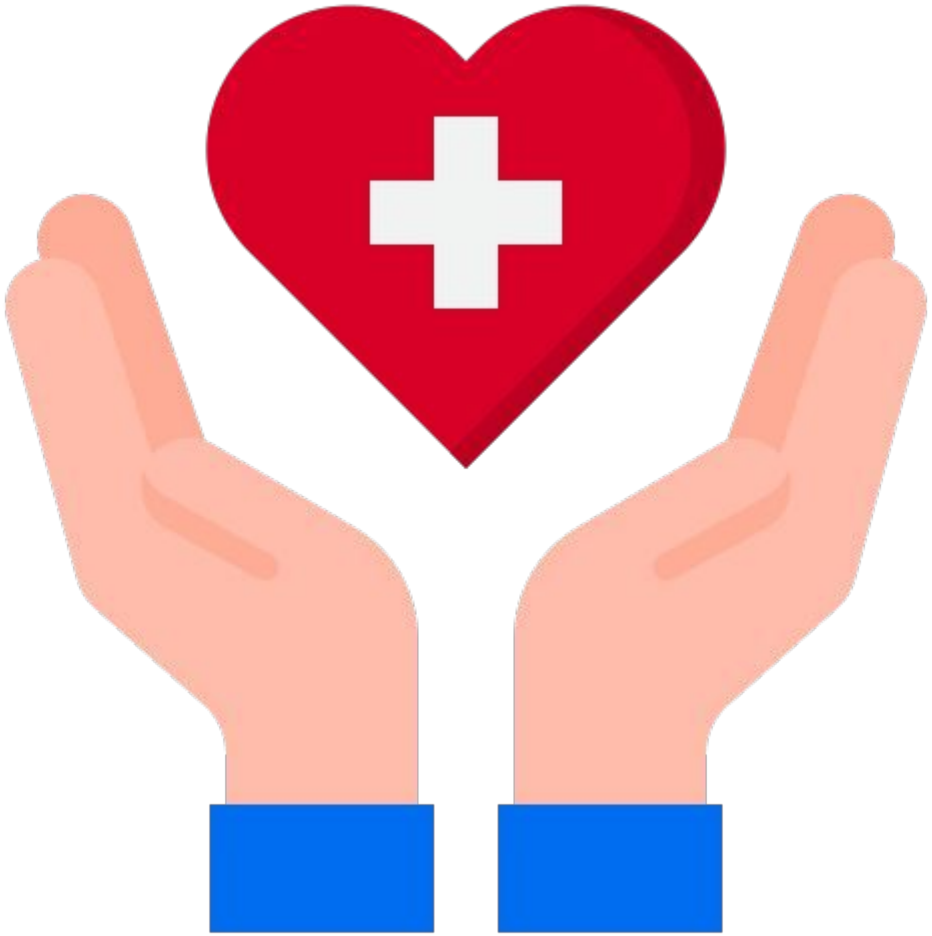
Linear Regression

Insurance Charges Prediction



Agenda

01 Business Problem

02 Solution Approach

03 ED
A

04 Feature Selection And Feature Scaling

05 Train and Test Split

06 Model Fitting

07 Performance of the model

Business Problem - Background

The objective of proposed work is to predict the insurance charges of a person and identify those patients with health insurance policy and medical details weather they have any health issues or not.

The level of treatment in crisis department vary drastically depending the type of health insurance a person has by this we predict the insurance charges of a person .

Solution Approach

Using linear regression model for health insurance prediction is proposed. Some factors like age, gender, bmi, smoker, and children , no.of.past consultation were input for developing the linear regression model .

This kind of model is useful for insurance companies to determine the yearly insurance premium charges for a person

EDA- Loading The Data

```
[ ]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     import warnings
```

Loading the dataset

```
[ ]  insurance = pd.read_csv('new_insurance_data.csv')
```

We have imported all the necessary libraries to perform the EDA, and have imported the dataset using the read_csv() from the pandas module.

EDA- Printing the First 5 Rows

```
insurance.head()
```

	age	sex	bmi	children	smoker	Claim_Amount	past_consultations	num_of_steps	Hospital_expenditure	NUmber_of_pa
0	18.0	male	23.21	0.0	no	29087.54313	17.0	715428.0	4720920.992	
1	18.0	male	30.14	0.0	no	39053.67437	7.0	699157.0	4329831.676	
2	18.0	male	33.33	0.0	no	39023.62759	19.0	702341.0	6884860.774	
3	18.0	male	33.66	0.0	no	28185.39332	11.0	700250.0	4274773.550	
4	18.0	male	34.10	0.0	no	14697.85941	16.0	711584.0	3787293.921	

To print the first five rows of the imported data, we are using the head() method for the pandas dataframe.

