In [89]:

```
!pip install numpy
!pip install pandas
!pip install seaborn
```

```
Requirement already satisfied: numpy in c:\users\agrocel\anaconda3\lib\site-
packages (1.24.3)
Requirement already satisfied: pandas in c:\users\agrocel\anaconda3\lib\site
-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\agrocel\an
aconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\agrocel\anaconda3\li
b\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\agrocel\anaconda3
\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in c:\users\agrocel\anaconda3\l
ib\site-packages (from pandas) (1.24.3)
Requirement already satisfied: six>=1.5 in c:\users\agrocel\anaconda3\lib\si
te-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: seaborn in c:\users\agrocel\anaconda3\lib\sit
e-packages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in c:\users\agrocel\anac
onda3\lib\site-packages (from seaborn) (1.24.3)
Requirement already satisfied: pandas>=0.25 in c:\users\agrocel\anaconda3\li
b\site-packages (from seaborn) (2.0.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in c:\users\agrocel\a
naconda3\lib\site-packages (from seaborn) (3.7.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\agrocel\anaconda
3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\agrocel\anaconda3\li
b\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\agrocel\anacond
a3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\agrocel\anacond
a3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\agrocel\anaconda3
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\agrocel\anaconda3\l
ib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (9.4.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\agrocel\ana
conda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\agrocel\anac
onda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\agrocel\anaconda3\li
b\site-packages (from pandas>=0.25->seaborn) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\agrocel\anaconda3
\lib\site-packages (from pandas>=0.25->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\agrocel\anaconda3\lib\si
te-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn)
(1.16.0)
```

In [90]:

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

Data Reading

In [91]:

```
df1 = pd.read_excel(r'Prediction data_Final.xlsx')
```

In [92]:

```
df1
```

Out[92]:

	Months	Agrocel Purchase Rate of Sulphur Granules(Rupees/ KG)	Market Crude Price ($ per Barrel)	Avg Temp (Deg C)	Precip(cm)	Exchange rate	Purchase Rate(Sulphuric Acid)	Imp
0	2021-04-01	23.31	63.400000	29.06	0.070	74.055	9.50	18
1	2021-05-01	22.66	66.950000	30.54	4.490	72.518	9.60	17
2	2021-06-01	22.71	71.980000	32.34	1.110	74.365	8.20	15
3	2021-07-01	24.19	73.540000	31.42	18.800	74.344	8.50	15
4	2021-08-01	23.09	69.800000	30.52	9.040	72.951	8.50	17
5	2021-09-01	22.50	73.130000	28.63	10.850	74.170	7.50	17
6	2021-10-01	25.90	82.110000	26.24	3.680	74.935	16.50	17
7	2021-11-01	28.12	80.640000	19.07	0.000	75.100	14.00	18
8	2021-12-01	29.60	73.300000	15.12	0.140	74.486	11.40	19
9	2022-01-01	30.70	84.670000	12.68	2.850	74.556	10.40	19
10	2022-02-01	32.71	94.070000	17.05	0.890	75.520	11.30	17
11	2022-03-01	33.40	112.870000	25.56	0.000	75.928	13.00	19
12	2022-04-01	45.54	102.970000	32.51	0.014	76.535	14.00	21
13	2022-05-01	46.13	109.510000	33.34	1.750	77.598	14.00	19
14	2022-06-01	47.73	116.010000	34.38	4.170	78.965	13.30	19
15	2022-07-01	45.25	105.490000	31.05	5.350	79.351	13.42	20
16	2022-08-01	25.55	97.400000	30.22	1.650	79.511	13.42	17
17	2022-09-01	19.50	90.710000	29.45	5.580	81.522	5.50	16
18	2022-10-01	18.34	91.700000	25.33	3.860	82.780	0.00	18
19	2022-11-01	20.76	87.550000	20.84	0.000	81.374	7.88	19
20	2022-12-01	23.30	78.100000	15.08	3.530	82.745	7.88	19

	Months	Agrocel Purchase Rate of Sulphur Granules(Rupees/ KG)	Market Crude Price ($ per Barrel)	Avg Temp (Deg C)	Precip(cm)	Exchange rate	Purchase Rate(Sulphuric Acid)	Imp
21	2023-01-01	23.25	80.920000	12.79	3.430	81.769	6.00	20
22	2023-02-01	21.12	82.280000	19.61	0.000	82.645	6.03	19
23	2023-03-01	19.21	78.540000	23.33	1.760	82.185	6.50	20
24	2023-04-01	19.20	83.755358	29.69	1.130	81.745	6.20	20
25	2023-05-01	13.80	74.981548	29.57	0.930	82.690	6.22	24
26	2023-06-01	14.46	74.928252	33.70	2.150	82.096	0.00	19
27	2023-07-01	11.64	80.368492	33.90	4.320	82.240	0.00	19
28	2023-08-01	16.00	86.426704	30.70	9.310	82.702	0.00	18
29	2023-09-01	13.70	93.539339	30.65	7.800	83.030	6.22	17
30	2023-10-01	14.30	90.080343	29.30	2.620	83.256	5.20	18
31	2023-11-01	13.74	83.455368	26.26	0.230	83.357	5.48	18

Data Preprocessing

In [93]:

```
df1.describe()
```

Out[93]:

	Months	Agrocel Purchase Rate of Sulphur Granules(Rupees/ KG)	Market Crude Price ($ per Barrel)	Avg Temp (Deg C)	Precip(cm)	Exchange rate	Purchase Rate(Sulphuric Acid)
count	32	32.000000	32.000000	32.000000	32.000000	32.00000	32.000000
mean	2022-07-16 21:00:00	24.731562	85.474231	26.560313	3.484500	78.78200	8.301562
min	2021-04-01 00:00:00	11.640000	63.400000	12.680000	0.000000	72.51800	0.000000
25%	2021-11-23 12:00:00	18.985000	74.968224	22.707500	0.725000	74.84025	6.022500
50%	2022-07-16 12:00:00	22.900000	82.867684	29.375000	2.385000	79.43100	8.040000
75%	2023-03-08 18:00:00	28.490000	92.159835	30.787500	4.362500	82.34125	11.800000
max	2023-11-01 00:00:00	47.730000	116.010000	34.380000	18.800000	83.35700	16.500000
std	NaN	9.930650	13.278095	6.560473	4.068915	3.80809	4.469140

In [94]:

```
df1.isnull().sum()
```

Out[94]:

```
Months                                                  0
Agrocel Purchase Rate of Sulphur Granules(Rupees/ KG)   0
Market Crude Price ($ per Barrel)                       0
Avg Temp (Deg C)                                        0
Precip(cm)                                              0
Exchange rate                                           0
Purchase Rate(Sulphuric Acid)                           0
Crude Quantity Imported(1000 MT)                        0
Total Crude processed (1000 MT)                         0
Elemental Sulphur Import Quantity in 1000s(KG)s(Market) 0
Elemental Sulphur Export Quantity in 1000s(KG)s(Market) 0
dtype: int64
```

In [95]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 11 columns):
 #   Column                                                Non-Null Count
Dtype
---  ------                                                --------------
-----
 0   Months                                                32 non-null
datetime64[ns]
 1   Agrocel Purchase Rate of Sulphur Granules(Rupees/ KG)    32 non-null
float64
 2   Market Crude Price ($ per Barrel)                     32 non-null
float64
 3   Avg Temp (Deg C)                                      32 non-null
float64
 4   Precip(cm)                                            32 non-null
float64
 5   Exchange rate                                         32 non-null
float64
 6   Purchase Rate(Sulphuric Acid)                         32 non-null
float64
 7   Crude Quantity Imported(1000 MT)                      32 non-null
float64
 8   Total Crude processed (1000 MT)                       32 non-null
int64
 9   Elemental Sulphur Import Quantity in 1000s(KG)s(Market)  32 non-null
int64
 10  Elemental Sulphur Export Quantity in 1000s(KG)s(Market)  32 non-null
int64
dtypes: datetime64[ns](1), float64(7), int64(3)
memory usage: 2.9 KB
```

Changing data type from datetime to object

In [96]:

