We can start the project by importing the required libraries and read the data

```
import pandas as pd
import numpy as np
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
import math
```

```
df=pd.read_csv('insurance.csv')
```

```
print(df.describe())
print(df.isna().sum())
df
```

```
              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
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

	age	sex	bmi	children	smoker	region	charges

Looking at the data above we can see there are more than 1338 rows and 7 column with no NA

Let's examine possible categorical outcomes for the region and smoker features.

3	33	male	22.705	0	no	northwest	21984.47061

```
print (df.region.unique())
print (df.smoker.unique())
```

```
['southwest' 'southeast' 'northwest' 'northeast']
['yes' 'no']
```

1334	18	female	31.920	0	no	northeast	2205.98080

In order to better analyze the data, I am going to hot end code the categorical data

1336	21	female	25.800	0	no	southwest	2007.94500

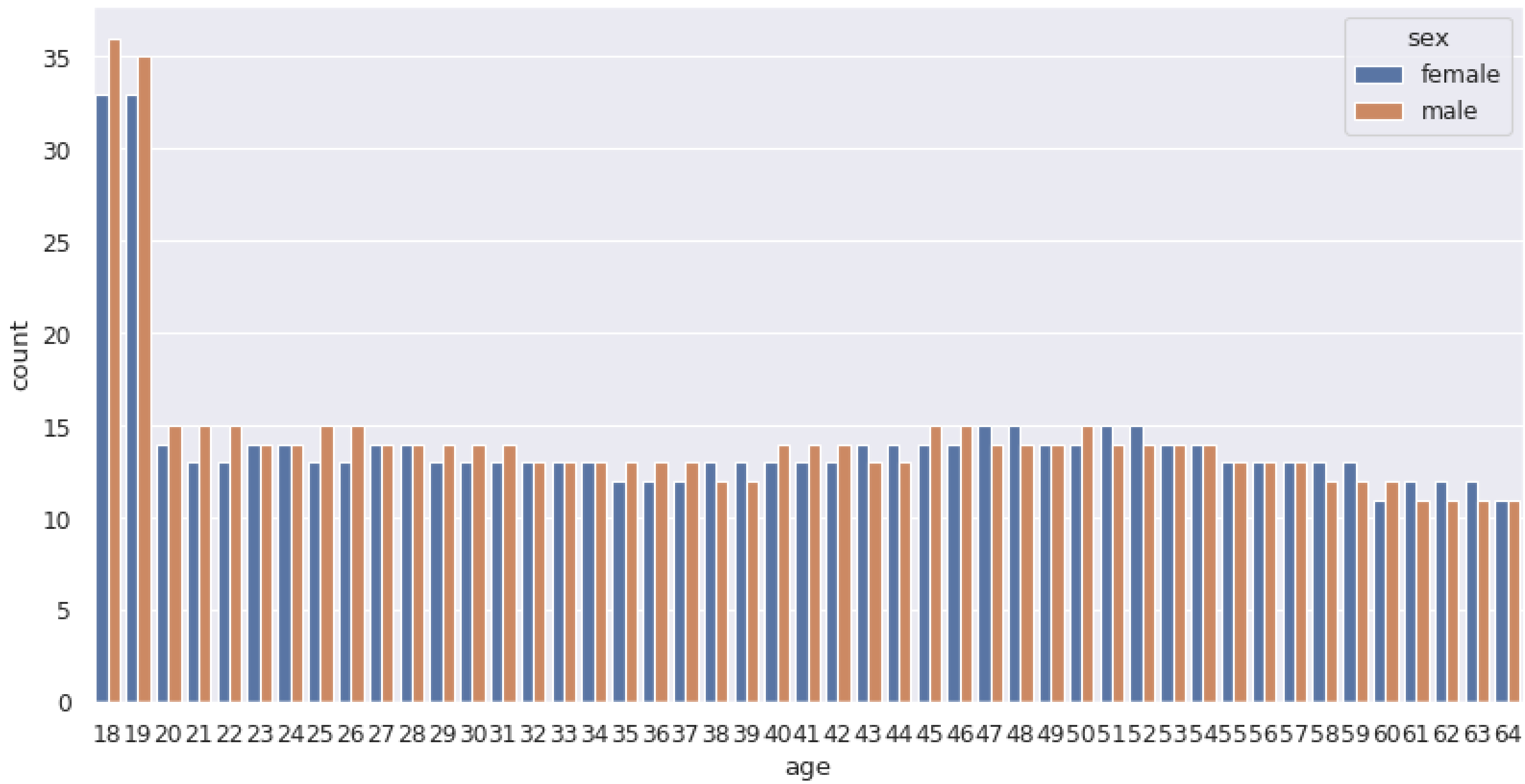
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['smoker'] = le.fit_transform(df['smoker'])
```

Exploratory Analysis

I would like to know if my data is skewed and visualize the data distribution.

