Problem Statement: The aim of the project is to determine the petrol pricing estimation for the given date provided by the user. The code is written in Python programming language with two methods – Decision Tree Regressor and Linear Regression.

Source Code:

Using Decision Tree Regression:

Importing the Packages

```
import numpy as np
import pandas as pd
```

Loading the Dataset

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

```
df
```

	Date	Price
0	1997-01-07	3.82
1	1997-01-08	3.80
2	1997-01-09	3.61
3	1997-01-10	3.92
4	1997-01-13	4.00
...	...	...
5933	2020-08-05	2.23
5934	2020-08-06	2.26
5935	2020-08-07	2.15
5936	2020-08-10	2.18
5937	2020-08-11	2.19

5938 rows × 2 columns

```
df.head()
```

	Date	Price
0	1997-01-07	3.82
1	1997-01-08	3.80
2	1997-01-09	3.61
3	1997-01-10	3.92
4	1997-01-13	4.00

```
df.describe()
```

	Price

count	5937.00000
mean	4.18923
std	2.19121
min	1.05000
25%	2.66000
50%	3.54000
75%	5.24000
max	18.48000

```
df.shape
```

```
(5938, 2)
```

```
df.dtypes
```

```
Date      object
Price    float64
dtype: object
```

Handling the missing values

```
df.isnull().sum()
```

```
Date     0
Price    1
dtype: int64
```

```
df['Price'].fillna(df['Price'].mean(),inplace=True)
```

```
df.isnull().sum()
```

```
Date     0
Price    0
dtype: int64
```

Converting the Date format to the int

```
import datetime
```

```
df['Date']=pd.to_datetime(df.Date,format='%Y-%m-%d')
df.dtypes
```

```
Date     datetime64[ns]
Price           float64
dtype: object
```

```
df['year']=df['Date'].dt.year
df['month']=df['Date'].dt.month
df['day']=df['Date'].dt.day
```

```
df['date']=(df['year']*100+df['month'])*100+df['day']
```

```
df['date'].head()
```

```
     19970107
0    19970108
1    19970109
2    19970110
3    19970113
4
Name: date, dtype: int64
```

Splitting the model to train and test

```
from sklearn.model_selection import train_test_split
```

```
X=df['date']
y=df['Price']
```

```
X.head()
```

```
     19970107
0    19970108
1    19970109
2    19970110
3    19970113
4
Name: date, dtype: int64
```

```
y.head()
```

```
0    3.82
1    3.80
2    3.61
3    3.92
4    4.00
Name: Price, dtype: float64
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=24)
```

Decision Tree Model

```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(np.array(X_train).reshape(-1,1),y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

Predicting the values

```
pred=regressor.predict(np.array(X_test).reshape(-1,1))
```

```
pred
```

```
array([2.335, 5.33 , 2.29 , ..., 2.11 , 2.05 , 6.81 ])
```

```
y_test = np.array(y_test)
```

```
y_test
```

```
array([2.37, 5.45, 2.27, ..., 2.39, 2.07, 7.38])
```

Mean_Squared_Error and r2_score

```
from sklearn.metrics import mean_squared_error,r2_score
import math
print(math.sqrt(mean_squared_error(y_test,pred)))
print(r2_score(y_test,pred))
```

```
0.3628421620628654
0.9758878397136411
```

```
print(mean_squared_error(y_test,pred))
```

```
0.1316544345704547
```

Convering to pickle file

```
import joblib
```

```
joblib.dump(regressor, 'Decision Tree.pkl')
```

```
['Decision Tree.pkl']
```

Using Linear Regression:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import pandas as pd
import numpy as np
```

```
data=pd.read_csv("daily_csv.csv")
```

```
data.head(10)
```

	Date	Price
0	1997-01-07	3.82
1	1997-01-08	3.80
2	1997-01-09	3.61
3	1997-01-10	3.92
4	1997-01-13	4.00
5	1997-01-14	4.01
6	1997-01-15	4.34
7	1997-01-16	4.71
8	1997-01-17	3.91
9	1997-01-20	3.26

```
data.shape
```

```
(5938, 2)
```

```
data.dtypes
```

```
Date      object
Price    float64
dtype: object
```

```
data.min()
```

```
Date     1997-01-07
Price          1.05
dtype: object
```

PreProcessing

```
data.isnull().any()
```

```
Date     False
Price     True
dtype: bool
```

```
mean_Price=data['Price'].mean()
```

```
data['Price'].fillna(value=mean_Price,inplace=True)
```

```
data.isnull().any()
```

```
Date     False
Price    False
dtype: bool
```

```
data[data.duplicated()].any()
```

```
Date     False
Price    False
dtype: bool
```

Datetime Preprocessing

```
from datetime import datetime
```

```
data['Date']=pd.to_datetime(data.Date,format='%Y-%m-%d')
data.dtypes
```

```
Date     datetime64[ns]
Price           float64
dtype: object
```

```
data['year']=data['Date'].dt.year
data['month']=data['Date'].dt.month
data['day']=data['Date'].dt.day
```

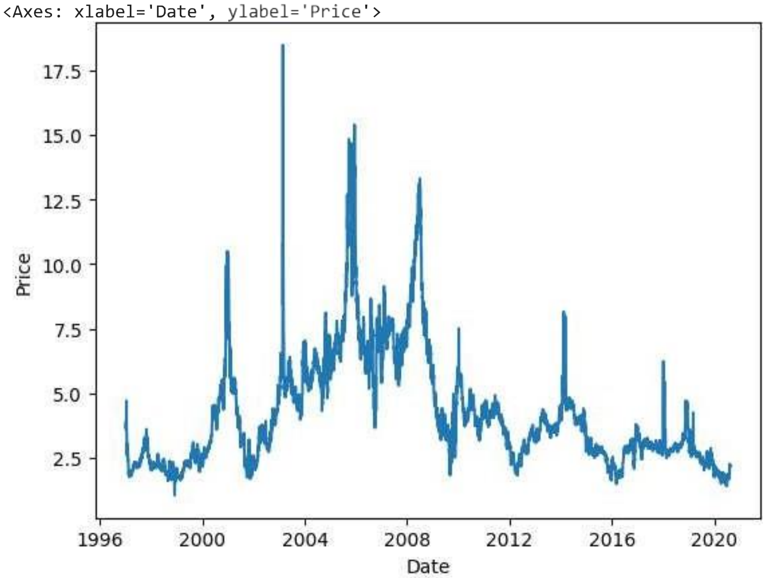
```
data['date']=(data['year']*100+data['month'])*100+data['day']
```

Visualizations

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
sns.lineplot(x='Date',y='Price',data=data)
```

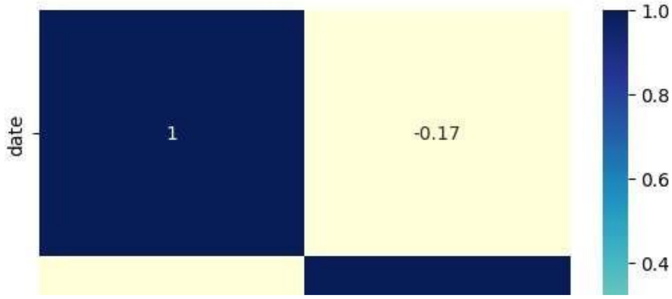
```
<Axes: xlabel='Date', ylabel='Price'>
```




```
plot_data=data[["date","Price"]]
```

```
sns.heatmap(plot_data.corr(),cmap='YlGnBu',annot=True)
```

```
<Axes: >
```



Splitting

```
from sklearn.model_selection import train_test_split
```

```
X=data['date']
y=data['Price']
X_std=data['date']
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,random_state=24)
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(5344,)
(594,)
```

```
from sklearn.preprocessing import StandardScaler
```

```
std=StandardScaler()
X_train_std=std.fit_transform(np.array(X_train).reshape(-1,1))
X_test_std=std.fit_transform(np.array(X_test).reshape(-1,1))
```

Model Linear

```
from sklearn.linear_model import LinearRegression
model_linear=LinearRegression()
```

```
model_linear.fit(np.array(X_train).reshape(-1,1),y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
preds_linear=model_linear.predict(np.array(X_test).reshape(-1,1))
```

```
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
import math
print("RMSE:",math.sqrt(mean_squared_error(y_test,preds_linear)))
print("R-squared:",r2_score(y_test,preds_linear))
print("MAE: ",mean_absolute_error(y_test,preds_linear))
```

```
RMSE: 2.4515378893368927
R-squared: 0.030521956108282455
MAE:  1.72666287053704
```

Model Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
```

```
model_forest=RandomForestRegressor(max_depth=2,random_state=100)
```

```
model_forest.fit(np.array(X_train).reshape(-1,1),y_train)
```

```
▾            RandomForestRegressor
RandomForestRegressor(max_depth=2, random_state=100)
```

```
preds_forest=model_forest.predict(np.array(X_test).reshape(-1,1))
```

```
print("RMSE:",math.sqrt(mean_squared_error(y_test,preds_forest)))
print("R-squared:",r2_score(y_test,preds_forest))
print("MAE: ",mean_absolute_error(y_test,preds_forest))
```

```
RMSE: 1.8360794523257309
R-squared: 0.4561943704639124
MAE:  1.1605551093847126
```

HyperParameter Tuning

```
from sklearn.model_selection import GridSearchCV
```

```
parameters={
    'n_estimators':[100,200,250,300],
    'max_depth':[2,6,8]
}
clf = GridSearchCV(RandomForestRegressor(), param_grid=parameters,verbose=2)

clf.fit(np.array(X_train).reshape(-1,1),y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV] END .....................max_depth=2, n_estimators=100; total time=   0.2s
[CV] END .....................max_depth=2, n_estimators=100; total time=   0.2s
[CV] END .....................max_depth=2, n_estimators=100; total time=   0.2s
[CV] END .....................max_depth=2, n_estimators=100; total time=   0.2s
[CV] END .....................max_depth=2, n_estimators=100; total time=   0.2s
[CV] END .....................max_depth=2, n_estimators=200; total time=   0.4s
[CV] END .....................max_depth=2, n_estimators=200; total time=   0.6s
[CV] END .....................max_depth=2, n_estimators=200; total time=   0.7s
[CV] END .....................max_depth=2, n_estimators=200; total time=   0.7s
[CV] END .....................max_depth=2, n_estimators=200; total time=   0.7s
[CV] END .....................max_depth=2, n_estimators=250; total time=   1.9s
[CV] END .....................max_depth=2, n_estimators=250; total time=   0.7s
[CV] END .....................max_depth=2, n_estimators=250; total time=   0.5s
[CV] END .....................max_depth=2, n_estimators=250; total time=   0.5s
[CV] END .....................max_depth=2, n_estimators=250; total time=   0.6s
[CV] END .....................max_depth=2, n_estimators=300; total time=   0.6s
[CV] END .....................max_depth=2, n_estimators=300; total time=   0.6s
[CV] END .....................max_depth=2, n_estimators=300; total time=   0.6s
[CV] END .....................max_depth=2, n_estimators=300; total time=   0.6s
[CV] END .....................max_depth=2, n_estimators=300; total time=   0.7s
[CV] END .....................max_depth=6, n_estimators=100; total time=   0.3s
[CV] END .....................max_depth=6, n_estimators=100; total time=   0.3s
[CV] END .....................max_depth=6, n_estimators=100; total time=   0.3s
[CV] END .....................max_depth=6, n_estimators=100; total time=   0.3s
```

```
clf.best_score_
```

```
0.9674210642034449
[CV] END .....................max_depth=6, n_estimators=200; total time=   0.7s
```

```
clf.best_params_
```

```
{'max_depth': 8, 'n_estimators': 200}
[CV] END .....................max_depth=6, n_estimators=250; total time=   0.8s
```

```
ideal_model=RandomForestRegressor(max_depth=8,n_estimators=300)
```

```
[CV] END .....................max_depth=6, n_estimators=300; total time=   1.0s
```

```
ideal_model.fit(np.array(X_train).reshape(-1,1),y_train)
```

```
▾            RandomForestRegressor
RandomForestRegressor(max_depth=8, n_estimators=300)
[CV] END .....................max_depth=8, n_estimators=100; total time=   0.4s
```

```
preds_ideal=ideal_model.predict(np.array(X_test).reshape(-1,1))
```

```
[CV] END .....................max_depth=8, n_estimators=200; total time=   3.0s
```

```
print("RMSE:",math.sqrt(mean_squared_error(y_test,preds_ideal)))
print("R-squared:",r2_score(y_test,preds_ideal))
```

```
RMSE: 0.5761278134236906
R-squared: 0.9464575250467651
[CV] END .....................max_depth=8, n_estimators=250; total time=   1.0s
```

```
ideal_model.predict(np.array(20201230).reshape(-1,1))
```

```
array([2.17331905])
[CV] END .....................max_depth=8, n_estimators=300; total time=   1.8s
```

STANDARDIZATION

```
ideal_model=RandomForestRegressor(max_depth=8,n_estimators=100)
```

```
from sklearn.model_selection import GridSearchCV

parameters={
    'n_estimators':[100,200,250,300],
    'max_depth':[2,6,8]
}
clf = GridSearchCV(RandomForestRegressor(), param_grid=parameters,verbose=2)

clf.fit(X_train_std,y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV] END .....................max_depth=2, n_estimators=100; total time=   0.2s
[CV] END .....................max_depth=2, n_estimators=100; total time=   0.2s
[CV] END .....................max_depth=2, n_estimators=100; total time=   0.2s
[CV] END .....................max_depth=2, n_estimators=100; total time=   0.2s
[CV] END .....................max_depth=2, n_estimators=100; total time=   0.2s
[CV] END .....................max_depth=2, n_estimators=200; total time=   0.4s
[CV] END .....................max_depth=2, n_estimators=200; total time=   0.4s
[CV] END .....................max_depth=2, n_estimators=200; total time=   0.5s
[CV] END .....................max_depth=2, n_estimators=200; total time=   0.4s
[CV] END .....................max_depth=2, n_estimators=200; total time=   0.4s
[CV] END .....................max_depth=2, n_estimators=250; total time=   0.6s
[CV] END .....................max_depth=2, n_estimators=250; total time=   0.5s
[CV] END .....................max_depth=2, n_estimators=250; total time=   0.6s
[CV] END .....................max_depth=2, n_estimators=250; total time=   0.5s
[CV] END .....................max_depth=2, n_estimators=250; total time=   0.7s
[CV] END .....................max_depth=2, n_estimators=300; total time=   1.1s
[CV] END .....................max_depth=2, n_estimators=300; total time=   1.1s
[CV] END .....................max_depth=2, n_estimators=300; total time=   1.1s
[CV] END .....................max_depth=2, n_estimators=300; total time=   0.9s
[CV] END .....................max_depth=2, n_estimators=300; total time=   0.7s
[CV] END .....................max_depth=6, n_estimators=100; total time=   0.3s
[CV] END .....................max_depth=6, n_estimators=100; total time=   0.4s
[CV] END .....................max_depth=6, n_estimators=100; total time=   0.3s
[CV] END .....................max_depth=6, n_estimators=100; total time=   0.3s
[CV] END .....................max_depth=6, n_estimators=100; total time=   0.3s
[CV] END .....................max_depth=6, n_estimators=200; total time=   0.7s
[CV] END .....................max_depth=6, n_estimators=200; total time=   0.7s
[CV] END .....................max_depth=6, n_estimators=200; total time=   0.7s
[CV] END .....................max_depth=6, n_estimators=200; total time=   0.7s
[CV] END .....................max_depth=6, n_estimators=200; total time=   0.7s
[CV] END .....................max_depth=6, n_estimators=250; total time=   0.9s
[CV] END .....................max_depth=6, n_estimators=250; total time=   0.9s
[CV] END .....................max_depth=6, n_estimators=250; total time=   0.8s
[CV] END .....................max_depth=6, n_estimators=250; total time=   0.9s
[CV] END .....................max_depth=6, n_estimators=250; total time=   1.1s
[CV] END .....................max_depth=6, n_estimators=300; total time=   1.8s
[CV] END .....................max_depth=6, n_estimators=300; total time=   1.8s
[CV] END .....................max_depth=6, n_estimators=300; total time=   1.2s
[CV] END .....................max_depth=6, n_estimators=300; total time=   1.0s
[CV] END .....................max_depth=6, n_estimators=300; total time=   1.0s
[CV] END .....................max_depth=8, n_estimators=100; total time=   0.4s
[CV] END .....................max_depth=8, n_estimators=100; total time=   0.4s
[CV] END .....................max_depth=8, n_estimators=100; total time=   0.4s
[CV] END .....................max_depth=8, n_estimators=100; total time=   0.4s
[CV] END .....................max_depth=8, n_estimators=100; total time=   0.4s
[CV] END .....................max_depth=8, n_estimators=200; total time=   0.8s
[CV] END .....................max_depth=8, n_estimators=200; total time=   0.8s
[CV] END .....................max_depth=8, n_estimators=200; total time=   0.8s
[CV] END .....................max_depth=8, n_estimators=200; total time=   0.8s
[CV] END .....................max_depth=8, n_estimators=200; total time=   0.8s
[CV] END .....................max_depth=8, n_estimators=250; total time=   1.0s
[CV] END .....................max_depth=8, n_estimators=250; total time=   1.6s
[CV] END .....................max_depth=8, n_estimators=250; total time=   1.7s
[CV] END .....................max_depth=8, n_estimators=250; total time=   1.5s
[CV] END .....................max_depth=8, n_estimators=250; total time=   1.0s
[CV] END .....................max_depth=8, n_estimators=300; total time=   1.2s
[CV] END .....................max_depth=8, n_estimators=300; total time=   1.2s
[CV] END .....................max_depth=8, n_estimators=300; total time=   1.2s
[CV] END .....................max_depth=8, n_estimators=300; total time=   1.2s
[CV] END .....................max_depth=8, n_estimators=300; total time=   1.2s
```

```
▸ GridSearchCV
```

```
clf.best_score_
```

```
0.9675327005894901
```

```
clf.best_params_
```

```
{'max_depth': 8, 'n_estimators': 200}
```

```
ideal_model=RandomForestRegressor(max_depth=8,n_estimators=250)
```

```
ideal_model.fit(X_train_std,y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor(max_depth=8, n_estimators=250)
```

```
preds_ideal=ideal_model.predict(X_test_std)


print("RMSE:",math.sqrt(mean_squared_error(y_test,preds_ideal)))
print("R-squared:",r2_score(y_test,preds_ideal))
```

```
RMSE: 1.5217581259685122
R-squared: 0.6264471817168132
```

Saving Model

```
# from joblib import Parallel, delayed
# import joblib


# # Save the model as a pickle in a file
# joblib.dump(ideal_model, 'Model.pkl')
```

Conclusion: The Decision Tree Regression model is selected over the Linear Regression model due to its slightly increased accuracy in the evaluation metrics. And the Decision tree regressor model is finalized and pinned as a .pkl file as the final model.