```
df1['Months'] = df1['Months'].dt.strftime('%B %Y')
```

In [97]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 11 columns):
 #   Column                                                Non-Null Count
Dtype
---  ------                                                --------------
-----
 0   Months                                                32 non-null
object
 1   Agrocel Purchase Rate of Sulphur Granules(Rupees/ KG)    32 non-null
float64
 2   Market Crude Price ($ per Barrel)                     32 non-null
float64
 3   Avg Temp (Deg C)                                      32 non-null
float64
 4   Precip(cm)                                            32 non-null
float64
 5   Exchange rate                                         32 non-null
float64
 6   Purchase Rate(Sulphuric Acid)                         32 non-null
float64
 7   Crude Quantity Imported(1000 MT)                      32 non-null
float64
 8   Total Crude processed (1000 MT)                       32 non-null
int64
 9   Elemental Sulphur Import Quantity in 1000s(KG)s(Market)  32 non-null
int64
 10  Elemental Sulphur Export Quantity in 1000s(KG)s(Market)  32 non-null
int64
dtypes: float64(7), int64(3), object(1)
memory usage: 2.9+ KB
```

Changing data type from object to float64

In [98]:

```
# Function to convert month names to numerical values
def convert_month_to_float(month_string):
    month_name, year = month_string.split()
    month_mapping = {
        'January': 1, 'February': 2, 'March': 3, 'April': 4, 'May': 5, 'June': 6,
        'July': 7, 'August': 8, 'September': 9, 'October': 10, 'November': 11, 'December':
    }
    month_number = month_mapping[month_name]
    return float(str(month_number) + '.' + year)

# Apply the conversion function to the 'Month' column
df1['Months'] = df1['Months'].apply(convert_month_to_float)
```

In [99]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 11 columns):
 #   Column                                                Non-Null Count
Dtype
---  ------                                                --------------
-----
 0   Months                                                32 non-null
float64
 1   Agrocel Purchase Rate of Sulphur Granules(Rupees/ KG)    32 non-null
float64
 2   Market Crude Price ($ per Barrel)                     32 non-null
float64
 3   Avg Temp (Deg C)                                      32 non-null
float64
 4   Precip(cm)                                            32 non-null
float64
 5   Exchange rate                                         32 non-null
float64
 6   Purchase Rate(Sulphuric Acid)                         32 non-null
float64
 7   Crude Quantity Imported(1000 MT)                      32 non-null
float64
 8   Total Crude processed (1000 MT)                       32 non-null
int64
 9   Elemental Sulphur Import Quantity in 1000s(KG)s(Market)  32 non-null
int64
 10  Elemental Sulphur Export Quantity in 1000s(KG)s(Market)  32 non-null
int64
dtypes: float64(8), int64(3)
memory usage: 2.9 KB
```

In [100]:

```
df1 = df1.rename(columns={'Agrocel Purchase Rate of Sulphur Granules(Rupees/ KG)': 'Sulphur
```

In [101]:

```
df1 = df1.rename(columns={'Elemental Sulphur Import Quantity in 1000s(KG)s(Market)': 'Sulph
```

In [102]:

```
df1 = df1.rename(columns={'Elemental Sulphur Export Quantity in 1000s(KG)s(Market)': 'Sulph
```

Correlation

In [103]:

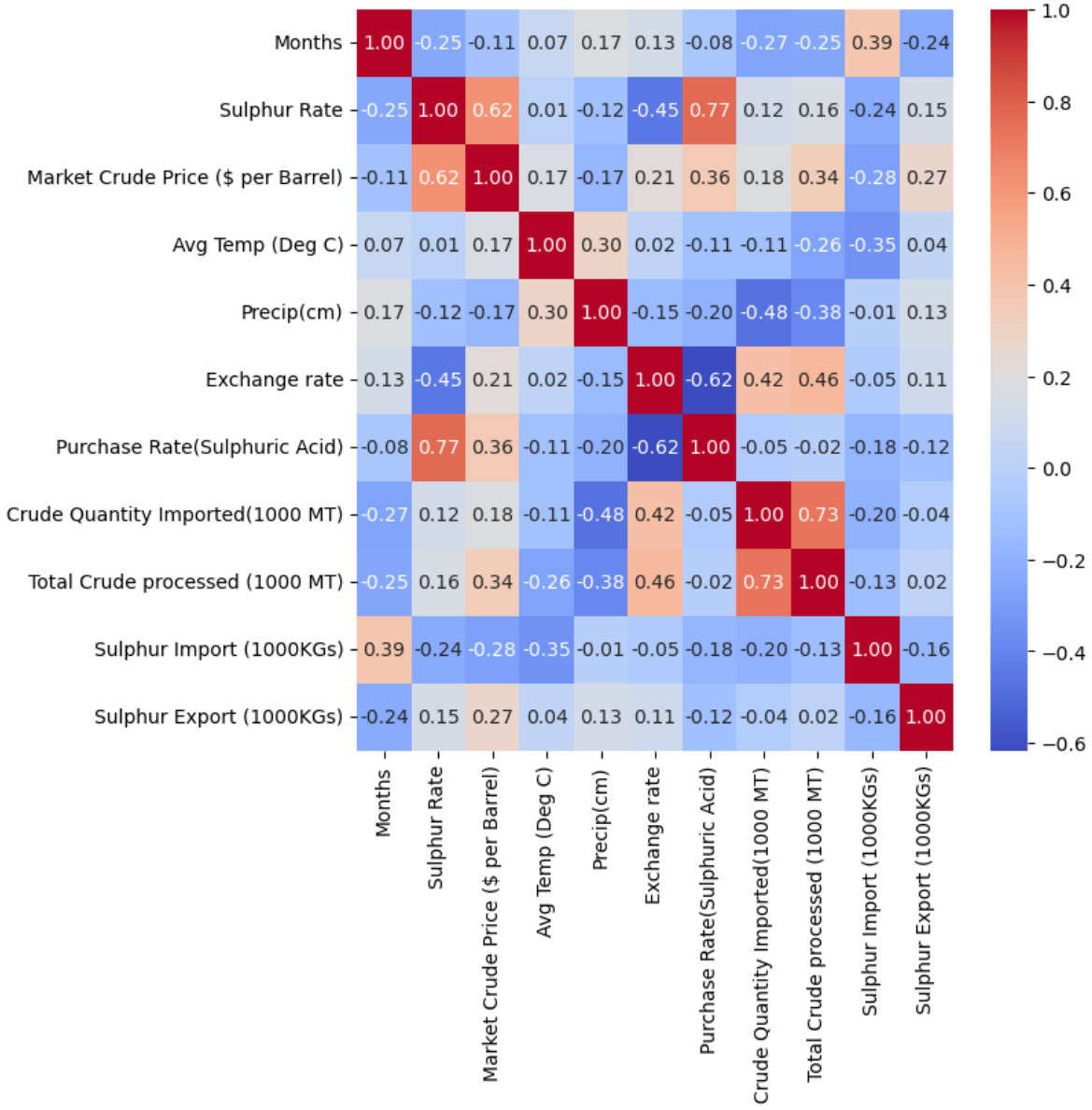
```
df1.corr()
```

Out[103]:

	Months	Sulphur Rate	Market Crude Price ($ per Barrel)	Avg Temp (Deg C)	Precip(cm)	Exchange rate	Purchase Rate(Sulphuric Acid
Months	1.000000	-0.250411	-0.110774	0.070246	0.172698	0.134183	-0.07824
Sulphur Rate	-0.250411	1.000000	0.623216	0.013281	-0.122135	-0.450460	0.77168
Market Crude Price ($ per Barrel)	-0.110774	0.623216	1.000000	0.167900	-0.170568	0.214295	0.35672
Avg Temp (Deg C)	0.070246	0.013281	0.167900	1.000000	0.298192	0.022647	-0.11332
Precip(cm)	0.172698	-0.122135	-0.170568	0.298192	1.000000	-0.153953	-0.19858
Exchange rate	0.134183	-0.450460	0.214295	0.022647	-0.153953	1.000000	-0.61740
Purchase Rate(Sulphuric Acid)	-0.078249	0.771681	0.356728	-0.113321	-0.198581	-0.617403	1.00000
Crude Quantity Imported(1000 MT)	-0.268726	0.116241	0.180082	-0.113823	-0.475184	0.417285	-0.04742
Total Crude processed (1000 MT)	-0.254112	0.158050	0.339786	-0.263830	-0.383673	0.455731	-0.01914
Sulphur Import (1000KGs)	0.386340	-0.237865	-0.283973	-0.351559	-0.012546	-0.052269	-0.17964
Sulphur Export (1000KGs)	-0.236995	0.146470	0.272977	0.037984	0.132405	0.107076	-0.12354

In [104]:

```
corr_matrix = df1.corr()
plt.figure(figsize = (7, 7))
sns.heatmap(corr_matrix, annot = True, cmap = 'coolwarm', fmt = ".2f")
plt.show()
```



VIF

In [105]:

```
X = df1.drop(['Sulphur Rate'], axis=1)
```

In [106]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor


def calc_vif(X):


    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]


    return(vif)
calc_vif(X)
```

Out[106]:

	variables	VIF
0	Months	8.944693
1	Market Crude Price ($ per Barrel)	83.473301
2	Avg Temp (Deg C)	25.398391
3	Precip(cm)	2.560515
4	Exchange rate	597.351031
5	Purchase Rate(Sulphuric Acid)	7.342513
6	Crude Quantity Imported(1000 MT)	322.890549
7	Total Crude processed (1000 MT)	715.925469
8	Sulphur Import (1000KGs)	8.511540
9	Sulphur Export (1000KGs)	6.820671

In [107]:

```
X = df1.drop(['Exchange rate','Total Crude processed (1000 MT)','Sulphur Rate','Avg Temp (D
```

In [108]:

```
def calc_vif(X):


    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]


    return(vif)
calc_vif(X)
```

Out[108]:

	variables	VIF
0	Months	7.190827
1	Market Crude Price ($ per Barrel)	21.631999
2	Precip(cm)	1.910044
3	Purchase Rate(Sulphuric Acid)	5.708690
4	Sulphur Import (1000KGs)	6.320999
5	Sulphur Export (1000KGs)	6.648293

In [109]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [110]:

```
LRdata = df1.drop(['Exchange rate','Avg Temp (Deg C)','Total Crude processed (1000 MT)'], a
```

In [111]:

```
LRdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 8 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Months                            32 non-null     float64
 1   Sulphur Rate                      32 non-null     float64
 2   Market Crude Price ($ per Barrel) 32 non-null     float64
 3   Precip(cm)                        32 non-null     float64
 4   Purchase Rate(Sulphuric Acid)     32 non-null     float64
 5   Crude Quantity Imported(1000 MT)  32 non-null     float64
 6   Sulphur Import (1000KGs)          32 non-null     int64
 7   Sulphur Export (1000KGs)          32 non-null     int64
dtypes: float64(6), int64(2)
memory usage: 2.1 KB
```

Linear Regression

In [112]:

```
## Linear Regression on dropped columns

x_lr = LRdata[['Months','Market Crude Price ($ per Barrel)','Precip(cm)','Purchase Rate(Sul
y_lr = LRdata['Sulphur Rate']


X_train, X_test,Y_train, Y_test = train_test_split(x_lr, y_lr, test_size=0.2,random_state=3


# Create a LinearRegression model
lr = LinearRegression()

# Fit the model to the data
lr.fit(X_train,Y_train)
print("Intercept:", lr.intercept_)
print("Coefficients:", lr.coef_)
```

```
Intercept: -9.86980941093081
Coefficients: [-6.55124410e-01  2.63325372e-01  1.38320053e-01  1.47193296e+
00
  1.32281707e-05  1.25373199e-05]
```

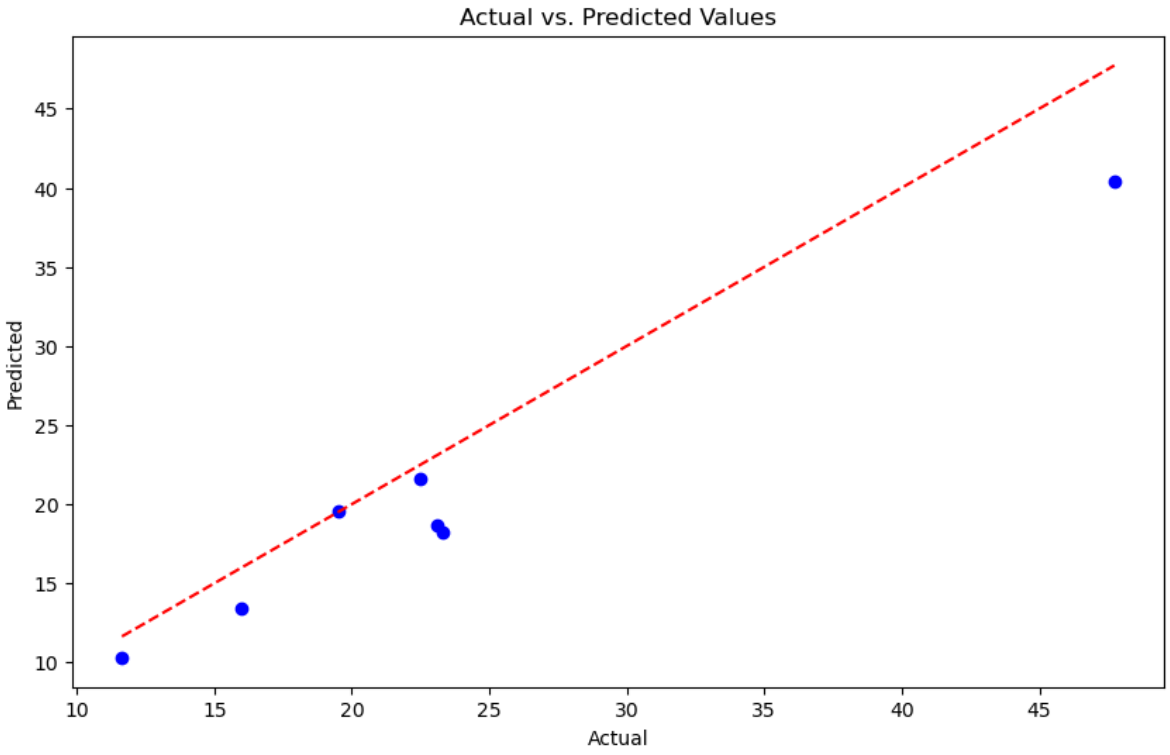
In [113]:

```
print("X_train dimension:", X_train.ndim)
print("X_test dimension:", X_test.ndim)
print("y_train dimension:", Y_train.ndim)
print("y_test dimension:", Y_test.ndim)
```

```
X_train dimension: 2
X_test dimension: 2
y_train dimension: 1
y_test dimension: 1
```

In [114]:

```
Y_pred = lr.predict(X_test)

# Step 5: Graphical Representation
plt.figure(figsize=(10, 6))

# Scatter plot of actual vs. predicted values
plt.scatter(Y_test, Y_pred, color='blue')
plt.plot([min(Y_test), max(Y_test)], [min(Y_test), max(Y_test)], '--', color='red')  # Diag
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs. Predicted Values')
plt.show()
```



In [115]:

```
## Linear Regression on whole dataset

x_var = df1[['Months','Market Crude Price ($ per Barrel)','Avg Temp (Deg C)','Precip(cm)','
y_var = df1['Sulphur Rate']



x_train, x_test,y_train, y_test = train_test_split(x_var, y_var, test_size=0.2,random_state


# Create a LinearRegression model
model = LinearRegression()


# Fit the model to the data
model.fit(x_train,y_train)
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)
```

```
Intercept: 75.22989238977476
Coefficients: [-1.17677011e-01  4.14958372e-01  4.13424756e-02  2.46866530e-
01
 -1.62551820e+00  4.51729550e-01  1.85119114e-03 -8.69435297e-05
  8.44261950e-06  2.50049837e-05]
```

In [116]:

```
print("X_train dimension:", x_train.ndim)
print("X_test dimension:", x_test.ndim)
print("y_train dimension:", y_train.ndim)
print("y_test dimension:", y_test.ndim)
```

```
X_train dimension: 2
X_test dimension: 2
y_train dimension: 1
y_test dimension: 1
```

