AIML Project Part A

2101005C PE04

Nagul Singh Rup Manipur

Importing the libraries

In [1]:
```
#importing the libraries
import numpy as np
import pandas as pd
from numpy import math

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

import matplotlib.pyplot as plt
```

Importing the dataset

In [2]:
```
#importing the dataset
dataset = pd.read_csv('./dataset/insurance.csv', skipinitialspace=True)
```

Displaying the data

In [3]:
```
dataset.head()
```

Out[3]:

	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

In [4]:
```
len(dataset)
```

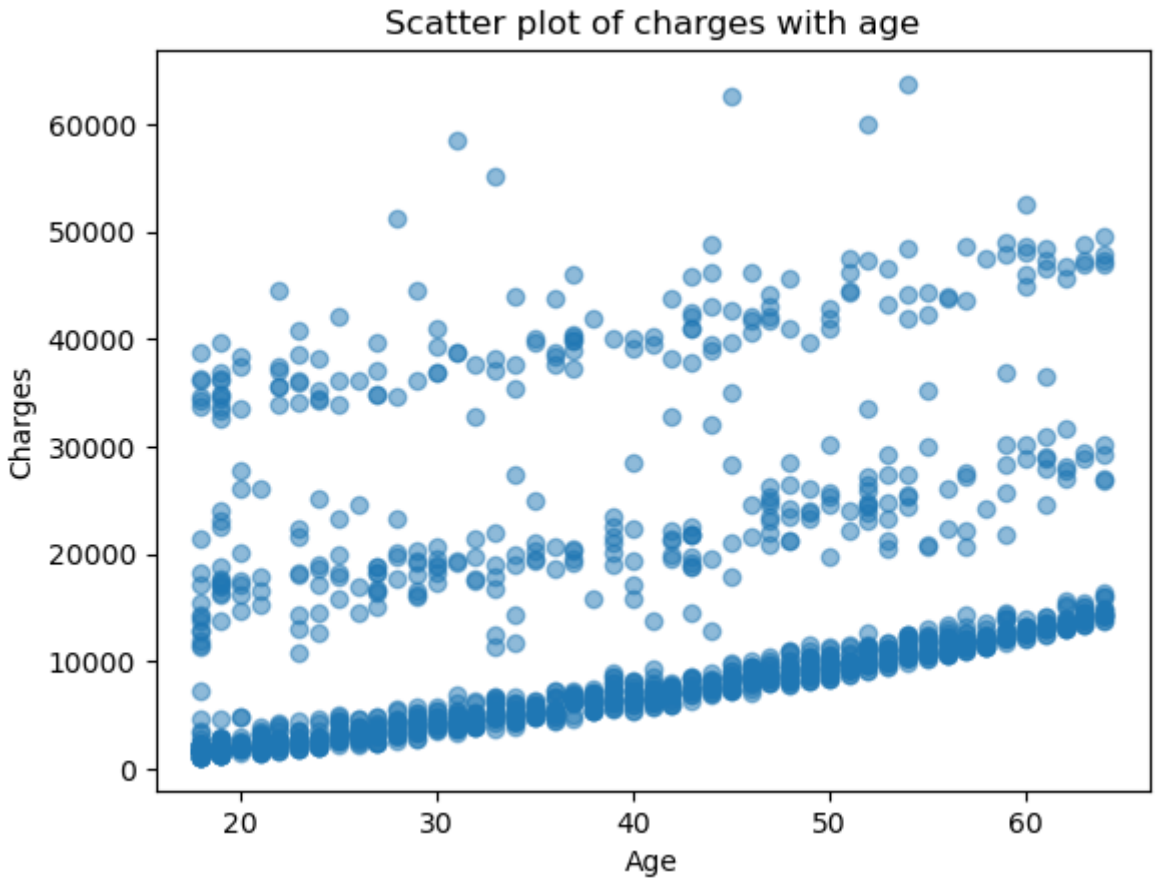
Out[4]:
```
1338
```