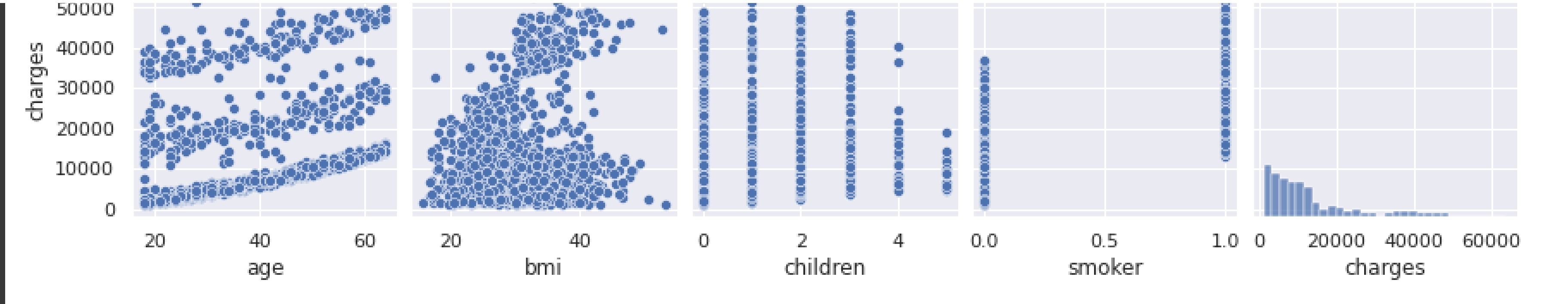
```
from matplotlib.pyplot import figure
figure(figsize=(12, 6), dpi=80)
sns.countplot(x=df['age'],hue=df['sex'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9575a218d0>
```




```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7f95759f5cd0>
```



Few insights from the pair plot: If the patient is a smoker, its likely the charge would be higher, The charges are primarily in the range of 0 to 20,000 (I need to further investigate with normalization). I need to investigate the BMI and charges further as the current graph does not provide enough insight.

Lets check of the charges distribution

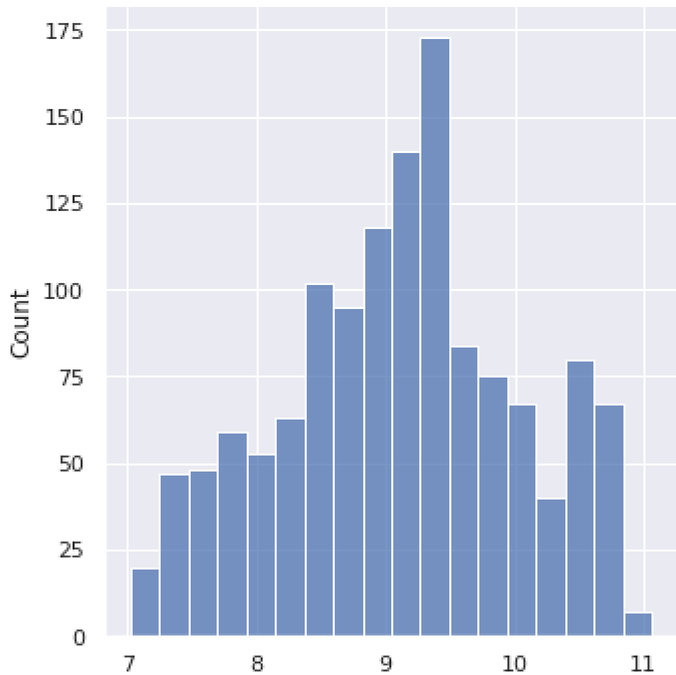
```
charges= df['charges'].values
sns.displot(charges)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f95739eaf90>
```

As we can see, the data for charges is not normalized. The logarithm of the charges has been taken to demonstrate the normal distribution for this variable

```
charges_norm=np.log(charges)
sns.displot(charges_norm)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f9573771690>
```