In [117]:

```
from sklearn.metrics import mean_squared_error, r2_score

y_pred = model.predict(x_test)

mse = mean_squared_error(y_test, model.predict(x_test))
print("Mean Square Error:", mse)


test_score = r2_score(y_test, model.predict(x_test)) * 100
train_score = r2_score(y_train, model.predict(x_train)) * 100

print("Test_Score:", test_score)
print("Train_Score:", train_score )
```

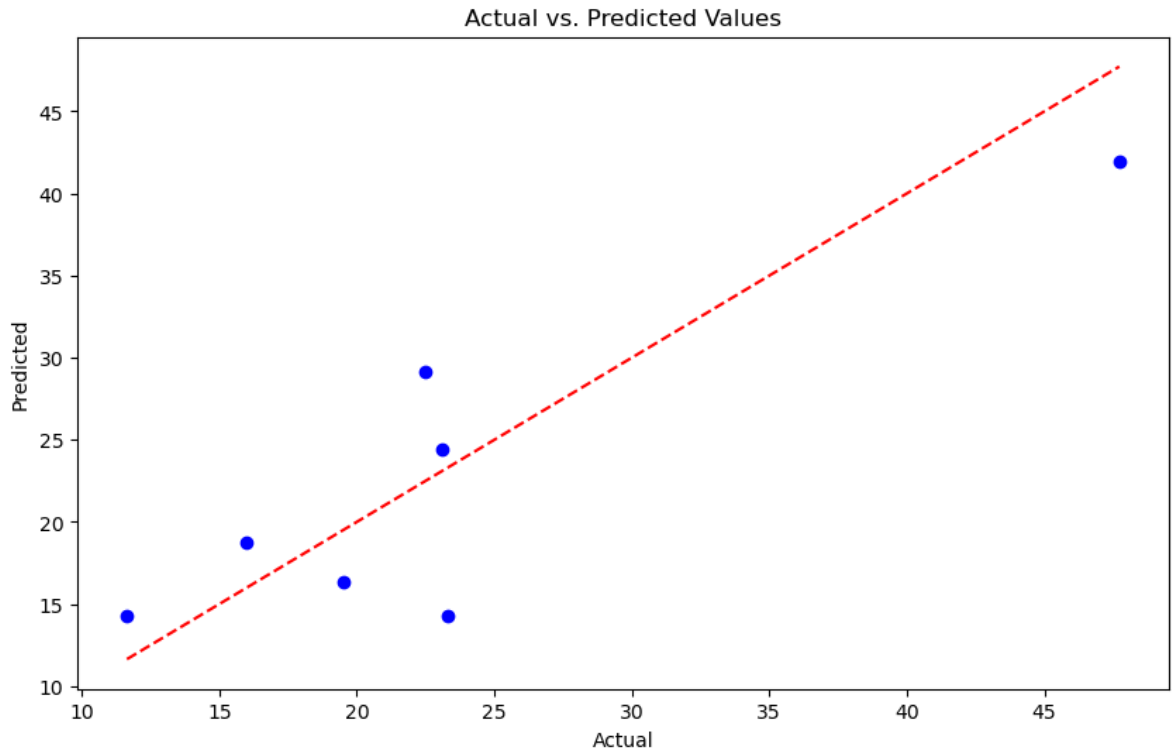
```
Mean Square Error: 26.53586357359308
Test_Score: 76.8139384152057
Train_Score: 84.24313455059583
```

In [118]:

```
import pandas as pd

# Define column names
columns = ['Model', 'Training Accuracy %', 'Testing Accuracy %']

# Create empty DataFrame with column names
results_df = pd.DataFrame(columns=columns)

# Now, let's add a new row to the DataFrame
new_row = ['Linear Regression', train_score, test_score]

# Create a DataFrame for the new row
new_row_df = pd.DataFrame([new_row], columns=results_df.columns)

# Append the new row to the existing DataFrame
results_df = pd.concat([results_df, new_row_df], ignore_index=True)
```

In [119]:

```
# Step 5: Graphical Representation
plt.figure(figsize=(10, 6))

# Scatter plot of actual vs. predicted values
plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--', color='red')  # Diag
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs. Predicted Values')
plt.show()
```



In [120]:

```
# 1. Prepare your new data
# For example, if your new data has features 'feature1' and 'feature2', you need to create
x_validation = pd.DataFrame([[12.2023,85,26,0.2,83,5.4,18593,21668,219096,95300]])

# 2. Use the predict() method of your fitted linear regression model
predictions = model.predict(x_validation)

# Now 'predictions' contains the predicted values for the new data
print(predictions)
```

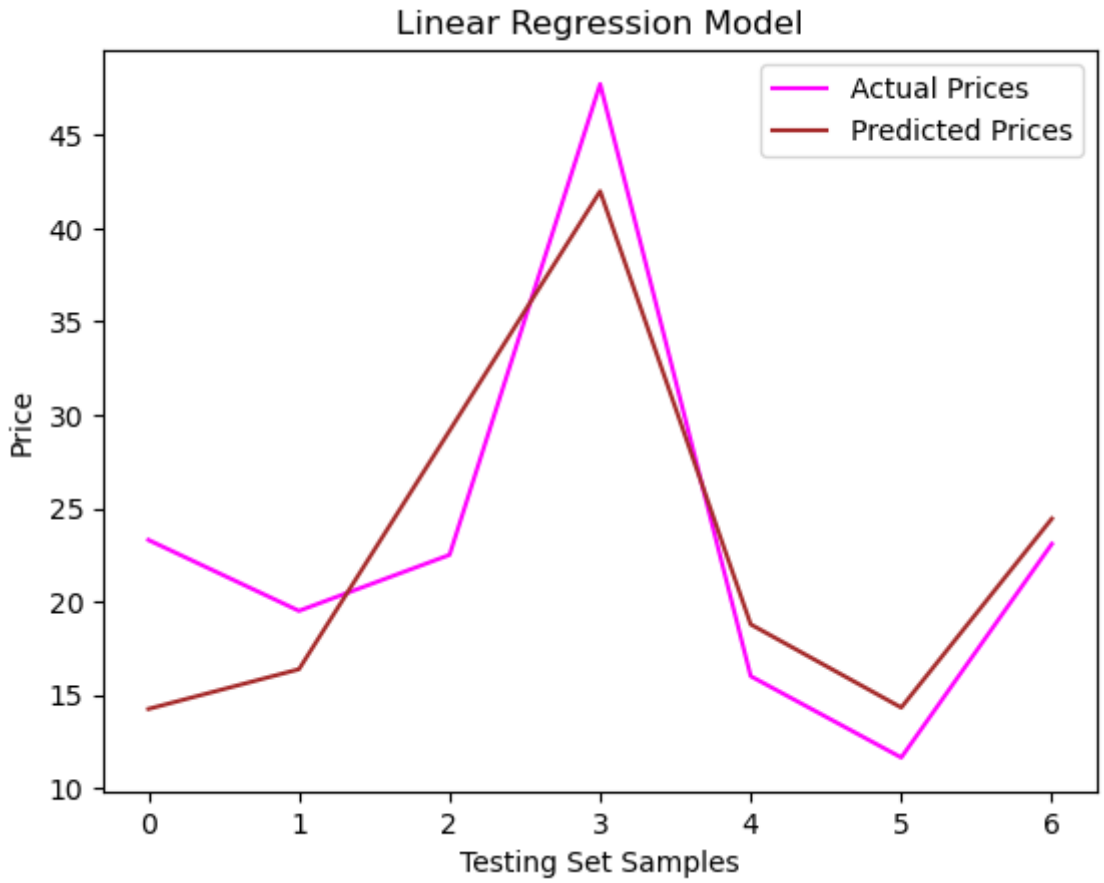
```
[14.47905439]

C:\Users\Agrocel\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but LinearRegression was fitted with
feature names
  warnings.warn(
```

In [121]:

```
Prediction_lr = model.predict(x_test)
```

In [163]:

```
xx = range(len(y_test))
# Plot actual values
plt.plot(xx, y_test, color='magenta', label='Actual Prices')

# Plot predicted values
plt.plot(xx, Prediction_lr, color='brown', label='Predicted Prices')

# Add labels and title
plt.xlabel('Testing Set Samples')
plt.ylabel('Price')
plt.title('Linear Regression Model')
plt.legend()

# Show the plot
plt.show()
```