In [5]:
```
dataset.shape
```

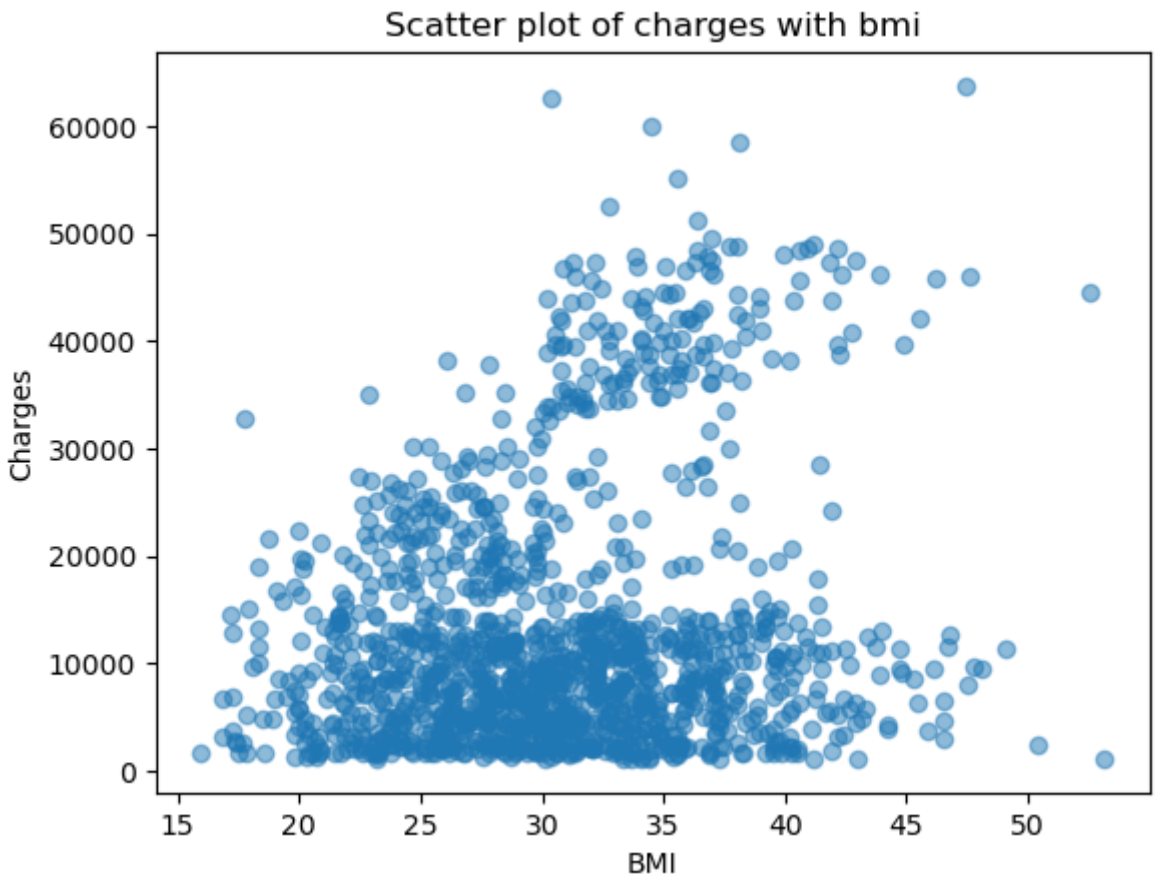
Out[5]:
```
(1338, 7)
```

Understanding the data

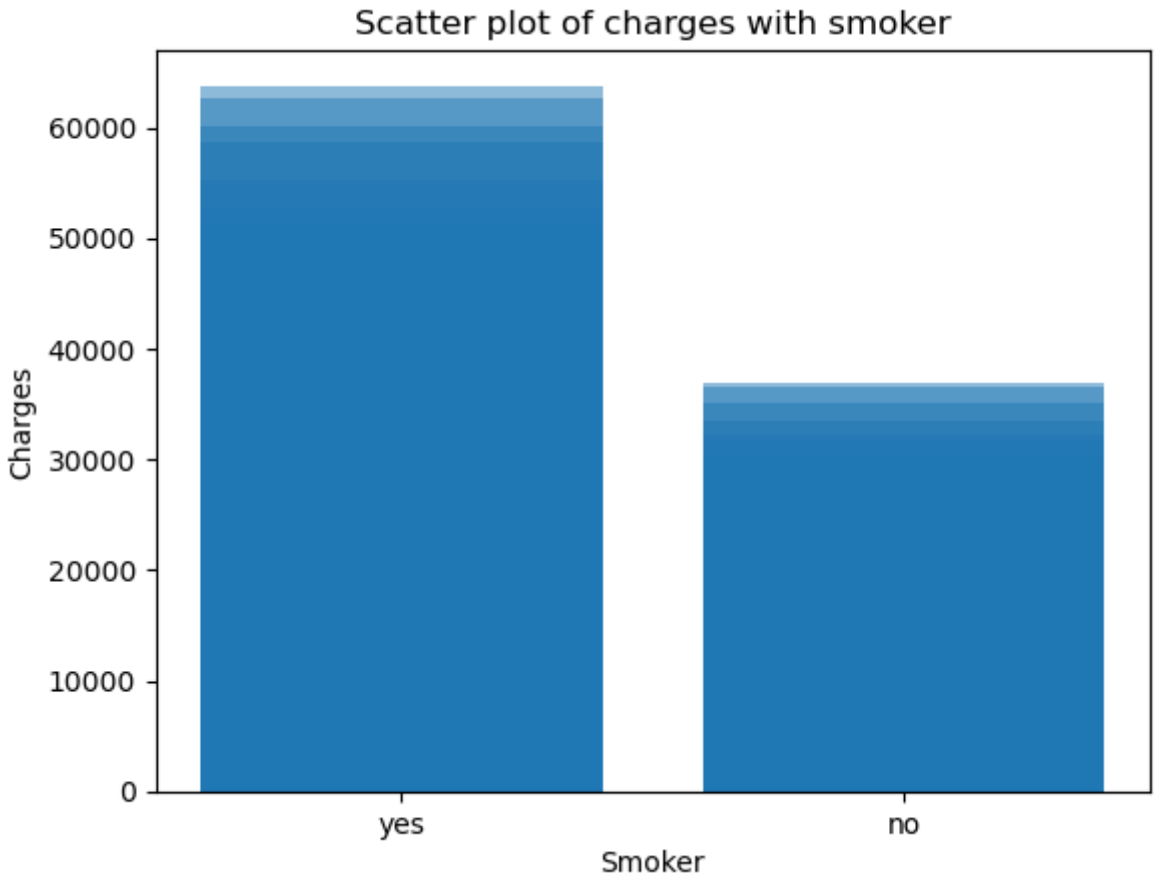
In [6]:
```
plt.scatter(dataset['age'], dataset['charges'], alpha=0.5)
plt.title('Scatter plot of charges with age')
plt.xlabel('Age')
plt.ylabel('Charges')
plt.show()
```



In [7]:
```
plt.scatter(dataset['bmi'], dataset['charges'], alpha=0.5)
plt.title('Scatter plot of charges with bmi')
plt.xlabel('BMI')
plt.ylabel('Charges')
plt.show()
```



In [8]:
```
plt.bar(dataset['smoker'], dataset['charges'], alpha=0.5)
plt.title('Scatter plot of charges with smoker')
plt.xlabel('Smoker')
plt.ylabel('Charges')
plt.show()
```



In [9]:
```
dataset.region.value_counts()
```

Out[9]:
```
southeast    364
southwest    325
northwest    325
northeast    324
Name: region, dtype: int64
```

In [10]:
```
dataset.smoker.value_counts()
```

Out[10]:
```
no     1064
yes     274
Name: smoker, dtype: int64
```

In [11]:
```
dataset.sex.value_counts()
```

Out[11]:
```
male      676
female    662
Name: sex, dtype: int64
```

Performing Label Encoding

In [12]:
```
#get_dummies to perform one hot encoding
dataset = pd.get_dummies(dataset, columns=['region', 'smoker', 'sex'])
```