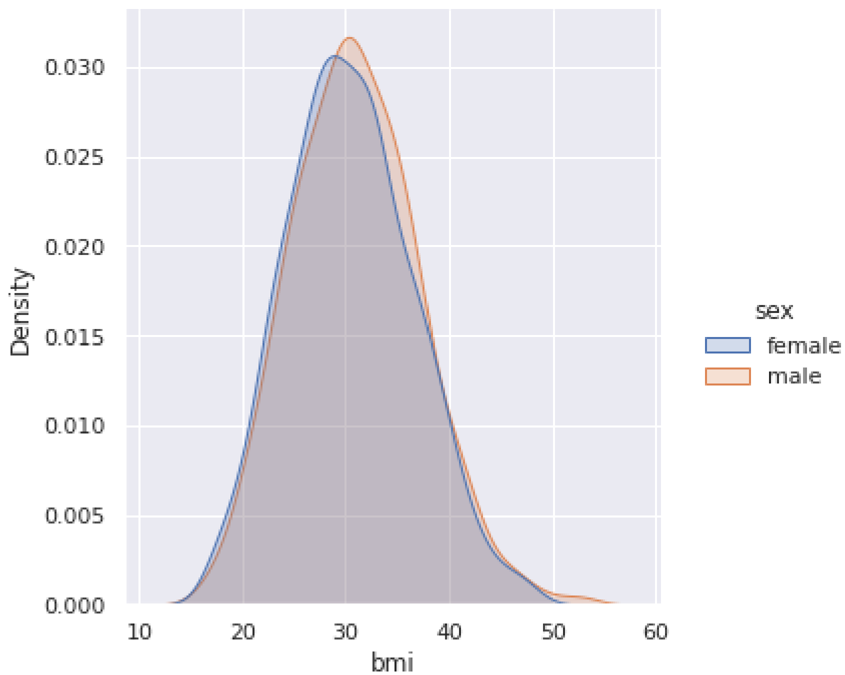


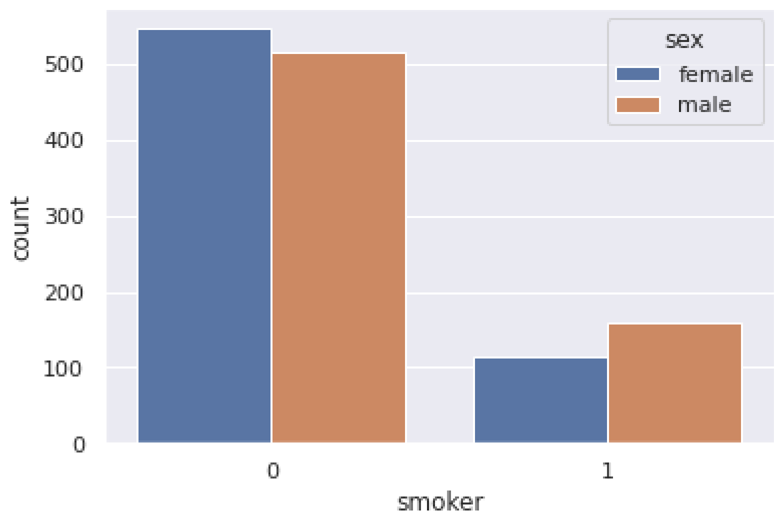
BMI

```
sns.displot(df, x='bmi', hue='sex', kind="kde", fill=True)
plt.show()

sns.countplot(data=df, x='smoker', hue='sex')
plt.show()
```

6





Is there a correlation between body mass and age? In order to better analyze the importance of age and how age impacts other criteria such as BMI, I create an age group range

```
print('The max age is :',df['age'].max())
print('The min age is :',df['age'].min())

age_group_interval =10
df['age_group']=df['age']/age_group_interval
df.loc[df['age_group']<= 2.8, 'age_group_range'] = '18-28'
df.loc[(df['age_group']> 2.8) & (df['age_group']<= 3.8), 'age_group_range'] = '29-38'
df.loc[(df['age_group']> 3.8) & (df['age_group']<= 4.8), 'age_group_range'] = '30-48'
df.loc[(df['age_group']> 4.8) & (df['age_group']<= 5.8), 'age_group_range'] = '49-58'
df.loc[(df['age_group']> 5.8) & (df['age_group']<= 6.8), 'age_group_range'] = '50-68'


df=df.drop('age_group',axis=1)
```

```
The max age is : 64
The min age is : 18
```

```
sns.displot(df, x='bmi', hue='age_group_range', kind="kde", fill=True)
plt.show()
```