Polynomial Regression

In [123]:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

# Assuming 'X' contains your original features and 'y' contains the target variable

# Define the degree of the polynomial
degree = 3 # You can change this as needed

# Create a PolynomialFeatures object with the specified degree
poly_features = PolynomialFeatures(degree=degree)

# Create a pipeline to combine polynomial feature generation with linear regression
poly_regression_model = make_pipeline(poly_features, LinearRegression())

# Fit the polynomial regression model to your data
poly_regression_model.fit(x_train,y_train )

y_pred = poly_regression_model.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 1060.2850867105712
R-squared: -8.264381108129697
```

In [124]:

```
test_score = r2_score(y_test, poly_regression_model.predict(x_test)) * 100
train_score = r2_score(y_train, poly_regression_model.predict(x_train)) * 100

print("Test_Score:", test_score)
print("Train_Score:", train_score )
```

```
Test_Score: -826.4381108129697
Train_Score: 100.0
```

In [125]:

```
# Now, let's add a new row to the DataFrame
new_row = ['Polynomial Regression', train_score, test_score]

# Create a DataFrame for the new row
new_row_df = pd.DataFrame([new_row], columns=results_df.columns)

# Append the new row to the existing DataFrame
results_df = pd.concat([results_df, new_row_df], ignore_index=True)
```

Random Forest Regression

In [126]:

```
from sklearn.ensemble import RandomForestRegressor

# Create a Random Forest Regression model
random_forest_model = RandomForestRegressor(n_estimators=3, random_state=2)  # You can adju

# Fit the model to your data
random_forest_model.fit(x_train, y_train)  # Assuming X_train contains your features and y_
```

Out[126]:

```
RandomForestRegressor(n_estimators=3, random_state=2)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [127]:

```
# Use the fitted model to make predictions on the test data

mse = mean_squared_error(y_test, random_forest_model.predict(x_test))  # Calculate mean squ
print ("Mean Square Error;",mse)

test_score = r2_score(y_test, random_forest_model.predict(x_test)) * 100
train_score = r2_score(y_train, random_forest_model.predict(x_train)) * 100

print("Test_Score:", test_score)
print("Train_Score:", train_score )
```

```
Mean Square Error; 9.448998412698407
Test_Score: 91.74381272710983
Train_Score: 90.52632257206604
```

In [128]:

```
# Now, let's add a new row to the DataFrame
new_row = ['Random Forest Regression', train_score, test_score]

# Create a DataFrame for the new row
new_row_df = pd.DataFrame([new_row], columns=results_df.columns)

# Append the new row to the existing DataFrame
results_df = pd.concat([results_df, new_row_df], ignore_index=True)
```

In [129]:

```
# For example, if your new data has features 'feature1' and 'feature2', you need to create
x_validation = pd.DataFrame([[12.2023,85,26,0.2,83,5.4,18593,21668,219096,95300]])

# 2. Use the predict() method of your fitted linear regression model
predictions = random_forest_model.predict(x_validation)

# Now 'predictions' contains the predicted values for the new data
print(predictions)
```

```
[16.44]

C:\Users\Agrocel\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but RandomForestRegressor was fitted
with feature names
  warnings.warn(
```

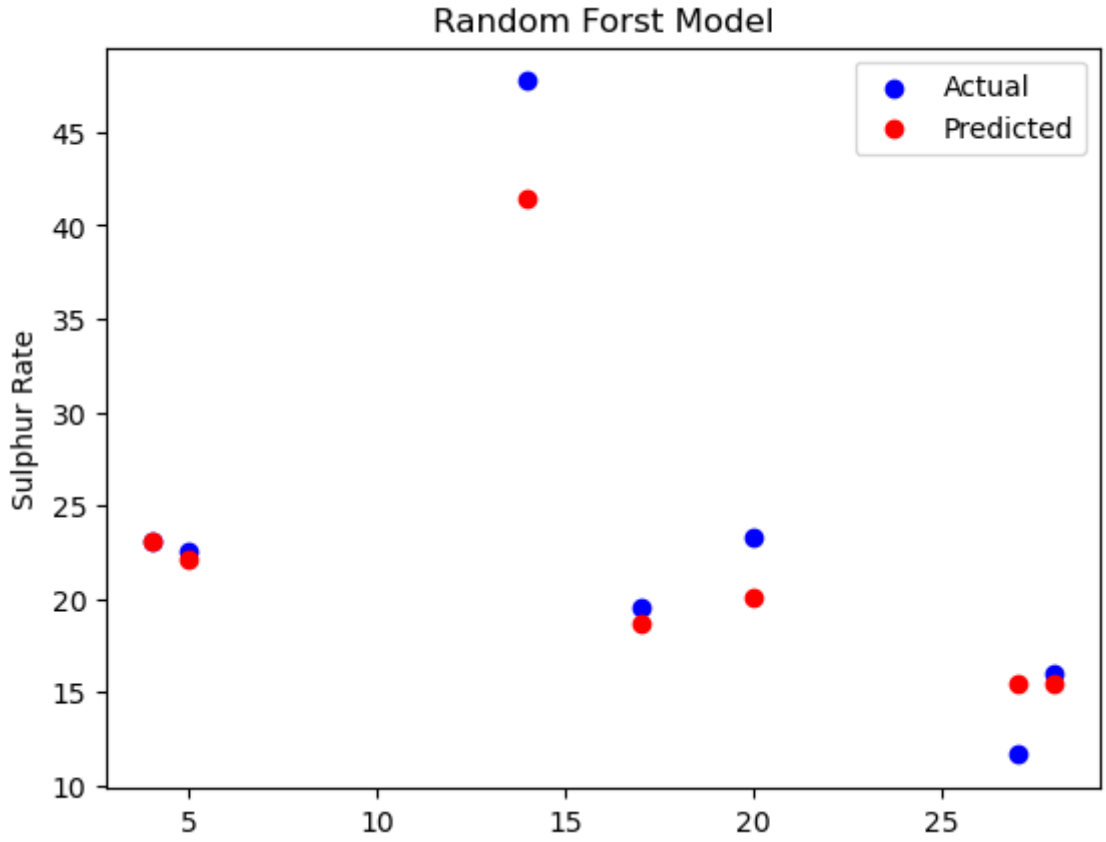
In [130]:

```
Prediction_rf = random_forest_model.predict(x_test)
print(Prediction_rf)
print(y_test)
```

```
[20.10666667 18.63       22.06       41.39666667 15.46       15.46
 23.09      ]
20    23.30
17    19.50
5     22.50
14    47.73
28    16.00
27    11.64
4     23.09
Name: Sulphur Rate, dtype: float64
```

