PREDICTING THE PRICES

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Load Required dataset

In [2]:
```
data= pd.read_csv(r"C:\Users\M.komala\Downloads\Expenses - Sheet1.csv")
```

Perform EDA

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

Out[3]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.920
1	18	male	33.770	1	no	southeast	1725.552
2	28	male	33.000	3	no	southeast	4449.462
3	33	male	22.705	0	no	northwest	21984.470
4	32	male	28.880	0	no	northwest	3866.855

In [4]:
```
data.region.value_counts()
```

Out[4]:
```
southeast    364
southwest    325
northwest    325
northeast    324
Name: region, dtype: int64
```

In [5]:
```
data.sex.value_counts()
```

Out[5]:
```
male      676
female    662
Name: sex, dtype: int64
```

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

In [7]:
```
data.shape
```

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

In [8]:
```
data.isnull().values.any()
```

Out[8]:
```
False
```

In [9]:
```
data.describe()
```

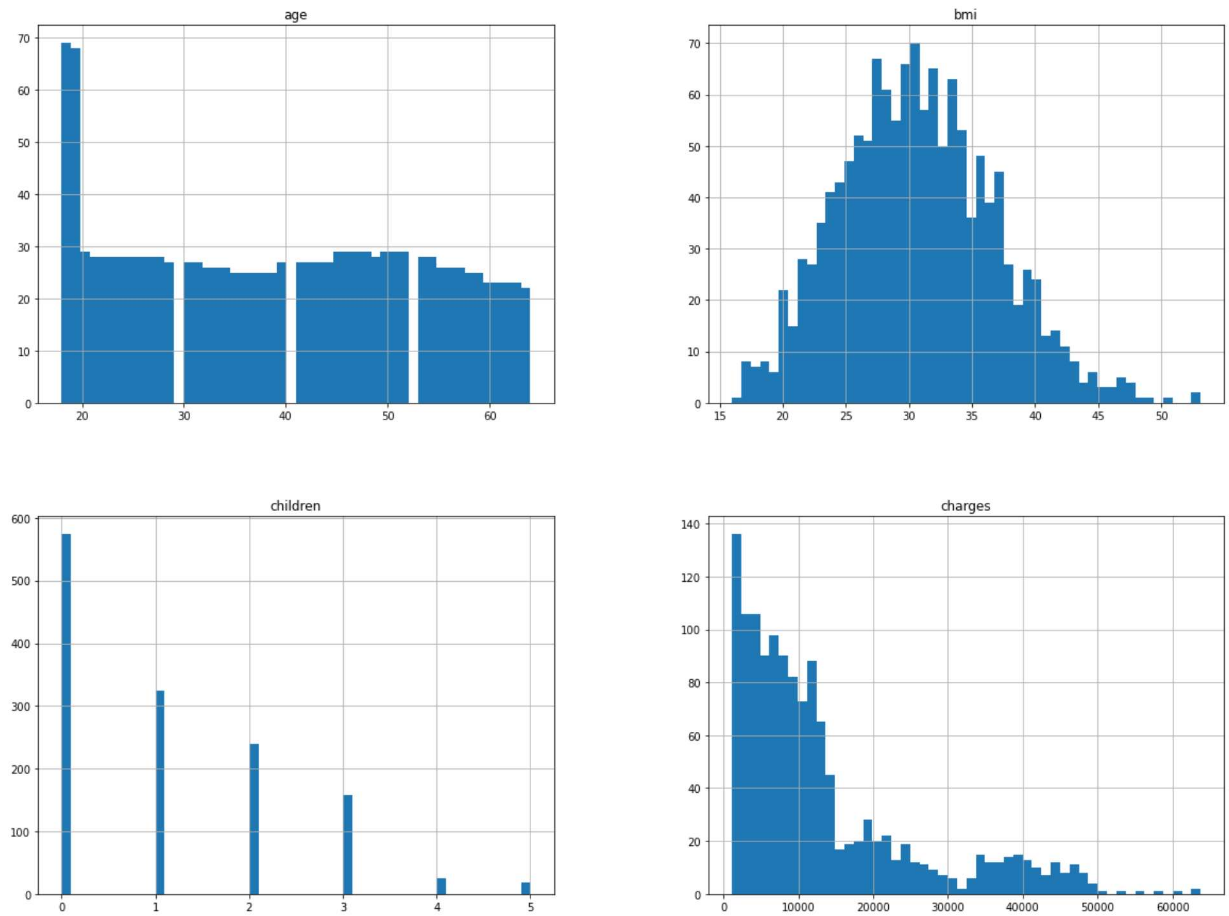
Out[9]:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422346
std	14.049960	6.098187	1.205493	12110.011277
min	18.000000	15.960000	0.000000	1121.874000
25%	27.000000	26.296250	0.000000	4740.287000
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.915000
max	64.000000	53.130000	5.000000	63770.430000