EDA- Label Encoding

```python
#label encoding on the sex and smoker column, and changing their data type to integer.
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
insurance['sex'] = le.fit_transform(insurance['sex'])
insurance['smoker'] = le.fit_transform(insurance['smoker'])

#changing the data type of the two columns to integer
insurance['sex'] = insurance['sex'].astype(int)
insurance['smoker'] = insurance['smoker'].astype(int)
```

To make the dataset clear we have transformed the columns ‘sex’ and ‘smoker’ using the label encoder and changed the data type to integer.

EDA- Label Encoding

```
insurance.head()
```

	age	sex	bmi	children	smoker	Claim_Amount	past_consultations	num_of_steps	Hospital_expenditure	NUmber_of_past_hosp
0	18.0	1	23.21	0.0	0	29087.54313	17.0	715428.0	4720920.992	
1	18.0	1	30.14	0.0	0	39053.67437	7.0	699157.0	4329831.676	
2	18.0	1	33.33	0.0	0	39023.62759	19.0	702341.0	6884860.774	
3	18.0	1	33.66	0.0	0	28185.39332	11.0	700250.0	4274773.550	
4	18.0	1	34.10	0.0	0	14697.85941	16.0	711584.0	3787293.921	

The first five rows of the dataset shows the smoker and sex column with the changed values instead of strings to integer data type.

EDA- Descriptive Analysis

```
#descriptive analysis on the data
insurance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 13 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   age                             1329 non-null   float64
 1   sex                             1338 non-null   int64
 2   bmi                             1335 non-null   float64
 3   children                        1333 non-null   float64
 4   smoker                          1338 non-null   int64
 5   Claim_Amount                    1324 non-null   float64
 6   past_consultations              1332 non-null   float64
 7   num_of_steps                    1335 non-null   float64
 8   Hospital_expenditure            1334 non-null   float64
 9   NUmber_of_past_hospitalizations 1336 non-null   float64
 10  Anual_Salary                    1332 non-null   float64
 11  region                          1338 non-null   object
 12  charges                         1338 non-null   float64
dtypes: float64(10), int64(2), object(1)
memory usage: 136.0+ KB
```

The info() function gives the information about the entire dataset. You will get information about the table columns, how many entries (not-null) are present for the columns with their respective data types.

EDA- Descriptive Analysis

```
#checking the column names in the data
insurance.columns
```

```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'Claim_Amount',
       'past_consultations', 'num_of_steps', 'Hospital_expenditure',
       'NUmber_of_past_hospitalizations', 'Anual_Salary', 'region', 'charges'],
      dtype='object')
```

```
#checking the shape of the data
insurance.shape
```

```
(1338, 13)
```

```
#checking the dimensions
insurance.ndim
```

```
2
```

For further information about the data, we can get the column names, the shape of the data and the dimensions of the data available to have more clarity.

EDA- Descriptive Analysis

```
#summary of the insurance data
insurance.describe()
```

	age	sex	bmi	children	smoker	Claim_Amount	past_consultations	num_of_steps	Hospital_expenditure	NUmber_of_
count	1329.000000	1338.000000	1335.000000	1333.000000	1338.000000	1324.000000	1332.000000	1.335000e+03	1.334000e+03	
mean	39.310008	0.505232	30.665112	1.090773	0.204783	33361.327180	15.216216	9.100047e+05	1.584179e+07	
std	14.034818	0.500160	6.101690	1.201856	0.403694	15617.288337	7.467723	9.188612e+04	2.669305e+07	
min	18.000000	0.000000	15.960000	0.000000	0.000000	1920.136268	1.000000	6.954300e+05	2.945253e+04	
25%	27.000000	0.000000	26.302500	0.000000	0.000000	20768.860390	9.000000	8.471995e+05	4.077633e+06	
50%	39.000000	1.000000	30.400000	1.000000	0.000000	33700.310675	15.000000	9.143000e+05	7.490337e+06	
75%	51.000000	1.000000	34.687500	2.000000	0.000000	45052.331957	20.000000	9.716840e+05	1.084082e+07	
max	64.000000	1.000000	53.130000	5.000000	1.000000	77277.988480	40.000000	1.107872e+06	2.616317e+08	

