An Insurance company provides Health Insurance to its customers. Various details of the customers are stored amongst which, age, gender, bmi, children, smoker, and region are to be considered. The insurance charges born by the company for each of their customers are also provided. It is required to build a model to predict the insurance charges for a customer given the features, age, gender, bmi, children, smoker, and region.

1. Give a formal description of this application in terms of Task, Experience, and Performance. (Make it a well posed problem)

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. For example: a) Task (T): Classify a tweet that has not been published as going to get retweets or not. b) Experience (E): A corpus of tweets for an account where some have retweets and some do not. c) Performance (P): Classification accuracy, the number of tweets predicted correctly out of all tweets considered as a percentage.

Answer : TASK : Predict the insurance charge for a customer. (Regression)

EXPERIENCE : (Data available) Data of customers with different data values and insurance charges.

PERFORMACE: How close was the predicted insurance charge to the actual insurance charge.

2. Do exploratory data analysis on the data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import LabelEncoder,OneHotEncoder, StandardScaler
import seaborn as sns
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files health_insurance.csv

- **health_insurance.csv**(application/vnd.ms-excel) - 55628 bytes, last modified: 2/8/2021 - 100% done

```
Saving health_insurance.csv to health_insurance (3).csv
```

```
insurance_data = pd.read_csv("health_insurance.csv")
insurance_data.head(5)
```

	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

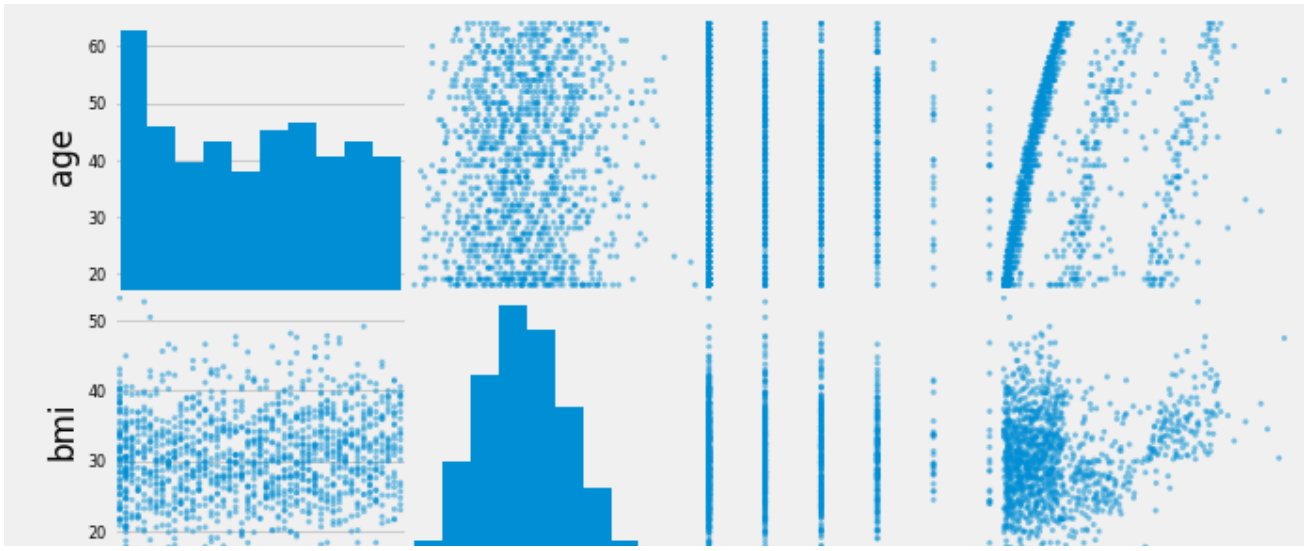
```
insurance_data.shape
```

```
(1338, 7)
```