In [10]:
```
data.skew().sort_values(ascending=False)
```

Out[10]:
```
charges     1.515880
children    0.938380
bmi         0.284047
age         0.055673
dtype: float64
```

In [11]:
```
import matplotlib.pyplot as plt
data.hist(bins=50, figsize=(20,15))
plt.show()
```

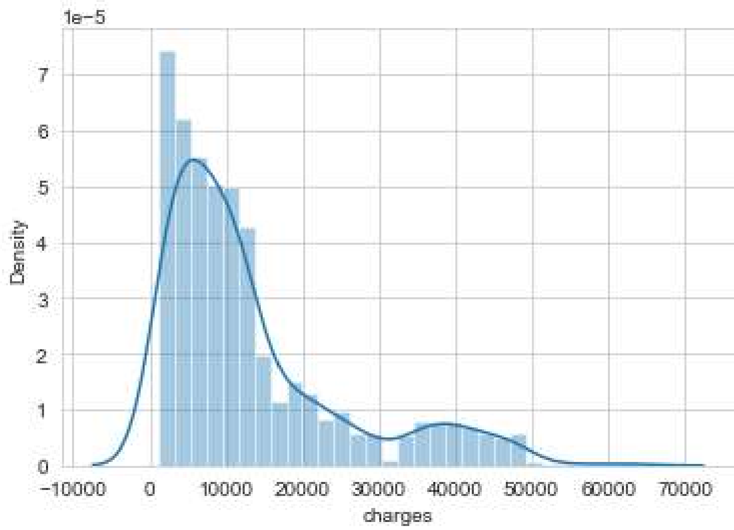


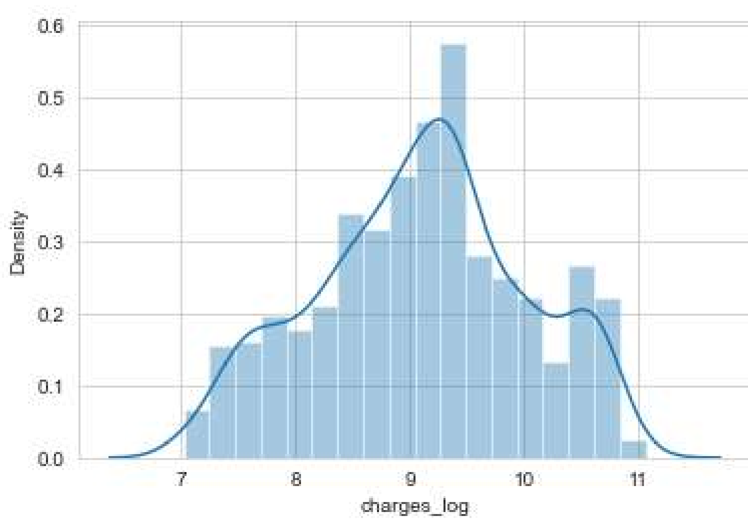
Plot for charges on a histogram.

The distribution of sale prices is right skewed, something that is expected. Here I perform my first bit of feature engineering. I'll apply a log transform to charges to compress outliers making the distribution normal.