In [13]:
```
dataset.head()
```

Out[13]:

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

Perparing data for Regression

In [14]:
```
dependent_variable = 'charges'
```

In [15]:
```
#create a list of independent variables
independent_variables = dataset.columns.tolist()
```

In [16]:
```
independent_variables.remove(dependent_variable)
```

In [17]:
```
independent_variables
```

Out[17]:
```
['age',
 'bmi',
 'children',
 'region_northeast',
 'region_northwest',
 'region_southeast',
 'region_southwest',
 'smoker_no',
 'smoker_yes',
 'sex_female',
 'sex_male']
```

In [18]:
```
#create the data of independent variables
X = dataset[independent_variables].values

#create the data of dependent variable
y = dataset[dependent_variable].values
```

Spliting the data for training and testing

In [19]:
```
#splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, r
```

In [20]:
```
#Transforming the data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [21]:
```
#display training set
X_train[0:10]
```

Out[21]:
```
array([[0.41304348, 0.48802798, 0.8       , 0.        , 0.        ,
        0.        , 1.        , 0.        , 1.        , 0.        ,
        1.        ],
       [0.        , 0.49690611, 0.        , 0.        , 0.        ,
        1.        , 0.        , 1.        , 0.        , 0.        ,
        1.        ],
       [0.10869565, 0.55716976, 0.4       , 1.        , 0.        ,
        0.        , 0.        , 0.        , 1.        , 1.        ,
        0.        ],
       [0.30434783, 0.51762174, 0.4       , 0.        , 0.        ,
        0.        , 1.        , 1.        , 0.        , 0.        ,
        1.        ],
       [0.86956522, 0.44215765, 0.2       , 1.        , 0.        ,
        0.        , 0.        , 1.        , 0.        , 1.        ,
        0.        ],
       [0.15217391, 0.291364  , 0.4       , 0.        , 1.        ,
        0.        , 0.        , 1.        , 0.        , 1.        ,
        0.        ],
       [0.39130435, 0.51762174, 0.2       , 0.        , 0.        ,
        1.        , 0.        , 0.        , 1.        , 0.        ,
        1.        ],
       [0.34782609, 0.46516008, 0.2       , 1.        , 0.        ,
        0.        , 0.        , 1.        , 0.        , 1.        ,
        0.        ],
       [0.76086957, 0.41404358, 0.        , 0.        , 0.        ,
        1.        , 0.        , 1.        , 0.        , 0.        ,
        1.        ],
       [0.58695652, 0.22464353, 1.        , 0.        , 0.        ,
        1.        , 0.        , 1.        , 0.        , 0.        ,
        1.        ]])
```

Fitting Multiple Linear Regression to the Training set

In [22]:
```
# Fitting Multiple Linear Regression to the Training set
regressor = LinearRegression(normalize=True)
regressor.fit(X_train, y_train)
```

```
/Users/nagul/opt/anaconda3/envs/tensorflow/lib/python3.9/site-packages/sklearn/linear_model/_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
```

Out[22]:
```
LinearRegression(normalize=True)
```

Prediciting the Test set results

In [23]:
```
#prediciting the Test set results
y_pred = regressor.predict(X_test)
```

In [24]:
```
#this is the amount of the error by the model
mse = math.sqrt(mean_squared_error(y_test, y_pred))
print(mse)
```

```
5641.963425821116
```

In [25]:
```
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
0.7999638104993303
```

In [26]:
```
print(f"Mean squared error: {mse:.2f}")
print(f"R^2 score: {r2:.2f}")
# The coefficient of determination: 1 is perfect prediction
```

