin_reg,x,y,cv=cv,scoring='r2')
          print('R_square score for K- Fold cross Validation :',mean(scores))

          R_square score for K- Fold cross Validation : 0.7480921370339916
```

- Next, we will be calculating R_square score forCross validation by taking hyper parameters cv=10 and scoring metric as r2.

- Next, we will be calculating R_square score for KFold Cross validation by taking hyper parameters n_splits=10 and random_state=1 and shuffle as True and allocating it to cv.Then we will calculate the score by taking hyper parameters cv=cv and scoring metric as r2.

2. Ridge Regression:

```
In [107]: # 2:- Ridge regression
```

```
In [108]: from sklearn.linear_model import Ridge
          Ridge = Ridge(alpha=1)
          Ridge.fit(x_train, y_train)
          print(Ridge.intercept_)
          print(Ridge.coef_)
          print('Training score:',round(Ridge.score(x_train,y_train),2))
          print('Testing score:',round(Ridge.score(x_test, y_test),2))

          -10713.000901952126
          [  248.94601291    87.52880069   304.75192595   514.9748104
           23211.51669009  -419.3926356 ]
          Training score: 0.76
          Testing score: 0.73
```

- First, we will import the Ridge regressionfrom sklearn.linear_model and assign it to Ridge and also taking alpha=1.
- Next, we will be fitting the Ridge to x_train and y_train.

- Then we will be printing intercept, coefficients of Ridge regression model fitted before.
- At last, we will be calculating the training and testing score.

Since training score and testing score is almost similar. We will try to improve it by increasing the regularisation value alpha=30

```
In [120]: from sklearn.linear_model import Ridge
          ridge = Ridge(alpha=30)
          ridge.fit(x_train, y_train)
          print(ridge.intercept_)
          print(ridge.coef_)
          print('Training score:',round(ridge.score(x_train,y_train),2))
          print('Testing score:',round(ridge.score(x_test, y_test),2))
```

```
-10108.90993708269
[  249.12227327   215.54670659   304.79132631   489.23766276
 19742.32460545  -404.64688295]
Training score: 0.74
Testing score: 0.72
```

Since there is a dip in both training and testing score compared to previous one so we will consider alpha=10 for finding R_square Cross validation score.

```
In [121]: # Cross validation for ridge regression
          ridge = Ridge(alpha=10)
          scores=cross_val_score(ridge,x,y,cv=10,scoring='r2')
          print('R_square score for Cross Validation:',mean(scores))
```

```
R_square score for Cross Validation: 0.7466306935656075
```

```
In [122]: # K Fold cross validation for ridge regression
          cv=KFold(n_splits=10,random_state=1,shuffle=True)
          scores=cross_val_score(ridge,x,y,cv=cv,scoring='r2')
          print('R_square score for K- Fold cross Validation :',mean(scores))
```

```
R_square score for K- Fold cross Validation : 0.7464522559099886
```

- Next, we will be calculating R_square score for Cross validation by taking hyper parameters cv=10 and scoring metric as r2 and alpha=10
- Next, we will be calculating R_square score for KFold Cross validation by taking hyper parameters n_splits=10 and random_state=1 and shuffle as True and allocating it to cv. Then we will calculate the score by taking hyper parameters cv=cv and scoring metric as r2.

Overall, we can see if we try to increase penalty there is a slight variation in training score and testing score. If we try to increase it further the scores are getting reduced. We will try another model and see if the score improves.

3. Lasso Regression:

Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters).

The acronym "LASSO" stands for Least Absolute Shrinkage and Selection Operator.

Lasso regression performs L1 regularization, which adds a penalty equal to the absolute value of the magnitude of coefficients. This type of regularization can result in sparse models with few coefficients; Some coefficients can become zero and eliminated from the model (Feature Selection).

Here's the implementation of Lasso regression in our project. We had created 6 different Lasso regression models with a combination of different hyper parameters.

In [139]:
```
from sklearn.linear_model import Lasso
Lasso = Lasso(alpha=30, fit_intercept=True, normalize=False, precompute=False, max_iter=1000,
              tol=0.0001, warm_start=False, positive=False, random_state=1, selection='random')
Lasso.fit(x_train, y_train)
print(Lasso.intercept_)
print(Lasso.coef_)
print('Training score:',round(Lasso.score(x_train,y_train),2))
print('Testing score:',round(Lasso.score(x_test, y_test),2))
```