Outliers can have devastating effects on models that use loss functions minimising squared error. Instead of removing outliers try applying a transformation.

In [12]:
```
x = data.charges
sns.set_style('whitegrid')
sns.distplot(x)
plt.show()

data['charges_log'] = np.log(data.charges)
x = data.charges_log
sns.distplot(x)
plt.show()
```





In [13]:
```
data= data.drop(columns=['charges'])
```

In [14]:
```
data.head()
```

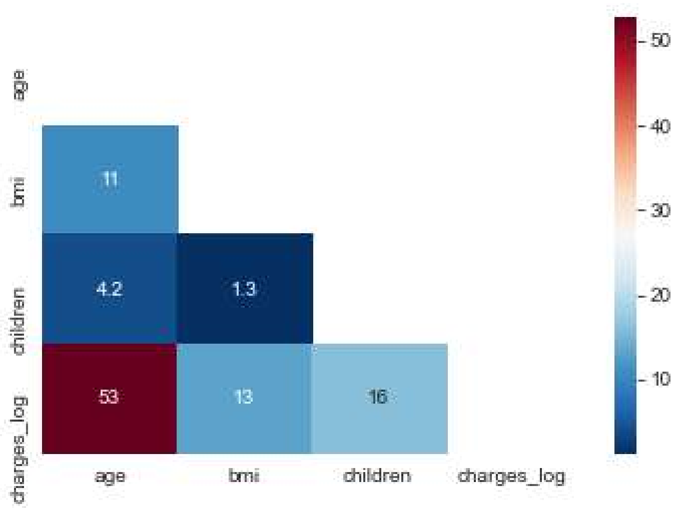
Out[14]:

	age	sex	bmi	children	smoker	region	charges_log
0	19	female	27.900	0	yes	southwest	9.734176
1	18	male	33.770	1	no	southeast	7.453302
2	28	male	33.000	3	no	southeast	8.400538
3	33	male	22.705	0	no	northwest	9.998092
4	32	male	28.880	0	no	northwest	8.260197

Correlation of Data

In [15]:
```
# mask out upper triangle
mask = np.zeros_like(data.corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
# heatmap
sns.heatmap(data.corr()*100,
           cmap='RdBu_r',
           annot = True,
           mask = mask)
```

Out[15]: <AxesSubplot:>



In [16]:
```
print(data.corr())
```

```
                  age       bmi  children  charges_log
age          1.000000  0.109272  0.042469     0.527834
bmi          0.109272  1.000000  0.012759     0.132669
children     0.042469  0.012759  1.000000     0.161336
charges_log  0.527834  0.132669  0.161336     1.000000
```

We have some Categorical Data so apply One hot Encoding

In [17]:
```
data=pd.get_dummies(data)
data.head(2)
```

Out[17]:

	age	bmi	children	charges_log	sex_female	sex_male	smoker_no	smoker_yes	region_north
0	19	27.90	0	9.734176	1	0	0	1	
1	18	33.77	1	7.453302	0	1	1	0	

Superwised Machine Learning Models

Build Linear Regression Model

In [18]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Data Cross Validation by using Train test split Method Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations.

In [19]:
```
# Split X and y

X = data.drop(['charges_log'], axis=1)
y = data[['charges_log']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
```

In [20]:
```
X.head(2)
```

Out[20]:

	age	bmi	children	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_
0	19	27.90	0	1	0	0	1	0	
1	18	33.77	1	0	1	1	0	0	

In [21]:
```
print("Train Data", X_train.shape)
print("Test Data", X_test.shape)
```