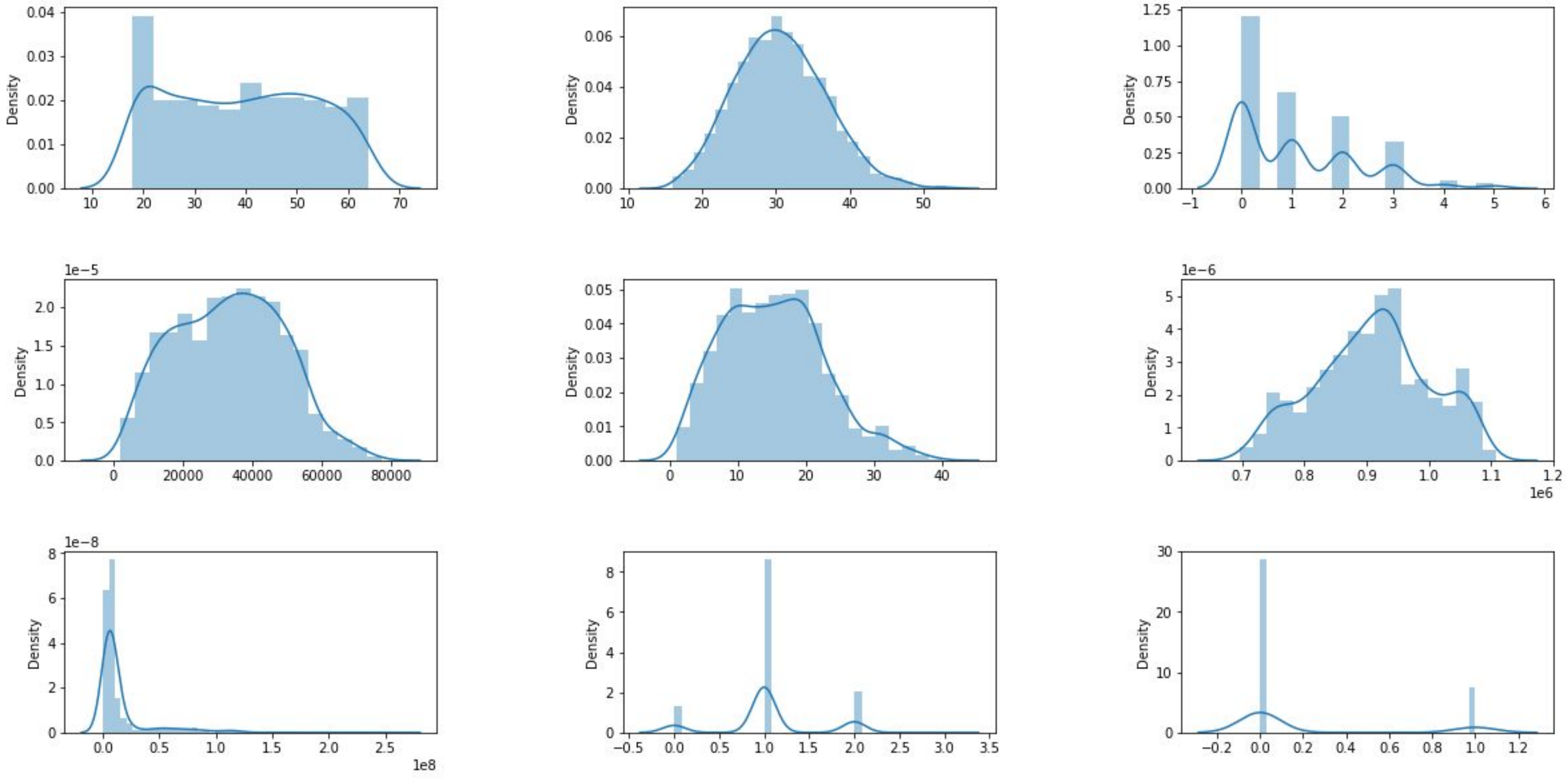
The describe() method gives the five point summary of the data that includes the count, mean, standard deviation, 25th percentile, 50th percentile, 75th percentile, minimum and maximum of each of the columns in the data that are of the type ‘numbers’.

EDA- Descriptive Analysis

```
#checking the distribution of each of the data points in the insurance data.

ig, axes = plt.subplots(3,3, figsize=(20,10))

plt.subplot(3,3,1)
sns.distplot(x=insurance['age'])

plt.subplot(3,3,2)
sns.distplot(x=insurance['bmi'])

plt.subplot(3,3,3)
sns.distplot(x=insurance['children'])

plt.subplot(3,3,4)
sns.distplot(x=insurance['Claim_Amount'])

plt.subplot(3,3,5)
sns.distplot(x=insurance['past_consultations'])

plt.subplot(3,3,6)
sns.distplot(x=insurance['num_of_steps'])

plt.subplot(3,3,7)
sns.distplot(x=insurance['Hospital_expenditure'])

plt.subplot(3,3,8)
sns.distplot(x=insurance['NUmber_of_past_hospitalizations'])

plt.subplot(3,3,9)
sns.distplot(x=insurance['smoker'])

warnings.filterwarnings("ignore")
plt.subplots_adjust(hspace=0.5 , wspace=0.5)
plt.show()
```

We can plot the distribution plots for the columns to get more clarity on the distribution of the data.