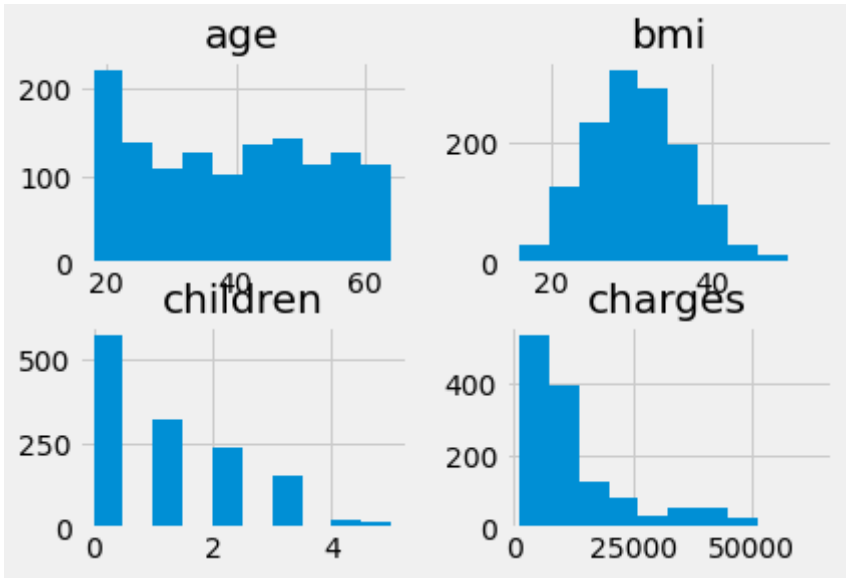
```
insurance_data.describe()
```

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

```
scatter_matrix(insurance_data,figsize=(10,10))
plt.show()
```




```
insurance_data.hist()
plt.show()
```



3. Do the required preprocessing on the dataset to make it suitable to apply gradient descent algorithm.

```
# Checking for missing data
for col in insurance_data.columns:
    pct_missing = np.mean(insurance_data[col].isnull())
    print('{} - {}%'.format(col, round(pct_missing*100)))
```

```
age - 0%
sex - 0%
bmi - 0%
children - 0%
smoker - 0%
region - 0%
charges - 0%
```

```
#changing sex attribute into binary
lb = LabelEncoder()
```

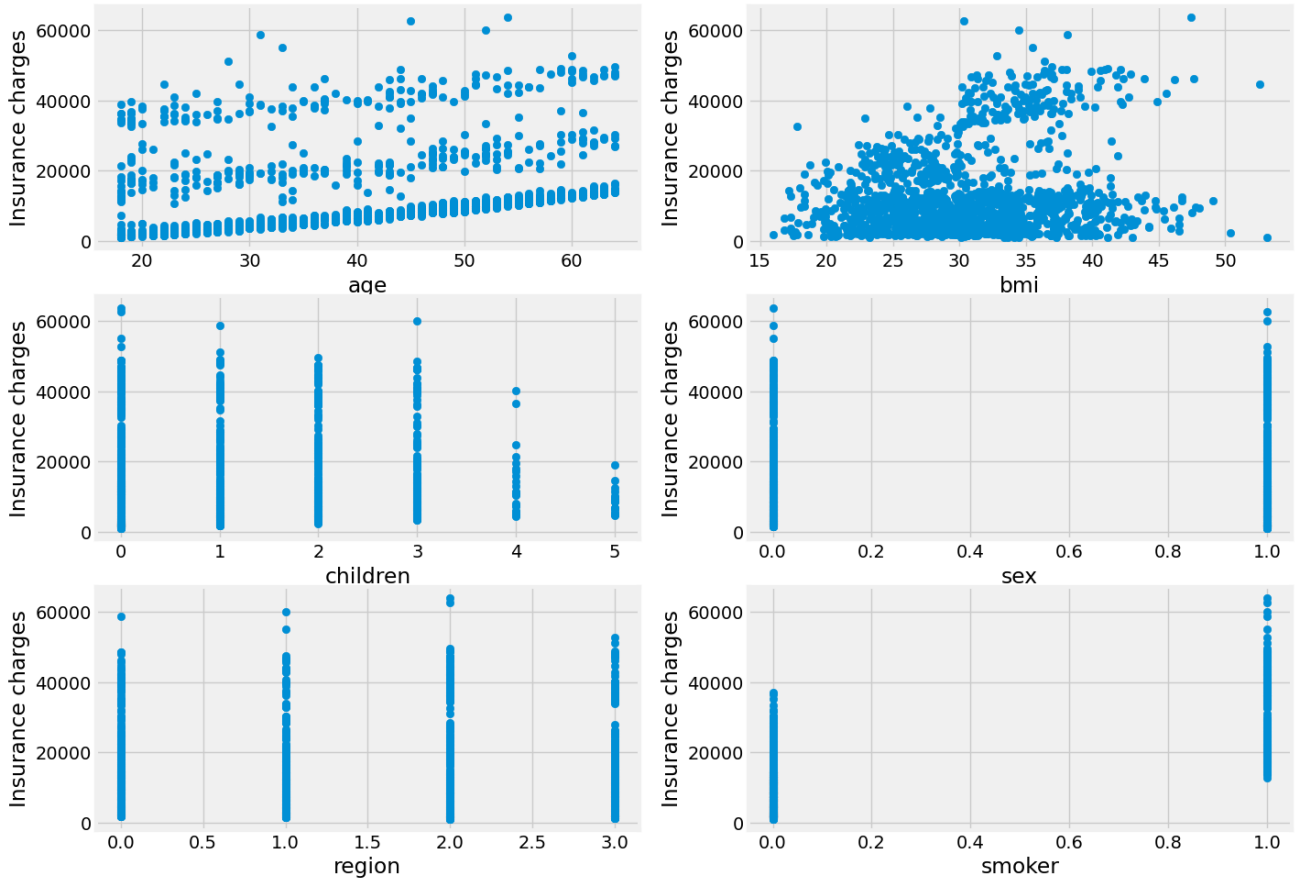
```
insurance_data['sex'] = lb.fit_transform(insurance_data.sex)

#changing region attribute into binary
lb = LabelEncoder()
insurance_data['smoker'] = lb.fit_transform(insurance_data.smoker)

#changing region attribute into binary
lb = LabelEncoder()
insurance_data['region'] = lb.fit_transform(insurance_data.region)


insurance_data.head(5)
```