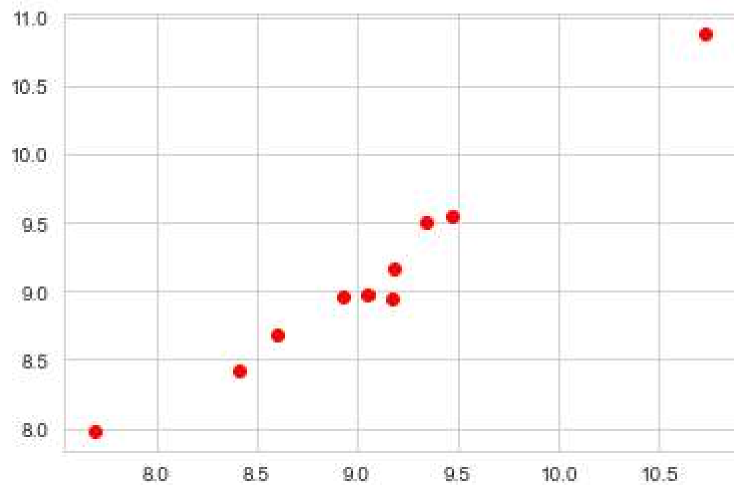
```
Train Data (936, 11)
Test Data (402, 11)
```

In [22]:
```
model= LinearRegression(n_jobs=-1)
model.fit(X_train, y_train)
y_pred= model.predict(X_test)
```

Explore the Predicted Values in graphical format

In [23]:
```
plt.scatter(y_test[0:10], y_pred[0:10], color='red')
```

Out[23]: <matplotlib.collections.PathCollection at 0x2bcf20e4bb0>



In [24]:
```
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from math import sqrt

explained_variance=metrics.explained_variance_score(y_test, y_pred)
mean_absolute_error=metrics.mean_absolute_error(y_test, y_pred)
mse=metrics.mean_squared_error(y_test, y_pred)
mean_squared_log_error=metrics.mean_squared_log_error(y_test, y_pred)
median_absolute_error=metrics.median_absolute_error(y_test, y_pred)
r2_linear=metrics.r2_score(y_test, y_pred)

print('explained_variance: ', round(explained_variance,4))
print('mean_squared_log_error: ', round(mean_squared_log_error,4))
print('r2: ', round(r2_linear,4))
print('MAE: ', round(mean_absolute_error,4))
print('MSE: ', round(mse,4))
print('RMSE: ', round(np.sqrt(mse),4))
```

```
explained_variance:  0.7726
mean_squared_log_error:  0.002
r2:  0.7725
MAE:  0.2798
MSE:  0.2006
RMSE:  0.4478
```

Linear Regression Model Accuracy

In [25]:
```
print('Linear Regression Model Accuracy is', r2_linear.round(2)*100, '%')
```

```
Linear Regression Model Accuracy is 77.0 %
```

In [ ]:

Build Ridge Regression Model

In [26]:
```
from sklearn.linear_model import Ridge
ridgeReg = Ridge(alpha=0.001, normalize=True)
ridgeReg.fit(X_train,y_train)
print(sqrt(mean_squared_error(y_train, ridgeReg.predict(X_train))))
print(sqrt(mean_squared_error(y_test, ridgeReg.predict(X_test))))
r2_ridge=ridgeReg.score(X_test, y_test)
print('R2 Value/Coefficient of Determination: {}'.format(ridgeReg.score(X_test, y
```

```
0.4412842303143742
0.4478136874495315
R2 Value/Coefficient of Determination: 0.7725298858092953
```

Ridge Regression Model Accuracy

In [27]:
```
print('Ridge Regression Model Accuracy is', r2_ridge.round(2)*100, '%')
```

```
Ridge Regression Model Accuracy is 77.0 %
```

In [ ]:

Build Lasso Regression Model

In [28]:
```
from sklearn.linear_model import Lasso
lassoreg = Lasso(alpha=0.001, normalize=True)
lassoreg.fit(X_train,y_train)
r2_lasso=lassoreg.score(X_test, y_test)

print(sqrt(mean_squared_error(y_train, lassoreg.predict(X_train))))
print(sqrt(mean_squared_error(y_test, lassoreg.predict(X_test))))
print('R2 Value/Coefficient of Determination: {}'.format(lassoreg.score(X_test, y
```

```
0.44839238098661743
0.45453134860358596
R2 Value/Coefficient of Determination: 0.7656541315645757
```