EDA- Descriptive Analysis



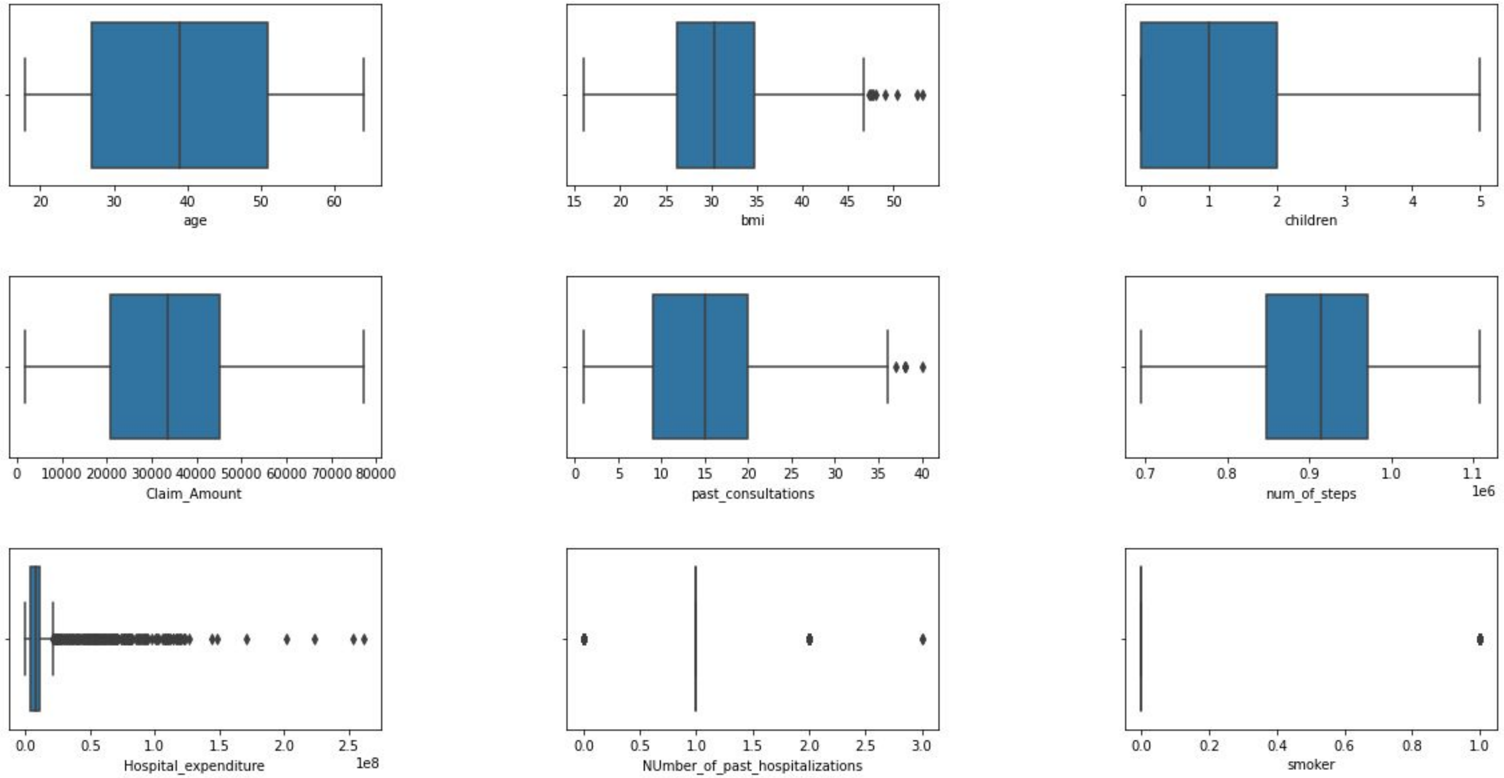
The distribution plots show the data to be skewed for some of the features and close to a normal distribution for some.