The above demonstrate that the mean of the BMI regardless of the age group is almost same, however as the age goes the variation in BMI becomes more

Let's analyse the smoker population

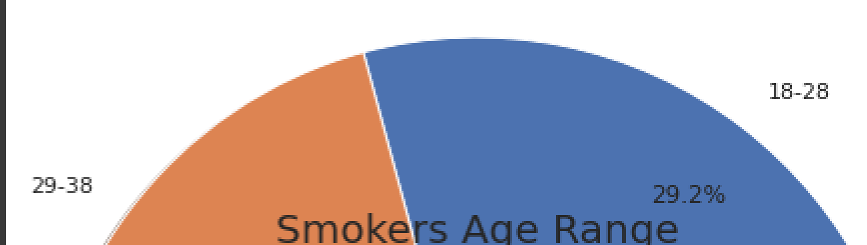

```
data=df[['smoker','age_group_range']].loc[df['smoker']==1].groupby('age_group_range').count()
data

labels = data.index

plt.pie(x=data, autopct="%.1f%%",labels=labels,pctdistance=0.8,shadow=True,radius=2.5)
plt.title('Smokers Age Range', fontsize=20);

```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: MatplotlibDeprecationWarning: Non-1D inputs to pie() ar
```



Now let's analyse if the smoker population pays higher bill with age range consideration

from above we can see the majority of the smoker population are in the age range of 18-28

```

data=df[['age_group_range','charges','smoker']]



sns.catplot(data=data, kind="violin", x='age_group_range', y='charges', hue='smoker', split=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f956dbad610>
```

We can see the medical cost for the smoker people is higher in every age group.

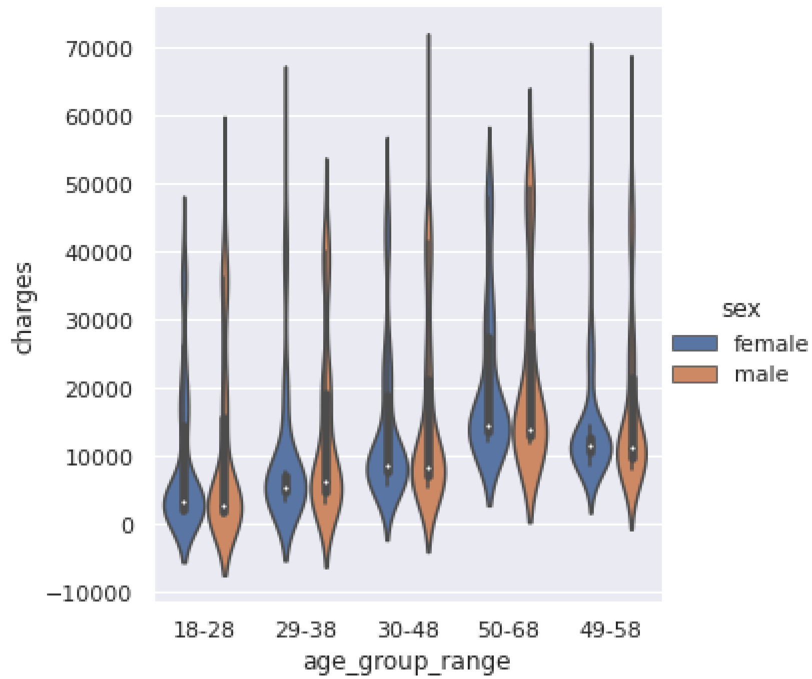
Is there any diffence between male and female in each age group when it comes to the medical bill?