```
-10641.83955649896
[  248.83152889     0.           303.86355949   495.27258738
 23165.4801199   -395.41691289]
Training score: 0.76
Testing score: 0.73
```

- The chosen alpha hyper parameter values are 1, 5, 10, 15, 20, 25 and 30
- The chosen selection hyper parameter values are 'cyclic' and 'random'
- The rest hyper parameters values are the default values

Here's the cross validation and K-Fold cross validation score of the model with alpha hyper parameter value set to 30.

In [140]:
```
# Cross Validation for Lasso regression
from sklearn.linear_model import Lasso
Lasso = Lasso(alpha=30, fit_intercept=True, normalize=False, precompute=False, max_iter=1000,
              tol=0.0001, warm_start=False, positive=False, random_state=1, selection='random')
scores=cross_val_score(Lasso,x,y,cv=10,scoring='r2')
print('R_square score for Cross Validation:',mean(scores))
```

```
R_square score for Cross Validation: 0.7482959977037553
```

In [141]:
```
# K Fold cross Validation for Lasso Regression
cv=KFold(n_splits=10,random_state=1,shuffle=True)
scores=cross_val_score(Lasso,x,y,cv=cv,scoring='r2')
print('R_square score for K- Fold cross Validation :',mean(scores))
```

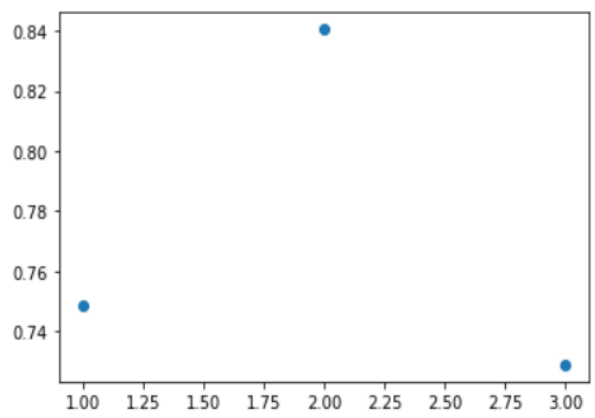
```
R_square score for K- Fold cross Validation : 0.7481506335779491
```

- ✓ By observing all these six models, we can come to know that the training and testing score remains the same without any noticeable change.
- ✓ Lasso supports feature selection to a great extent but does not support feature importance ranking. We still tried to implement feature importance ranking but we faced errors so we had dropped that idea.
- ✓ Now, we proceed to the next model i.e., Polynomial Regression.

4. ***Polynomial Regression***

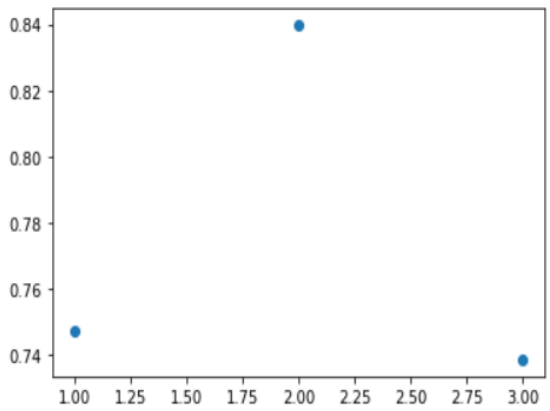
In [76]:
```
# n_splits=5
def create_polynomial_regression_model(degree):
    poly_features = PolynomialFeatures(degree=degree)
    X_poly = poly_features.fit_transform(x)
    poly = LinearRegression()
    cv=KFold(n_splits=5,random_state=1,shuffle=True)
    return np.mean(cross_val_score(poly, X_poly, y, cv=cv,scoring='r2'))
poly_cv = []
for i in range(1,4):
    poly_cv.append(create_polynomial_regression_model(i))
plt.scatter(range(1,4),poly_cv)
```

Out[76]: <matplotlib.collections.PathCollection at 0x17e0774c7f0>



In [77]:
```
# n_splits=10
def create_polynomial_regression_model(degree):
    poly_features = PolynomialFeatures(degree=degree)
    X_poly = poly_features.fit_transform(x)
    poly = LinearRegression()
    cv=KFold(n_splits=10,random_state=1,shuffle=True)
    return np.mean(cross_val_score(poly, X_poly, y, cv=cv,scoring='r2'))
poly_cv = []
for i in range(1,4):
    poly_cv.append(create_polynomial_regression_model(i))
plt.scatter(range(1,4),poly_cv)
```

Out[77]:
```
<matplotlib.collections.PathCollection at 0x17e0779e550>
```



- First, we will be plotting degree of polynomial on x axis and cv score on y axis with having splits=5 and splits=10. From this we can see that polynomial of degree 2 has better cv score than the other degrees.

In [70]:
```
# Polynomial regression
from sklearn.preprocessing import PolynomialFeatures
x = data.drop(['charges', 'sex', 'region'], axis = 1)
y = data.charges
```

In [71]:
```
pol = PolynomialFeatures (degree = 2)
x_pol = pol.fit_transform(x)
x_train, x_test, y_train, y_test = holdout(x_pol, y, test_size=0.2, random_state=0)
Pol_reg = LinearRegression()
Pol_reg.fit(x_train, y_train)
y_train_pred = Pol_reg.predict(x_train)
y_test_pred = Pol_reg.predict(x_test)
print(Pol_reg.intercept_)
print(Pol_reg.coef_)
print(Pol_reg.score(x_test, y_test))
```