EDA- Descriptive Analysis

```
#checking the peakedness and outliers in the data
ig, axes = plt.subplots(3,3, figsize=(20,10))

plt.subplot(3,3,1)
sns.boxplot(x=insurance['age'])

plt.subplot(3,3,2)
sns.boxplot(x=insurance['bmi'])

plt.subplot(3,3,3)
sns.boxplot(x=insurance['children'])

plt.subplot(3,3,4)
sns.boxplot(x=insurance['Claim_Amount'])

plt.subplot(3,3,5)
sns.boxplot(x=insurance['past_consultations'])

plt.subplot(3,3,6)
sns.boxplot(x=insurance['num_of_steps'])

plt.subplot(3,3,7)
sns.boxplot(x=insurance['Hospital_expenditure'])

plt.subplot(3,3,8)
sns.boxplot(x=insurance['NUmber_of_past_hospitalizations'])

plt.subplot(3,3,9)
sns.boxplot(x=insurance['smoker'])

plt.subplots_adjust(hspace=0.5 , wspace=0.5)
plt.show()
```

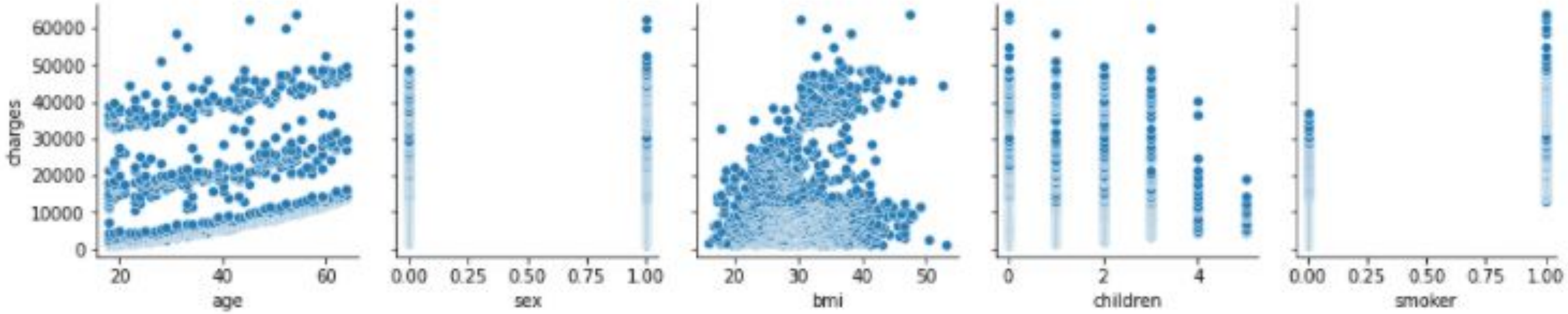
To get a better idea about the peakedness and presence of outliers in the data, we can make use of the boxplots.

EDA- Descriptive Analysis

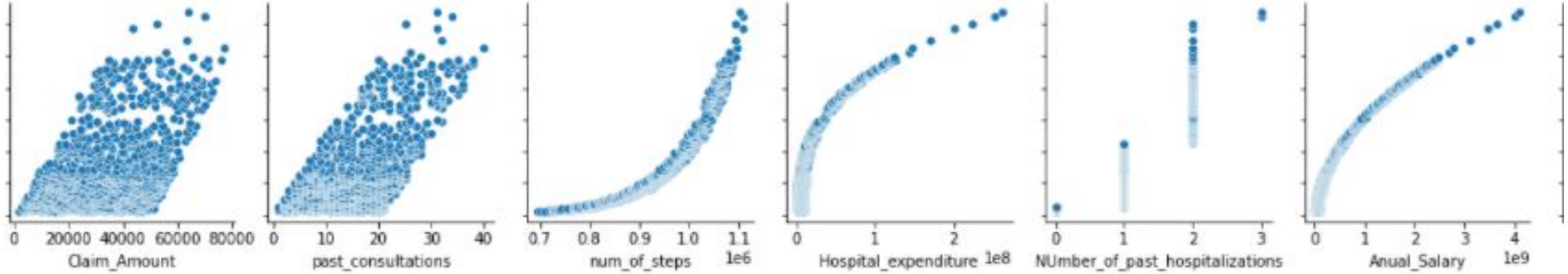


The plots show presence of outliers in some of the features, but we can use the data since the values/outliers does not have to be dealt with at all times. Sometimes, keeping the outliers can still yield good results.

EDA- Descriptive Analysis



The pair plot showcases the relationship between each of the features with the target variable.



EDA- Handling Null Values

```
insurance.isna().sum()
```

```
age                                9
sex                                0
bmi                                3
children                           5
smoker                             0
Claim_Amount                      14
past_consultations                 6
num_of_steps                       3
Hospital_expenditure               4
NUmber_of_past_hospitalizations    2
Anual_Salary                       6
region                             0
charges                            0
dtype: int64
```

Assuming that the data is missing completely at random, we will use the mean/median imputation and replace the missing values with mean and median values of the respective columns. For the distributions that are skewed, we will take the median values to avoid any bias.