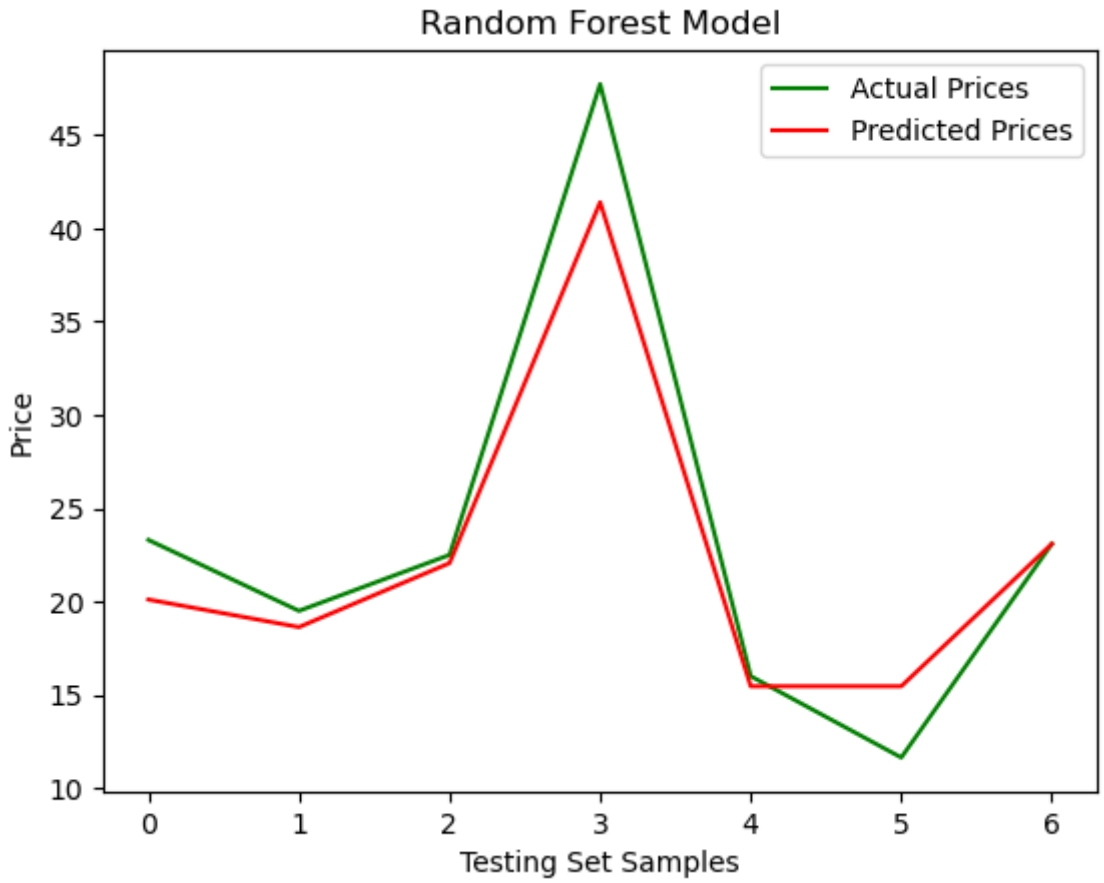
In [151]:

```
# Plot actual test data
plt.scatter(y_test.index,y_test, label='Actual', color='blue')

# Plot predicted values
plt.scatter(y_test.index,Prediction_rf, label='Predicted', color='red')

# Set plot labels and title

plt.ylabel('Sulphur Rate')
plt.title('Random Forst Model')

# Display legend
plt.legend()

# Show plot
plt.show()
```



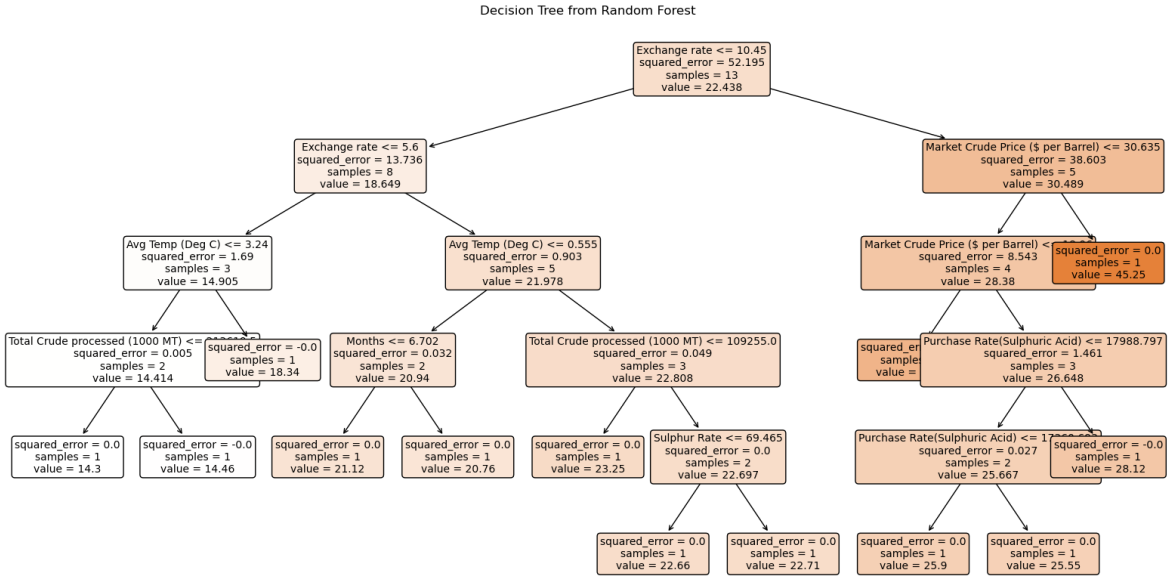
In [150]:

```
xx = range(len(y_test))
# Plot actual values
plt.plot(xx, y_test, color='green', label='Actual Prices')

# Plot predicted values
plt.plot(xx, Prediction_rf, color='red', label='Predicted Prices')

# Add labels and title
plt.xlabel('Testing Set Samples')
plt.ylabel('Price')
plt.title('Random Forest Model')
plt.legend()

# Show the plot
plt.show()
```



In [133]:

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Assuming regressor is your trained Random Forest model
# Pick one tree from the forest, e.g., the first tree (index 0)
tree_to_plot = random_forest_model.estimators_[1]

# Plot the decision tree
plt.figure(figsize=(20, 10))
plot_tree(tree_to_plot, feature_names=df1.columns.tolist(), filled=True, rounded=True, font
plt.title("Decision Tree from Random Forest")
plt.show()
```



AdaBoost

In [134]:

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor


# Initialize AdaBoost regressor with decision tree as base estimator
base_estimator = DecisionTreeRegressor(max_depth=1)  # Weak learner
adaboost_model = AdaBoostRegressor(base_estimator=base_estimator, n_estimators=50, random_s

# Fit the AdaBoost model
adaboost_model.fit(x_train, y_train)
```

```
C:\Users\Agrocel\anaconda3\Lib\site-packages\sklearn\ensemble\_base.py:156:
FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 an
d will be removed in 1.4.
  warnings.warn(
```

Out[134]:

```
AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=1),
                  random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [135]:

```
mse = mean_squared_error(y_test, adaboost_model.predict(x_test))  # Calculate mean squared
print ("Mean Square Error:",mse)

test_score = r2_score(y_test, adaboost_model.predict(x_test)) * 100
train_score = r2_score(y_train, adaboost_model.predict(x_train)) * 100

print("Test_Score:", test_score)
print("Train_Score:", train_score )
```

```
Mean Square Error: 22.47788454105906
Test_Score: 80.35965123880376
Train_Score: 80.52728208237994
```

In [136]:

```
# Now, let's add a new row to the DataFrame
new_row = ['AdaBoost', train_score, test_score]