```
data=df[['age_group_range','charges','sex']]

sns.catplot(data=data, kind="violin", x="age_group_range", y="charges", hue="sex", split=False)

```

```
<seaborn.axisgrid.FacetGrid at 0x7f956da30290>
```



The answer is no

```
corrMatrix = df.corr()
```

```
sn.heatmap(corrMatrix, annot=True)
plt.show()
```

Let's start the MI with different regression model

Data Processing

```
# Dropping·the·feature·that·was·added·for·analysis
df=df.drop('age_group_range',axis=1)
df
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	1	southwest	16884.92400
1	18	male	33.770	1	0	southeast	1725.55230
2	28	male	33.000	3	0	southeast	4449.46200
3	33	male	22.705	0	0	northwest	21984.47061
4	32	male	28.880	0	0	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	0	northwest	10600.54830

```
# Creating the array for machine learning
X = df.iloc[:, :6].values
y = df.iloc[:, -1].values
```

```
# There are categorical input in our data set, including sex and region.
# Performing onehotencoder operation on column 1 and 5

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1,5])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

```
#testing onehoteencoder
print(X[0])
print(X[1])
```

```
[1.0 0.0 0.0 0.0 0.0 1.0 19 27.9 0 1]
[0.0 1.0 0.0 0.0 1.0 0.0 18 33.77 1 0]
```

```
#splitting the trainig set and test set using sklearn
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
```

Multiple Linear Regression

```
#fitting the multiple linear regression
rom sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
#creating a prediction matrix using test set
y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[ 4383.68  1646.43]
 [12885.04 11353.23]
 [12589.22  8798.59]
 [13286.23 10381.48]
 [  544.73  2103.08]
 [32117.58 38746.36]
 [12919.04  9304.7 ]
 [12318.62 11658.12]
 [ 3784.29  3070.81]
 [29468.46 19539.24]
 [11002.81 12629.9 ]
 [17539.69 11538.42]
 [ 8681.35  6338.08]
 [ 8349.04  7050.64]
 [ 3130.13  1137.47]
 [10445.84  8968.33]
 [ 3863.74 21984.47]
 [ 6944.63  6414.18]
 [15009.63 28287.9 ]
 [14441.6  13462.52]
 [12543.66  9722.77]
 [32958.73 40932.43]
 [ 9072.64  8026.67]
 [ 8986.86  8444.47]
 [ 3022.86  2203.47]
```

```
 [ 8164.97  6664.69]
 [ 9556.08  8606.22]
 [10743.2   8283.68]
 [ 7694.02  5375.04]
 [ 4373.44  3645.09]
 [14140.94 11674.13]
 [ 5811.79 11737.85]
 [34631.91 24873.38]
 [27009.11 33750.29]
 [33348.14 24180.93]
 [ 9532.97  9863.47]
 [30421.65 36837.47]
 [26648.91 17942.11]
 [15157.78 11856.41]
 [33895.76 39725.52]
 [ 6303.39  4349.46]
 [14059.15 11743.93]
 [10713.45 19749.38]
 [15089.36 12347.17]
 [ 4187.95  4931.65]
 [13106.43 30260.  ]
 [ 4336.2  27724.29]
 [28607.06 34672.15]
 [ 7243.57  9644.25]
 [14269.46 14394.4 ]
 [13282.37 12557.61]
 [12329.61 11881.36]
 [ 1851.87  2352.97]
 [ 8876.28  9101.8 ]
 [26089.18 17178.68]
 [10125.82  3994.18]
 [34218.77 40941.29]
 [14537.7  12644.59]
 [ 3232.08 22395.74]
```

```
# evaluating the model performance using R^2
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

```
0.7623311844057112
```

The multiple linear regression R^2 score was not great, lets try support ventor algorithim