Null values in the dataset can cause inefficiency in the model. Therefore has to be dealt with – either we can drop the null values, or we can replace/fill the null values with mean, median, mode of the column.

EDA- Handling Null Values

```
insurance['age'] = insurance['age'].fillna(insurance['age'].mean())
insurance['bmi'] = insurance['bmi'].fillna(insurance['bmi'].mean())
insurance['children'] = insurance['children'].fillna(insurance['children'].median())
insurance['Claim_Amount'] = insurance['Claim_Amount'].fillna(insurance['Claim_Amount'].mean())
insurance['past_consultations'] = insurance['past_consultations'].fillna(insurance['past_consultations'].mean())
insurance['num_of_steps'] = insurance['num_of_steps'].fillna(insurance['num_of_steps'].mean())
insurance['Hospital_expenditure'] = insurance['Hospital_expenditure'].fillna(insurance['Hospital_expenditure'].median())
insurance['NUmber_of_past_hospitalizations'] = insurance['NUmber_of_past_hospitalizations'].fillna(insurance['NUmber_of_past_hospitalizations'].median())
insurance['Anual_Salary'] = insurance['Anual_Salary'].fillna(insurance['Anual_Salary'].median())
```

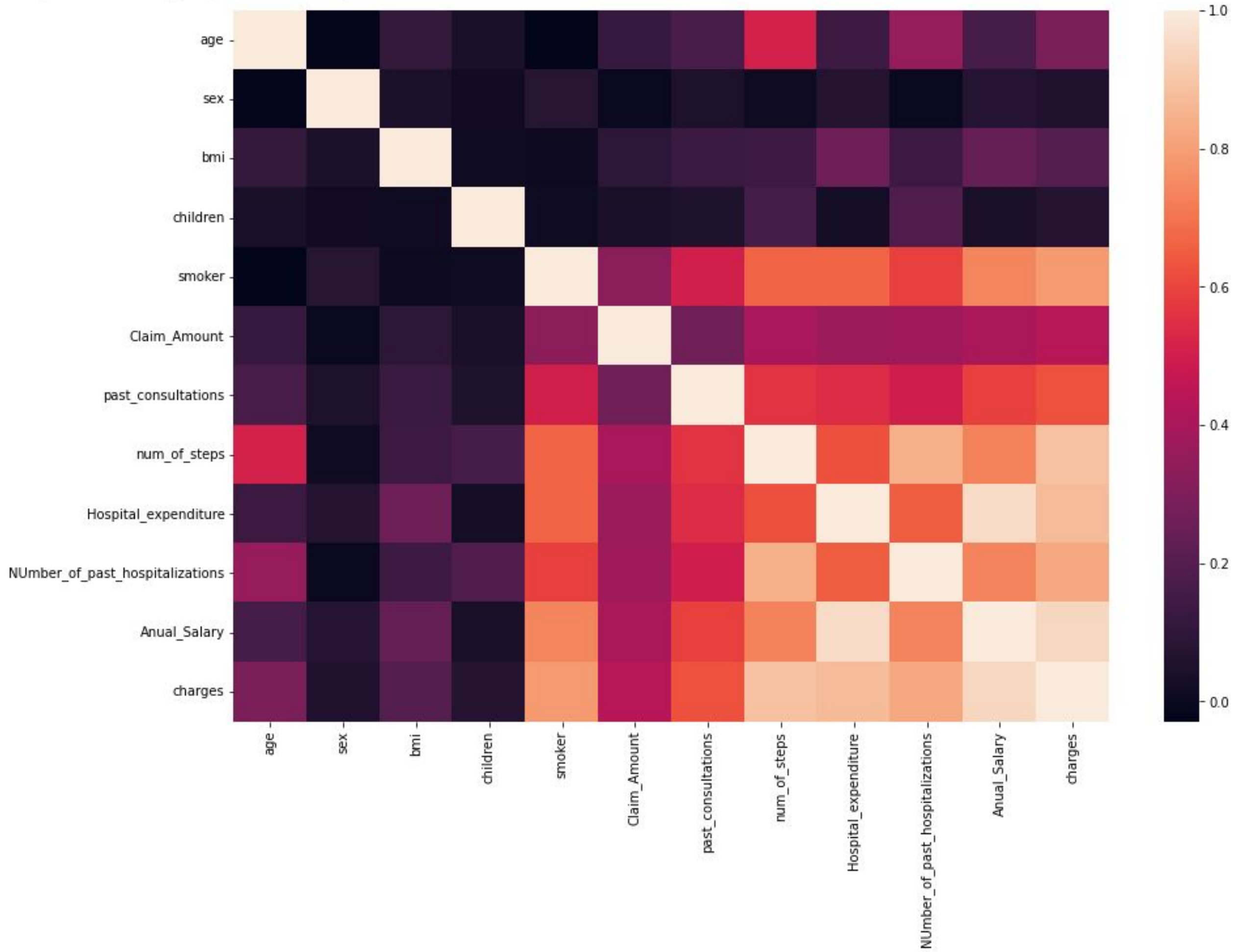
```
insurance.isna().sum()
```

```
age                                0
sex                                0
bmi                                0
children                           0
smoker                             0
Claim_Amount                       0
past_consultations                 0
num_of_steps                       0
Hospital_expenditure               0
NUmber_of_past_hospitalizations    0
Anual_Salary                       0
region                             0
charges                            0
dtype: int64
```