```
Mean squared error: 5641.96
R^2 score: 0.80
```

In [27]:
```
display = pd.DataFrame()
display['Original'] = y_test
display['Predicted'] = y_pred
display
```

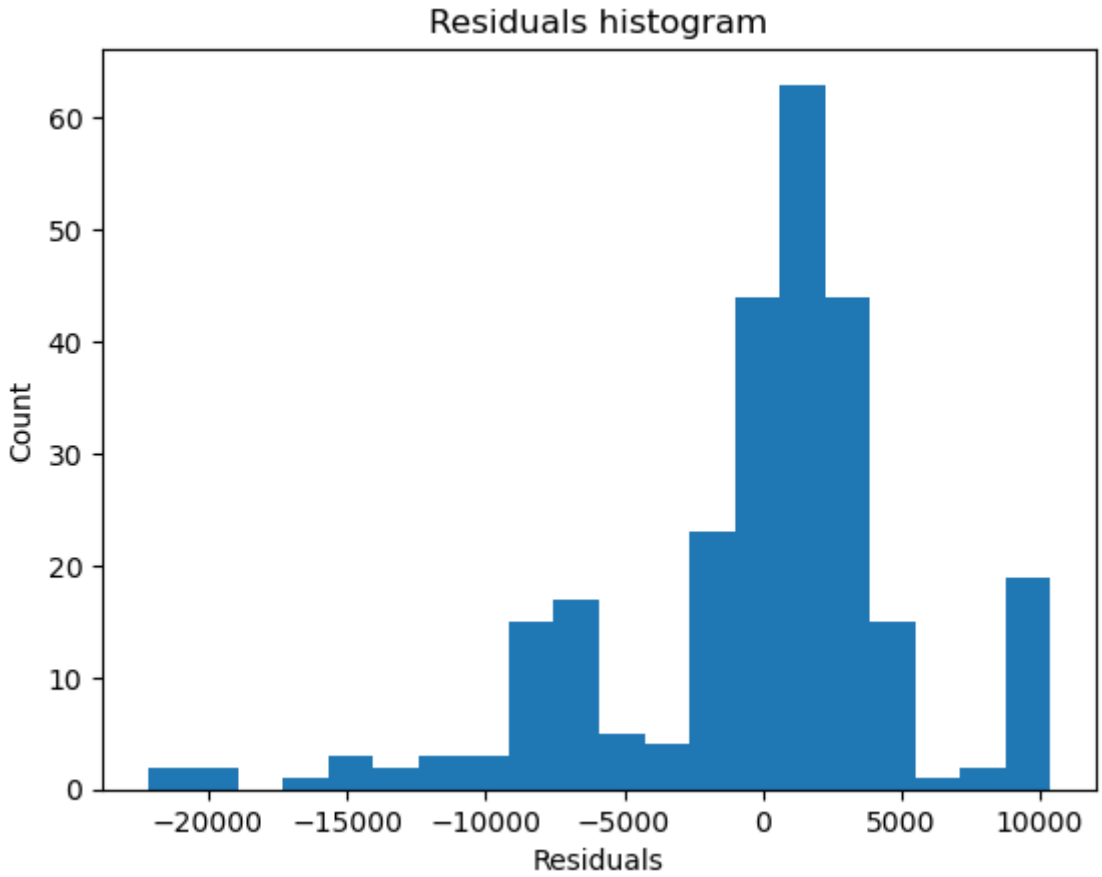
Out[27]:

	Original	Predicted
0	9724.53000	11296.0
1	8547.69130	9440.0
2	45702.02235	38240.0
3	12950.07120	16352.0
4	9644.25250	6912.0
...	...	...
263	15019.76005	14784.0
264	6664.68595	8320.0
265	20709.02034	16192.0
266	40932.42950	32864.0
267	9500.57305	9472.0

268 rows × 2 columns

In [28]:
```
# Calculate the residuals
residuals = y_pred - y_test

# Plot the residuals
plt.hist(residuals, bins=20)
plt.xlabel('Residuals')
plt.ylabel('Count')
plt.title('Residuals histogram')
plt.show()
```



Conclusion

The R-square value of this model is 0.80 and has a mean squared error of 5641.96. This means that the model has a reasonably low amount of error which makes the prediction more precise, therefore it is suitable to use this model for predicting insurance prices.