SVR

```
#splitting the training and test set (this was not really required)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
```

```
# Feature Scaling is required for SVM model, 
# reshaping the y_tain to the acceptable format for feature scaling 
y_train= y.reshape(1338,1)
```

```
# performing feature scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X_train = sc_X.fit_transform(X_train)
y_train = sc_y.fit_transform(y_train)
```

```
# fitting the model
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was pa
  y = column_or_1d(y, warn=True)
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
# prediction
y_pred = sc_y.inverse_transform(regressor.predict(sc_X.transform(X_test)))
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
 [14981.42 14390.63]
 [ 5257.35  3732.63]
 [ 6585.64  5846.92]
 [13649.86 12731.  ]
 [14015.56 13616.36]
 [10127.05  8988.16]
 [ 8683.89  7650.77]
 [ 4479.99  3594.17]
 [24033.18 18246.5 ]
 [ 3246.96  2155.68]
 [ 9685.99  8569.86]
 [ 8877.98  7526.71]
 [ 9766.22  9222.4 ]
 [15530.95 14119.62]
 [48214.21 47269.85]
 [ 4385.42  3260.2 ]
 [ 3922.75  2709.11]
 [ 7906.74  6933.24]
 [10495.49  9264.8 ]
 [ 7903.46  7243.81]
 [ 3240.02  2134.9 ]
 [11840.24 11520.1 ]
 [ 9088.56  8233.1 ]
 [ 7037.58  6289.75]
 [ 8208.55  7371.77]
 [13286.45 12094.48]
 [ 7172.65 23563.02]
 [ 7257.04  6457.84]
 [ 2981.65  1615.77]
 [ 7630.58  6600.21]
 [ 7829.68  7046.72]
 [ 4534.72  1984.45]
 [12606.72 11455.28]
 [ 5224.12  4137.52]
 [20759.59 23244.79]
 [ 5911.82 17128.43]
 [ 4881.35  3987.93]

 [ 5150.63  4670.64]
 [38958.27 47291.06]
 [11859.1  10796.35]
 [24961.24 35595.59]
 [ 2327.18  1136.4 ]
 [32469.03 38998.55]
```

```
 [ 3619.39  2459.72]
 [19907.27 21195.82]
 [13472.26 12268.63]
 [ 5906.49  4827.9 ]
 [ 2251.5   1635.73]
 [ 2700.52  1969.61]
 [ 5003.64  4357.04]
 [ 3344.87 10795.94]
 [25768.58 17081.08]
 [14673.74 13887.97]
 [ 5239.82  3579.83]
 [15218.63 14001.29]
 [48027.06 47462.89]
 [ 7789.3   6753.04]
 [12863.09 12096.65]
 [11410.3  10214.64]
 [13506.68 17361.77]
```

```
# evaluating the model
r2_score(y_test, y_pred)
```

```
0.8576258777612062
```

Its a better score, but now lets try with Random Forest

Random Forest

```
# re-creatign array
X = df.iloc[:, :6].values
y = df.iloc[:, -1].values
```

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1,5])], remainder='passthrough')
X = np.array(ct.fit_transform(X))

```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
```

```
X_train[1]
```

```
array([0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 53, 21.4, 1, 0], dtype=object)
```

```
# fitting the model
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators =70, random_state = 0)
regressor.fit(X, y)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=70, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

```
# prediction
y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[ 1908.84  1646.43]
 [11669.4  11353.23]
 [ 9098.64  8798.59]
 [10610.9  10381.48]
 [ 2135.65  2103.08]
 [38894.89 38746.36]
 [ 9677.89  9304.7 ]
 [11642.53 11658.12]
 [ 2994.28  3070.81]
 [19651.01 19539.24]
 [14375.2  12629.9 ]
 [11956.48 11538.42]
 [ 9116.85  6338.08]
 [ 7040.67  7050.64]
```

```
 [ 1587.89  1137.47]
 [ 9407.12  8968.33]
 [14853.59 21984.47]
 [ 6550.67  6414.18]
 [24597.78 28287.9 ]
 [13500.5  13462.52]
 [11711.21  9722.77]
 [41061.55 40932.43]
 [ 9698.11  8026.67]
 [ 8557.22  8444.47]
 [10589.28  2203.47]
 [ 6634.56  6664.69]
 [10373.55  8606.22]
 [ 8808.89  8283.68]
 [ 5871.44  5375.04]
 [ 3645.36  3645.09]
 [11971.93 11674.13]
 [10052.   11737.85]
 [24919.45 24873.38]
 [34134.22 33750.29]
 [25082.28 24180.93]
 [10467.04  9863.47]
 [37328.62 36837.47]
 [17673.69 17942.11]
 [12491.8  11856.41]
 [40962.38 39725.52]
 [ 5472.58  4349.46]
 [11740.45 11743.93]
 [16715.53 19749.38]
 [16906.14 12347.17]
 [ 4846.23  4931.65]
 [25017.41 30260.  ]
 [16601.44 27724.29]
 [35477.51 34672.15]
 [ 9530.48  9644.25]
 [15532.29 14394.4 ]
 [12920.8  12557.61]
 [11986.73 11881.36]
 [ 3222.28  2352.97]
 [ 8976.33  9101.8 ]
 [17554.39 17178.68]
 [ 5326.08  3994.18]
 [41069.32 40941.29]
```

```
[12673.3  12644.59]
```

```
# evaluating
r2_score(y_test, y_pred)
```

```
0.9777613303259217
```

The random forest model is the most accurate algorithim, with ~98% accuracy