The features where the distribution was skewed, the null values are replaced with the median, and the rest are replaced with the mean values. If there were object type features, we could have replaced them with the mode values.

Feature Selection Using Correlation

<matplotlib.axes._subplots.AxesSubplot at 0x7f5180100290>



The columns that show good correlation with the target variable are going to be used for the prediction.

Splitting the Data into Train And Test

```python
from sklearn.model_selection import train_test_split
#splitting the data
X = insurance.drop(['charges', 'age', 'sex', 'bmi', 'children', 'region'], axis=1)
y = insurance.iloc[:,-1]

#splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train
```

	smoker	Claim_Amount	past_consultations	num_of_steps	Hospital_expenditure	NUmber_of_past_hospitalizations	Anual_Salary
560	0	29622.26103	6.0	886052.0	7.579867e+06	1.0	1.589946e+08
1285	1	66824.70947	23.0	1063413.0	8.042196e+07	2.0	1.919607e+09
1142	0	36320.75384	16.0	1001618.0	1.772151e+07	2.0	7.139574e+08
969	0	24827.43078	8.0	962113.0	1.214312e+07	1.0	2.928227e+08
486	0	47348.03370	10.0	888358.0	6.034962e+06	1.0	5.093163e+07
...	...	...	...	...	...	...	...
1095	0	47554.34106	19.0	1007896.0	1.132722e+07	1.0	5.662888e+08
1130	0	63672.07916	14.0	1000863.0	2.295519e+07	2.0	6.472972e+08
1294	1	42578.49702	24.0	1061168.0	9.188836e+07	2.0	2.038383e+09
860	0	27369.02461	22.0	943007.0	3.634140e+06	1.0	1.877743e+08
1126	0	37385.45533	26.0	1029035.0	1.579127e+07	2.0	6.863093e+08

1070 rows × 7 columns

Splitting the dataset into training and testing data in the ratio 80:20.

Feature Scaling

```
#feature scaling using the standardscalar
from sklearn.preprocessing import StandardScaler
#scaling the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

X_train
```

```
array([[-0.50874702, -0.24550745, -1.2335211 , ..., -0.29745518,
        -0.10122056, -0.36540273],
       [ 1.96561348,  2.14294016,  1.08058381, ...,  2.44746197,
         1.76612428,  2.78222091],
       [-0.50874702,  0.18454486,  0.12771708, ...,  0.08471359,
         1.76612428,  0.62675985],
       ...,
       [ 1.96561348,  0.58630046,  1.21670762, ...,  2.87955205,
         1.76612428,  2.99456786],
       [-0.50874702, -0.39016829,  0.94445999, ..., -0.44614248,
        -0.10122056, -0.31395035],
       [-0.50874702,  0.25290014,  1.48895526, ...,  0.01197612,
         1.76612428,  0.57733069]])
```

Feature scaling through standardization is a necessary practice to normalize the features so that they will have properties of a standard normal distribution i.e. mean is zero and standard deviation is one.

Linear Regression Model

```
[ ]  from sklearn.linear_model import LinearRegression

     model = LinearRegression()
     #fitting the model
     model.fit(X_train, y_train)

     LinearRegression()
```

```
[ ]  #making predictions
     predictions= model.predict(X_test)
```

We can create a linear regression model and fit the training data using the fit() method, and make predictions on the test or new data using the predict() method.

Model Evaluation – r2 and adjusted r2 score

```python
from sklearn.metrics import r2_score
#r2 score
r2 = r2_score(y_test, predictions)
print("r2 score is: {} ".format(r2))

#adjusted r2 score
adj_r2 = 1 - (1 - r2 )*len(y_train)/(len(y_train)-X_train.shape[1]-1)
print("adjusted r2 score is : {}".format(adj_r2))
```

```
r2 score is: 0.9710072494452009
adjusted r2 score is : 0.9707888483110781
```

R squared tells us the goodness of fit. The larger the r2 score, the better the regression model fits the observations.