```
-5912.886035125941
[ 0.00000000e+00 -1.17396032e+02  6.26370495e+02  1.11113642e+03
 -9.00898057e+03  4.31905688e+00  1.11467949e+00  4.75719512e+00
 -1.39004881e+01 -1.03253930e+01 -9.38476710e+00  1.37636686e+03
 -8.00566253e+01 -2.47002618e+02 -9.00898057e+03]
0.8356067877734482
```

- Next, we will be developing a polynomial regression model as shown in above snap.

Tabulating cross validation Scores for all the models -

```
In [82]: # Tabulating  K Fold cross validation for all the models with n_splits=10
```

```
In [83]: info={'Regression Models':['Linear Regression','Ridge Regression','Lasso Regression','Polynomial Regression'],
              'K Fold Cross Validation score':[0.748,0.746,0.748,0.839]}
```

```
In [84]: print(tabulate(info, headers='keys',tablefmt='fancy_grid',showindex=True))
```

	Regression Models	K Fold Cross Validation score
0	Linear Regression	0.748
1	Ridge Regression	0.746
2	Lasso Regression	0.748
3	Polynomial Regression	0.839

- As we can see that polynomial regression has the highest cross validation score. So, we will be taking this model into consideration and proceeding further for predicting the charges.

```
In [88]: y_test_pred = Pol_reg.predict(x_test)
         #Comparing the actual output values with the predicted values
         df = pd.DataFrame({'Actual': np.round(y_test,decimals=2), 'Predicted': np.round(y_test_pred,decimals=2)})
         df
```

Out[88]:

	Actual	Predicted
667	40003.33	34910.75
412	14455.64	8790.32
1073	12096.65	13431.86
205	4337.74	5458.43
1203	9964.06	11599.95

- Medical insurance charges are predicted using polynomial regression.

Summary and Conclusion

- After performing the feature importance ranking using random forest repressor, we can come to know that smoking contributes the maximum for medical insurance charges.
- Polynomial Regression model has the highest C.V (Cross-Validation) score compared to other models.
- From the predictions we can come to know that there is an ample amount of high insurance medical cost and the reason for that is smoking.

References

https://scikit-learn.org/stable/modules/linear_model.html

https://scikit-learn.org/stable/modules/ensemble.html

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html

https://medium.com/

https://www.analyticsvidhya.com/

https://towardsdatascience.com/

https://stackoverflow.com/