Lasso Regression Model Accuracy

In [29]:
```
print('Lasso Regression Model Accuracy is', r2_lasso.round(2)*100, '%')
```

```
Lasso Regression Model Accuracy is 77.0 %
```

In [ ]:

Build Elastic Net Regression Model

In [30]:
```
from sklearn.linear_model import ElasticNet
Elas = ElasticNet(alpha=0.001, normalize=True)
Elas.fit(X_train, y_train)
r2_Elastic=Elas.score(X_test, y_test)
print(sqrt(mean_squared_error(y_train, Elas.predict(X_train))))
print(sqrt(mean_squared_error(y_test, Elas.predict(X_test))))
print('R2 Value/Coefficient of Determination: {}'.format(Elas.score(X_test, y_tes
```

```
0.495963203478928
0.4988882844182607
R2 Value/Coefficient of Determination: 0.7176835209715573
```

Elastic Net Regression Model Accuracy

In [31]:
```
print('Elastic Net Regression Model Accuracy is', r2_Elastic.round(2)*100, '%')
```

```
Elastic Net Regression Model Accuracy is 72.0 %
```

In [ ]:

Build XGB Regressor

The benefit of using ensembles of decision tree methods like gradient boosting is that they can automatically provide estimates of feature importance from a trained predictive model

In [32]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from xgboost import XGBRegressor
from numpy import absolute
```

In [33]:
```
import xgboost as xgb
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error as mse
```

In [34]:
```
# Instantiate an XGBoost object with hyperparameters
xgb_reg = xgb.XGBRegressor(max_depth=3, n_estimators=100, n_jobs=2,
                           objectvie='reg:squarederror', booster='gbtree',
                           random_state=42, learning_rate=0.05)

# Train the model with train data sets
xgb_reg.fit(X_train, y_train)

y_pred = xgb_reg.predict(X_test) # Predictions
y_true = y_test # True values

MSE = mse(y_true, y_pred)
RMSE = np.sqrt(MSE)
```

```
[18:32:15] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.6.
0/src/learner.cc:627:
Parameters: { "objectvie" } might not be used.

  This could be a false alarm, with some parameters getting used by language bi
ndings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such c
ases.
```

In [35]:
```
# trained XGBoost model automatically calculates feature importance on our predic

# These importance scores are available in the feature_importances_ member variab
```

Model importance

Importance provides a score that indicates how useful or valuable each feature was in the construction of the boosted decision trees within the model. The more an attribute is used to make key decisions with decision trees, the higher its relative importance.

This importance is calculated explicitly for each attribute in the dataset, allowing attributes to be ranked and compared to each other.

Importance is calculated for a single decision tree by the amount that each attribute split point improves the performance measure, weighted by the number of observations the node is responsible for. The performance measure may be the purity (Gini index) used to select the split points or another more specific error function.

The feature importances are then averaged across all of the the decision trees within the model.

In [36]:
```
print(xgb_reg.feature_importances_)
```

```
[0.25473997 0.01346319 0.02865839 0.01029253 0.         0.6662077
 0.         0.02195938 0.00175523 0.         0.00292374]
```

In [37]:
```
R_squared = r2_score(y_true, y_pred)

print("\nRMSE: ", np.round(RMSE, 2))
print()
print("R-Squared: ", np.round(R_squared, 2)*100, '%')
```

```
RMSE:  0.38

R-Squared:  84.0 %
```

XGB Regressor Model Accuracy

In [38]:
```
print('XGB Regression Model Accuracy is', R_squared.round(2)*100, '%')
```

```
XGB Regression Model Accuracy is 84.0 %
```

In [ ]:

Decision Tree Model

In [39]:
```
# import the regressor
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
regressor.fit(X_train, y_train)
```

Out[39]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor(random_state=0)
```