	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

4. Find the relationship of each feature with the target feature insurance charge by using a scatter plot.

For example, you can plot a graph with samples (Ai,Bi) with A on the X-axis and B on the Y-axis. A is one of the features, say, age, and B is the target feature, insurance charges. Repeat it for other features too such as sex, bmi, children, smoker, region.

```
plt.figure(figsize=(15,15), dpi =100,facecolor='white')
n =0
for x in ['age', 'bmi','children','sex','region','smoker']:
  n +=1
  plt.subplot(4,2,n)
  plt.scatter(insurance_data[x],insurance_data["charges"])
  #plt.title(x)
  plt.xlabel(x)
  plt.ylabel('Insurance charges')
plt.show()
```




```
# plotting the correlation plot for the dataset
f, ax = plt.subplots(figsize = (7, 7))

corr = insurance_data.corr()
sns.heatmap(corr, mask = np.zeros_like(corr, dtype = np.bool),
            cmap = sns.diverging_palette(50, 10, as_cmap = True), square = True, ax = ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fa499b550>
```

5. Implement linear regression using inbuilt package of python scikit.

This will help to estimate the function f(X)=Y where X is one of the input features and Y is the target feature, insurance charges.

```
from sklearn.linear_model import LinearRegression
```

```
# splitting the dependent and independent variable
x = insurance_data["age"]
y = insurance_data["charges"]

print(x.shape)
print(y.shape)
```

```
(1338,)
(1338,)
```

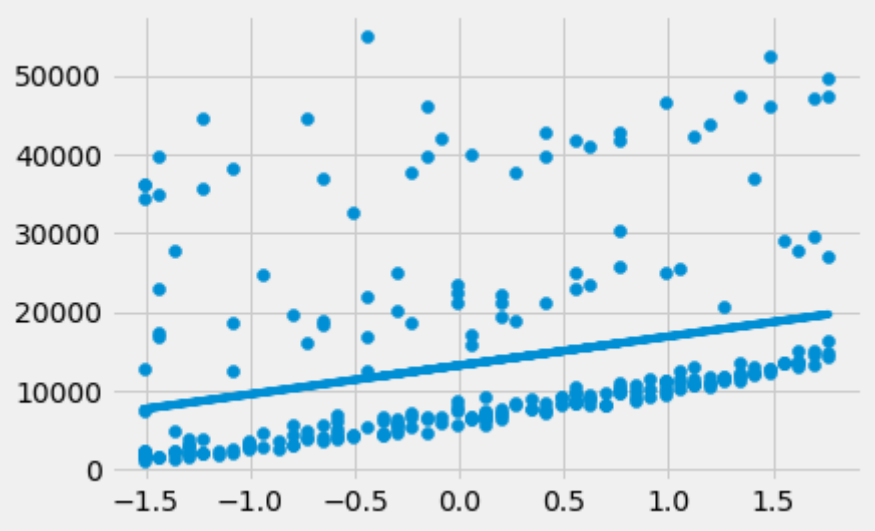
```
 # standard scaling
from sklearn.preprocessing import StandardScaler
# creating a standard scaler
sc = StandardScaler()
# feeding independents sets into the standard scaler
x = np.array(x)
x = x.reshape(-1, 1)
x = sc.fit_transform(x)
```

```
#Splitting data into training and test
from sklearn.model_selection import cross_val_score, train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state =
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1070, 1)
(268, 1)
(1070,)
(268,)
```

```
# creating the model
model = LinearRegression()
x_train = np.array(x_train).reshape(-1, 1)
# feeding the training data to the model
model.fit(x_train, y_train)

# predicting the test set results
y_pred = model.predict(x_test)


plt.plot(x_test,y_pred)
plt.scatter(x_test,y_test)
plt.show()
```