# Create a DataFrame for the new row
new_row_df = pd.DataFrame([new_row], columns=results_df.columns)

# Append the new row to the existing DataFrame
results_df = pd.concat([results_df, new_row_df], ignore_index=True)
```

In [137]:

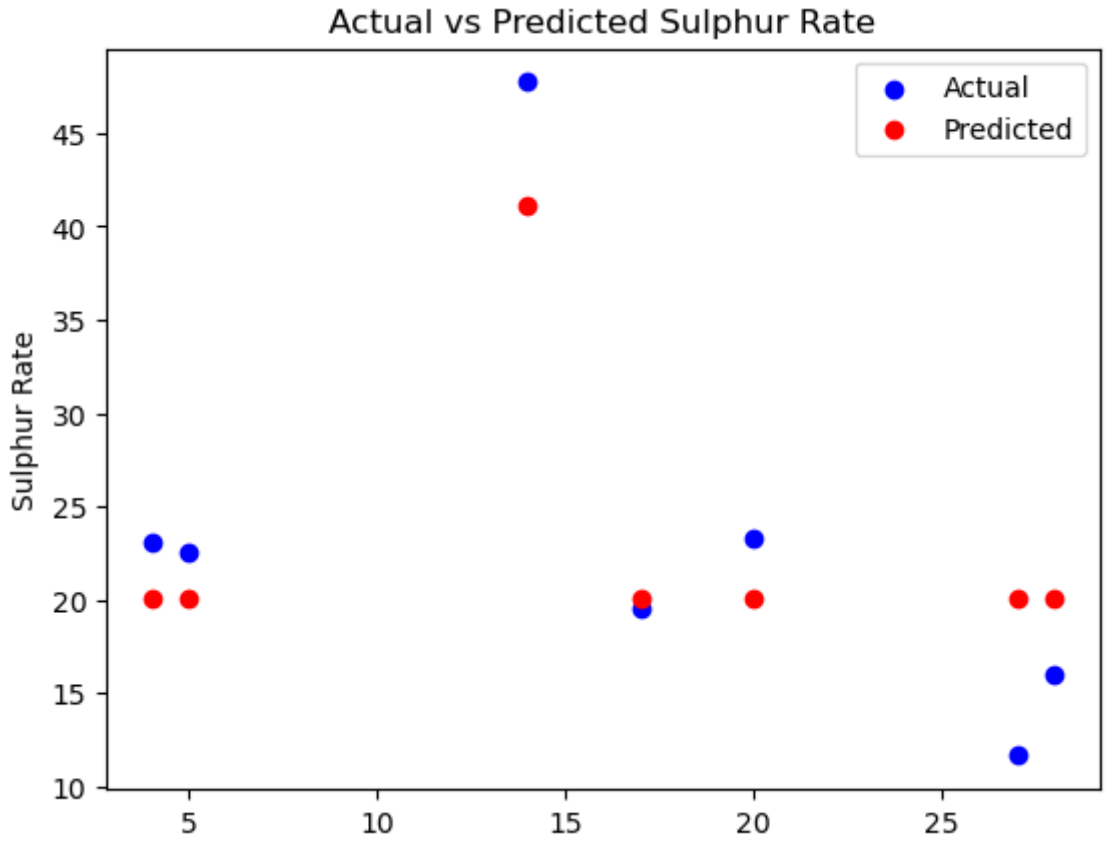
```
#For example, if your new data has features 'feature1' and 'feature2', you need to create a
x_validation = pd.DataFrame([[12.2023,85,26,0.2,83,5.4,18593,21668,219096,95300]])

# 2. Use the predict() method of your fitted linear regression model
predictions = adaboost_model.predict(x_validation)

# Now 'predictions' contains the predicted values for the new data
print(predictions)
```

```
[20.06529412]

C:\Users\Agrocel\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but AdaBoostRegressor was fitted wit
h feature names
  warnings.warn(
```

In [138]:

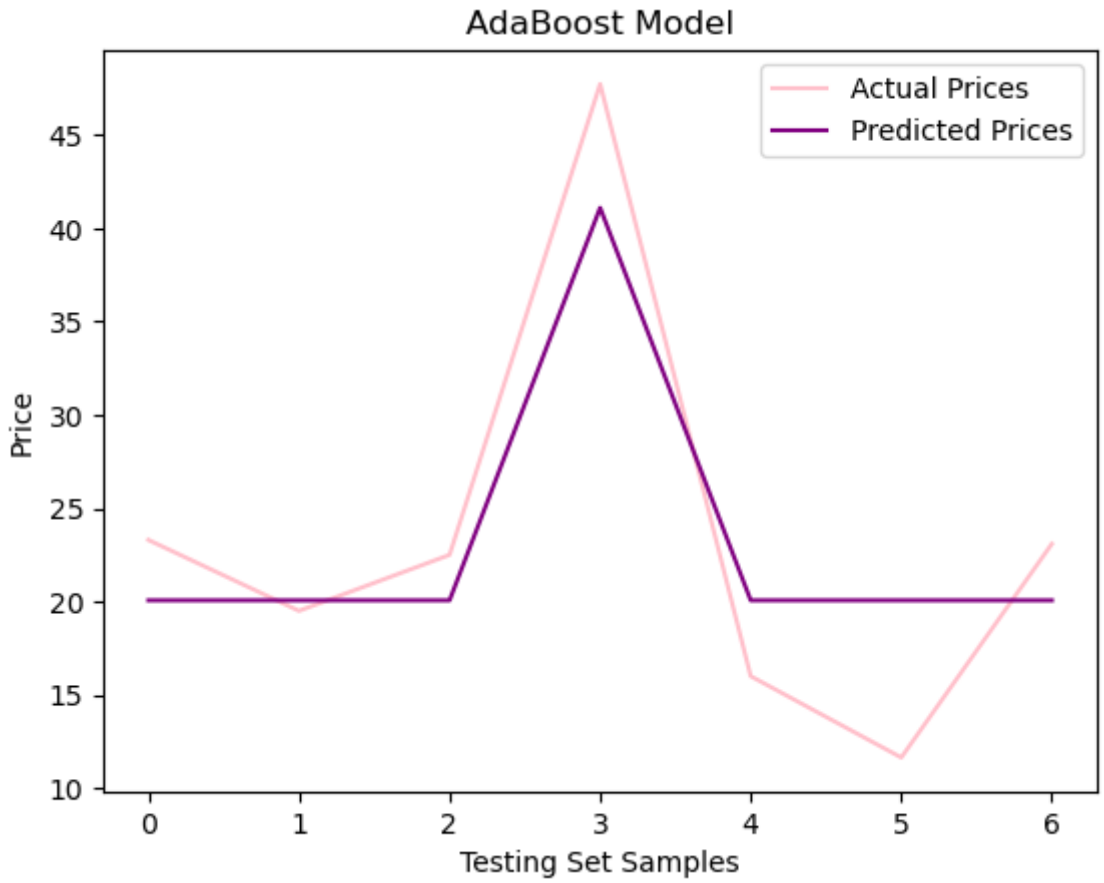
```
Prediction_ad = adaboost_model.predict(x_test)
print(Prediction_ad)
print(y_test)


# Plot actual test data
plt.scatter(y_test.index,y_test, label='Actual', color='blue')

# Plot predicted values
plt.scatter(y_test.index,Prediction_ad, label='Predicted', color='red')

# Set plot labels and title

plt.ylabel('Sulphur Rate')
plt.title('Actual vs Predicted Sulphur Rate')

# Display legend
plt.legend()

# Show plot
plt.show()
```

```
[20.06529412 20.06529412 20.06529412 41.09875    20.06529412 20.06529412
 20.06529412]
20    23.30
17    19.50
5     22.50
14    47.73
28    16.00
27    11.64
4     23.09
Name: Sulphur Rate, dtype: float64
```



In [160]:

```
xx = range(len(y_test))
# Plot actual values
plt.plot(xx, y_test, color='pink', label='Actual Prices')

# Plot predicted values
plt.plot(xx, Prediction_ad, color='purple', label='Predicted Prices')

# Add labels and title
plt.xlabel('Testing Set Samples')
plt.ylabel('Price')
plt.title('AdaBoost Model')
plt.legend()

# Show the plot
plt.show()
```



XGBoost Regression

In [140]:

```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\agrocel\anaconda3\lib\sit
e-packages (2.0.3)
Requirement already satisfied: numpy in c:\users\agrocel\anaconda3\lib\site-
packages (from xgboost) (1.24.3)
Requirement already satisfied: scipy in c:\users\agrocel\anaconda3\lib\site-
packages (from xgboost) (1.11.1)
```

In [141]:

```
from xgboost import XGBRegressor


# Initialize XGBoost regressor
xgb_model = XGBRegressor(learning_rate=0.2)

# Fit the XGBoost model
xgb_model.fit(x_train, y_train)
```

Out[141]:

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.2, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [142]:

```
test_score = r2_score(y_test, xgb_model.predict(x_test)) * 100
train_score = r2_score(y_train, xgb_model.predict(x_train)) * 100

print("Test_Score:", test_score)
print("Train_Score:", train_score )
```

```
Test_Score: 94.76751538150413
Train_Score: 99.99999969078166
```

In [143]:

```
# Now, let's add a new row to the DataFrame
new_row = ['XGBoost', train_score, test_score]