The adjusted R squared statistic takes into account the number of predictor variables and helps us in determining the goodness of fit in presence of new predictor variables.

Model Evaluation – Mean Squared Error

```
from sklearn.metrics import *
#mean squared error
rmse_on_test = mean_squared_error(y_test, predictions, squared=False)
print("mean squared error on test data is : {}".format(rmse_on_test))
```

```
mean squared error on test data is : 2152.8961180440347
```

```
train_predict = model.predict(X_train)
```

```
rmse_on_train = mean_squared_error(y_train, train_predict, squared=False)
print("mean squared error on train data is : {}".format(rmse_on_train))
```

```
mean squared error on train data is : 1526.9983846830644
```

Calculates the average of squares of the errors of the estimators.

Model Evaluation – Mean Absolute Percentage Error

```
#Mean absolute percentage error
mape_on_train=mean_absolute_percentage_error(y_train, train_predict)
print("mean absolute percentage error on train data is : {}".format(mape_on_train))
```

```
mean absolute percentage error on train data is : 0.1845062145631791
```

```
mape_on_test=mean_absolute_percentage_error(y_test, predictions)
print("mean absolute percentage error on test data is : {}".format(mape_on_test))
```

```
mean absolute percentage error on test data is : 0.1494710727533142
```

Mean absolute percentage error gives you an estimate of the percentage error between the actual and predicted values.

Model Evaluation – Plotting the best fit line

```
error_pred=pd.DataFrame(columns={'Actual_data','Prediction_data'})

error_pred['Actual_data']=y_test
error_pred['Prediction_data']=predictions
error_pred['Error']=error_pred['Actual_data']-error_pred['Prediction_data']
```

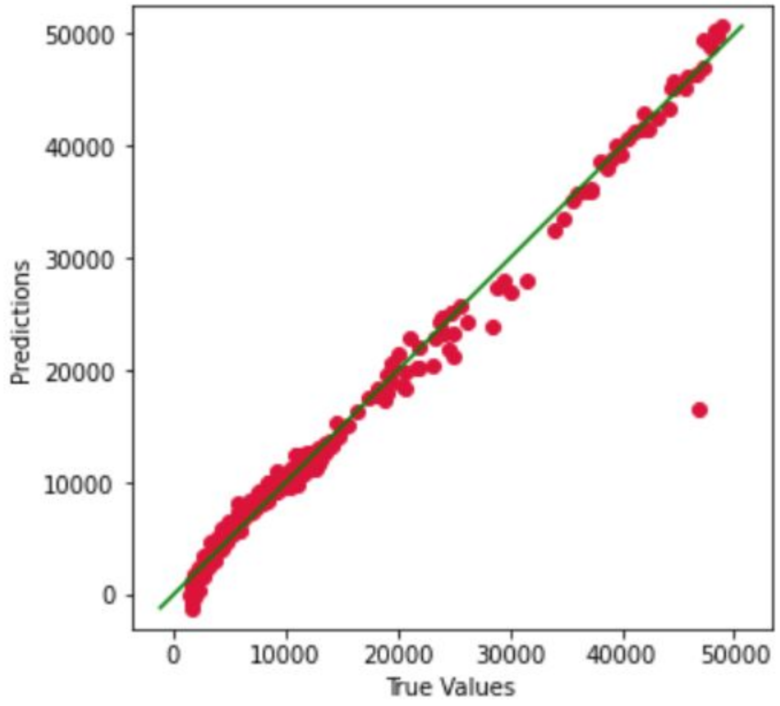
```
error_pred.head()
```

	Prediction_data	Actual_data	Error
764	10732.075670	10928.84900	196.773330
887	11370.964876	12648.70340	1277.738524
890	11967.708103	12797.20962	829.501517
1293	43364.988159	44202.65360	837.665441
259	4472.219237	3925.75820	-546.461037

Using the actual and predicted values to plot the best fit line and understand the error.

Model Evaluation – Plotting the best fit line

```
plt.figure(figsize=(5,5))
plt.scatter(error_pred['Actual_data'], error_pred['Prediction_data'], c='crimson')

p1 = max(max(error_pred['Prediction_data']), max(error_pred['Actual_data']))
p2 = min(min(error_pred['Prediction_data']), min(error_pred['Actual_data']))
plt.plot([p1, p2], [p1, p2], '-g')
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.axis('equal')
plt.show()
```



Plotting the best fit line and the actual and predicted values.