6. Implement gradient descent algorithm with the function prototype

```
def cost_function(X, y, theta):
    m = y.size
    error = np.dot(X, theta.T) - y
    cost = 1/(2*m) * np.dot(error.T, error)
    return cost, error

def gradient_descent(X, y, theta, alpha, iters=150):
    cost_array = np.zeros(iters)
    m = y.size
    for i in range(iters):
        cost, error = cost_function(X, y, theta)
        theta = theta - (alpha * (1/m) * np.dot(X.T, error))
```

```
        cost_array[i] = cost
    return theta, cost_array

def run():

    # Extract data into X and y
    X =insurance_data[['age', 'smoker']]
    y =insurance_data['charges']

    # Normalize our features
    X = (X - X.mean()) / X.std()

    # Add a 1 column to the start to allow vectorized gradient descent
    X = np.c_[np.ones(X.shape[0]), X]

    # Set hyperparameters
    alpha = 0.01
    iterations = 1500

    # Initialize Theta Values to 0
    theta = np.zeros(X.shape[1])
    initial_cost, _ = cost_function(X, y, theta)

    print('With initial theta values of {0}, cost error is {1}'.format(theta, initial_cost

    # Run Gradient Descent
    theta, cost_num = gradient_descent(X, y, theta, alpha, iterations)

    final_cost, _ = cost_function(X, y, theta)

    print('With final theta values of {0}, cost error is {1}'.format(theta, final_cost))




if __name__ == "__main__":
    run()
```

```
With initial theta values of [0. 0. 0.], cost error is 161323436.79435235
With final theta values of [13270.41850091  3861.92701096  9630.24092534], cost error
```

7. Vary learning rate from 0.1 to 0.9 and observe the learned parameter

```
theta, cost_num = gradient_descent(X, y, alpha, iterations)


X = insurance_data[['age', 'smoker']]
y = insurance_data['charges']

# Normalize our features
X = (X - X.mean()) / X.std()

    # Add a 1 column to the start to allow vectorized gradient descent
```

```
X = np.c_[np.ones(X.shape[0]), X]
theta = np.zeros(X.shape[1])

a = np.linspace(0,1,10,endpoint=False)
import math
for var in a:
  theta, cost_num = gradient_descent(X, y, theta,var,1500)
  print('theta values of {0}, cost error is {1}'.format(theta, cost_num))
```

```
theta values of [0. 0. 0.], cost error is [1.61323437e+08 1.61323437e+08 1.61323437e+
 1.61323437e+08 1.61323437e+08]
theta values of [13270.42226514  3861.92927093  9630.24431832], cost error is [1.6132
 2.04133468e+07 2.04133468e+07]
theta values of [13270.42226514  3861.92927093  9630.24431832], cost error is [204133
 20413346.84615945 20413346.84615945 20413346.84615945]
theta values of [13270.42226514  3861.92927093  9630.24431832], cost error is [204133
 20413346.84615945 20413346.84615945 20413346.84615945]
theta values of [13270.42226514  3861.92927093  9630.24431832], cost error is [204133
 20413346.84615945 20413346.84615945 20413346.84615945]
theta values of [13270.42226514  3861.92927093  9630.24431832], cost error is [204133
 20413346.84615945 20413346.84615945 20413346.84615945]
theta values of [13270.42226514  3861.92927093  9630.24431832], cost error is [204133
 20413346.84615945 20413346.84615945 20413346.84615945]
theta values of [13270.42226514  3861.92927093  9630.24431832], cost error is [204133
 20413346.84615945 20413346.84615945 20413346.84615945]
theta values of [13270.42226514  3861.92927093  9630.24431832], cost error is [204133
 20413346.84615945 20413346.84615945 20413346.84615945]
theta values of [13270.42226514  3861.92927093  9630.24431832], cost error is [204133
 20413346.84615945 20413346.84615945 20413346.84615945]
```