# Create a DataFrame for the new row
new_row_df = pd.DataFrame([new_row], columns=results_df.columns)

# Append the new row to the existing DataFrame
results_df = pd.concat([results_df, new_row_df], ignore_index=True)
```

In [144]:

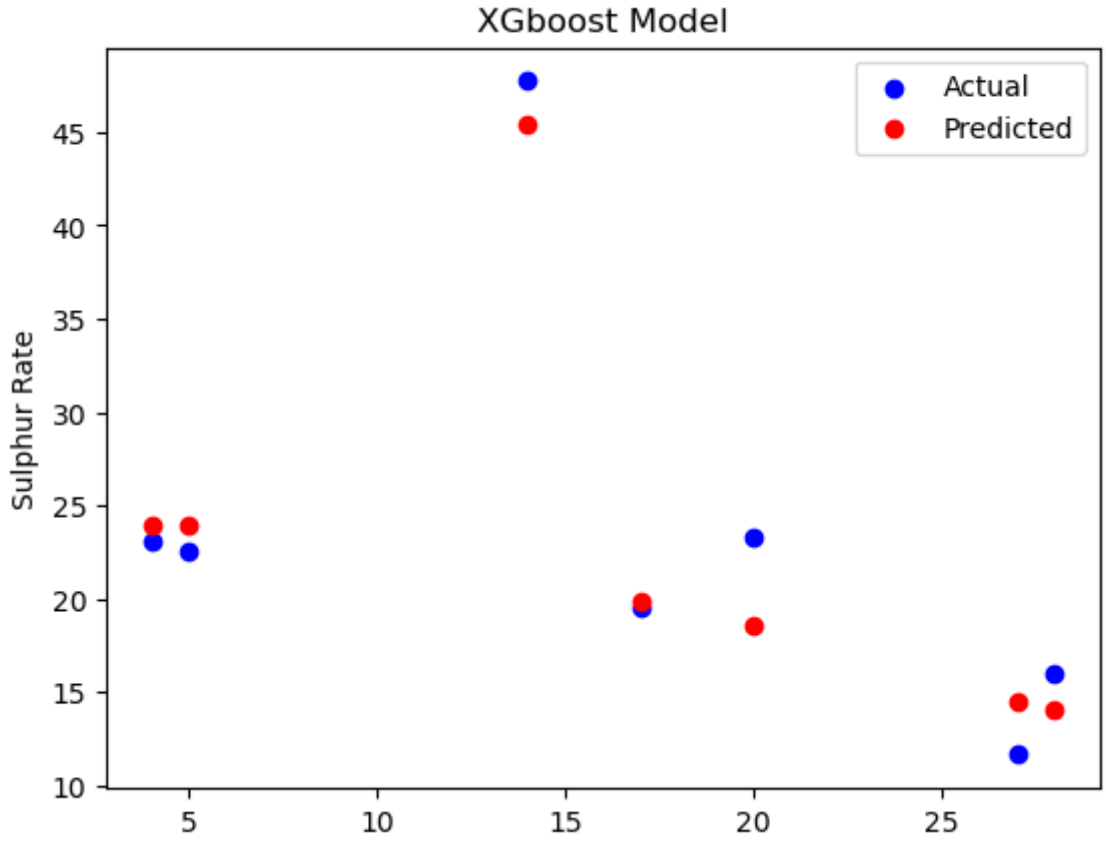
```
#For example, if your new data has features 'feature1' and 'feature2', you need to create a

x_validation.columns = ['Months', 'Market Crude Price ($ per Barrel)', 'Avg Temp (Deg C)',
x_validation = pd.DataFrame([[12.2023,85,26,0.2,83,5.4,18593,21668,219096,95300]])

# 2. Use the predict() method of your fitted linear regression model
predictions = xgb_model.predict(x_validation, validate_features=False)

# Now 'predictions' contains the predicted values for the new data
print(predictions)
```

```
[14.065116]
```

In [149]:

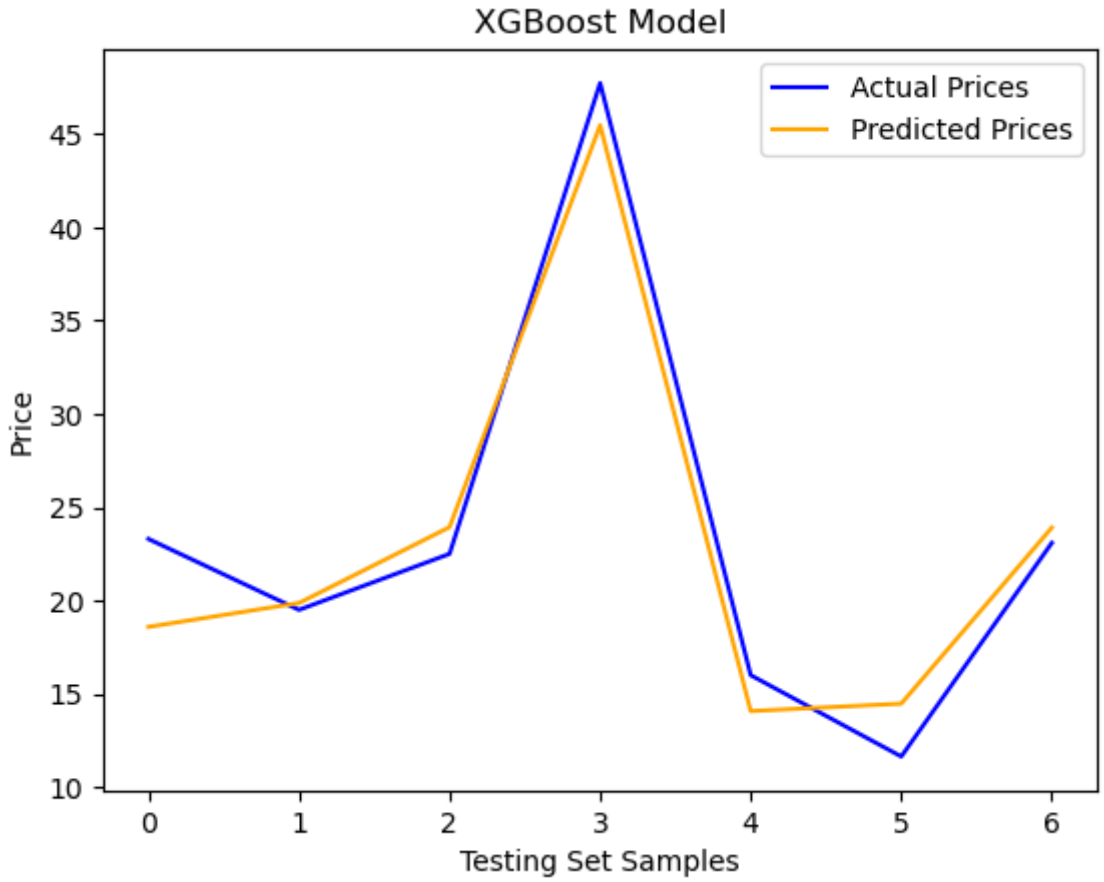
```
Prediction_xg = xgb_model.predict(x_test)
print(Prediction_xg)
print(y_test)


# Plot actual test data
plt.scatter(y_test.index,y_test, label='Actual', color='blue')

# Plot predicted values
plt.scatter(y_test.index,Prediction_xg, label='Predicted', color='red')

# Set plot labels and title

plt.ylabel('Sulphur Rate')
plt.title('XGboost Model')

# Display legend
plt.legend()

# Show plot
plt.show()
```

```
[18.586782 19.865921 23.933828 45.469273 14.074727 14.47335  23.908237]
20    23.30
17    19.50
5     22.50
14    47.73
28    16.00
27    11.64
4     23.09
Name: Sulphur Rate, dtype: float64
```



In [153]:

```
xx = range(len(y_test))
# Plot actual values
plt.plot(xx, y_test, color='blue', label='Actual Prices')

# Plot predicted values
plt.plot(xx, Prediction_xg, color='orange', label='Predicted Prices')

# Add labels and title
plt.xlabel('Testing Set Samples')
plt.ylabel('Price')
plt.title('XGBoost Model')
plt.legend()

# Show the plot
plt.show()
```



Type *Markdown* and LaTeX: α^2

In [147]:

```
# Display the updated DataFrame
print(results_df)
```

```
                    Model  Training Accuracy %  Testing Accuracy %
0         Linear Regression            84.243135           76.813938
1     Polynomial Regression           100.000000         -826.438111
2  Random Forest Regression            90.526323           91.743813
3                  AdaBoost            80.527282           80.359651
4                   XGBoost           100.000000           94.767515
```

In [ ]:

```

```

In [ ]:

```

```