In [40]:
```
# predicting value
predictions = regressor.predict(X_test)
```

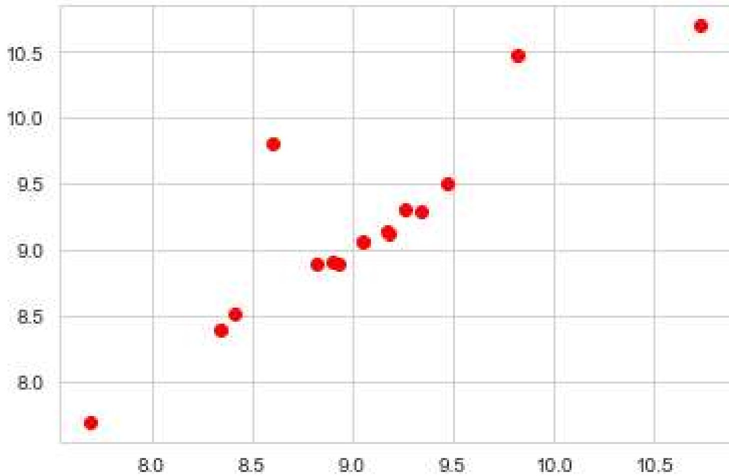
In [41]:
```
r2_deci = r2_score(y_test, predictions)
```

In [42]:
```
r2_deci
```

Out[42]:
```
0.6809362697367225
```

In [43]:
```
plt.scatter(y_test[0:15], predictions[0:15], color='red')
```

Out[43]: <matplotlib.collections.PathCollection at 0x2bcf254a820>



Decision Tree Model Accuracy

In [44]:
```
print('Decision Tree Regression Model Accuracy is', r2_deci.round(2)*100, '%')
```

```
Decision Tree Regression Model Accuracy is 68.0 %
```

In [ ]:

Random Forest Model

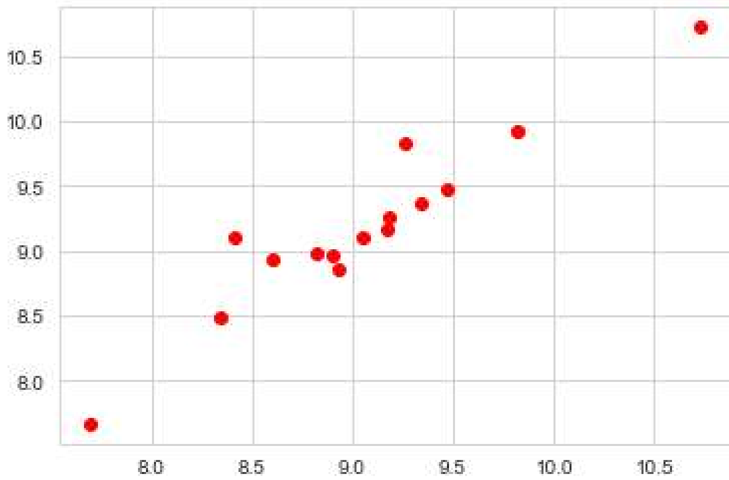
In [45]:
```
# Fitting Random Forest Regression to the dataset
# import the regressor
from sklearn.ensemble import RandomForestRegressor

# create regressor object
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

# fit the regressor with x and y data
regressor.fit(X_train, y_train)
```

Out[45]:
```
▾ RandomForestRegressor
RandomForestRegressor(random_state=0)
```

In [46]:
```
Predictionsrandomforest = regressor.predict(X_test) # test the output by changing
```

In [47]:
```
plt.scatter(y_test[0:15], Predictionsrandomforest[0:15], color='red')
```

Out[47]: <matplotlib.collections.PathCollection at 0x2bcf26d6fa0>



In [48]:
```
r2 = r2_score(y_test,Predictionsrandomforest)
r2
```

Out[48]: 0.8103091729430505

In [49]:
```
print('Random Forest Regression Model Accuracy is ', r2.round(2)*100, '%')
```

```
Random Forest Regression Model Accuracy is  81.0 %
```

In [ ]:

Conclusion

By Comparing the all Models XGB Regressor is the best fith 84% Accuracy

In [ ]:

In [ ]: