Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Exe19 - Decision Tree Exercise 1

Name: Chuay Xiang Ze

IC Number: 021224070255

Date : 20/07/2023

Introduction : Learning how to use DecisionTree

Conclusion : Managed to complete tasks relating to the topic.

Section 1

Reference: https://www.kaggle.com/vinicius150987/bank-full-machine-learning/notebook (https://www.kaggle.com/vinicius150987/bank-full-machine-learning/notebook)

Decision Tree

In [5]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
```

Read "bank-full.csv"

In [6]:

```
data = pd.read_csv("./bank-full.csv", sep=';')
df = pd.DataFrame(data)
df
```

Out[6]:

	age	job	marital	education	default	balance	housing	loan	contact	day
0	58	management	married	tertiary	no	2143	yes	no	unknown	5
1	44	technician	single	secondary	no	29	yes	no	unknown	5
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5
4	33	unknown	single	unknown	no	1	no	no	unknown	5
...	...	...	...	...	...	...	...	...	...	..
45206	51	technician	married	tertiary	no	825	no	no	cellular	17
45207	71	retired	divorced	primary	no	1729	no	no	cellular	17
45208	72	retired	married	secondary	no	5715	no	no	cellular	17
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17

45211 rows × 17 columns

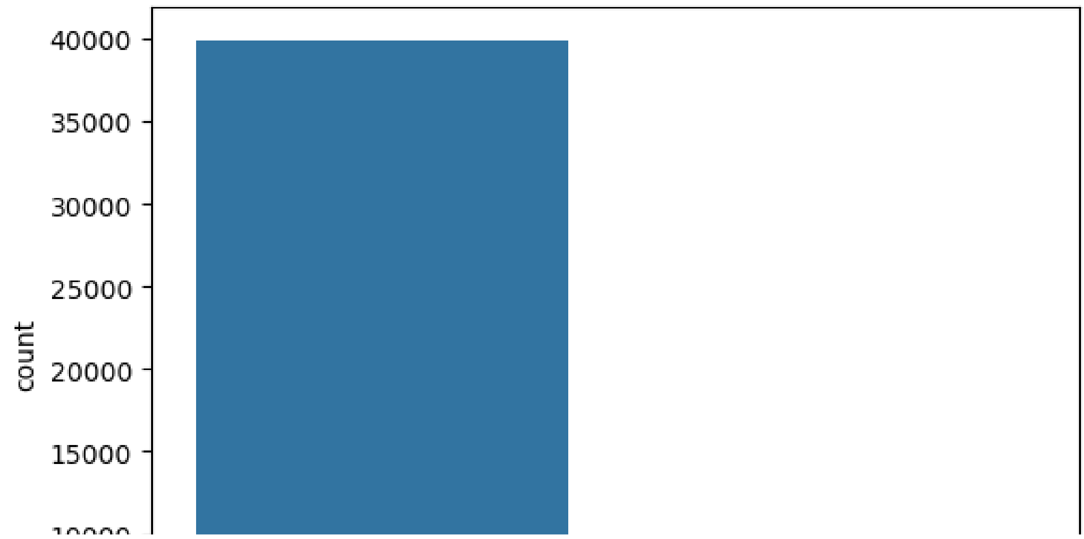
Check the distribution of labels (‘yes’, ‘no’) are distributed.

In [7]:

```
sns.countplot(x=df["y"])
```

Out[7]:

```
<Axes: xlabel='y', ylabel='count'>
```



In [ ]:

```

```

Counts of "yes" and "no" with "age"

In [8]:

```
df.groupby('age')['y'].value_counts()
```

Out[8]:

```
age  y
18   yes     7
     no      5
19   no     24
     yes    11
20   no     35
            ..
92   yes     2
93   yes     2
94   no      1
95   no      1
     yes     1
Name: y, Length: 148, dtype: int64
```

Correlation between the data

In [9]:

```
df.corr(numeric_only= True)
```

Out[9]:

	age	balance	day	duration	campaign	pdays	previous
age	1.000000	0.097783	-0.009120	-0.004648	0.004760	-0.023758	0.001288
balance	0.097783	1.000000	0.004503	0.021560	-0.014578	0.003435	0.016674
day	-0.009120	0.004503	1.000000	-0.030206	0.162490	-0.093044	-0.051710
duration	-0.004648	0.021560	-0.030206	1.000000	-0.084570	-0.001565	0.001203
campaign	0.004760	-0.014578	0.162490	-0.084570	1.000000	-0.088628	-0.032855
pdays	-0.023758	0.003435	-0.093044	-0.001565	-0.088628	1.000000	0.454820
previous	0.001288	0.016674	-0.051710	0.001203	-0.032855	0.454820	1.000000

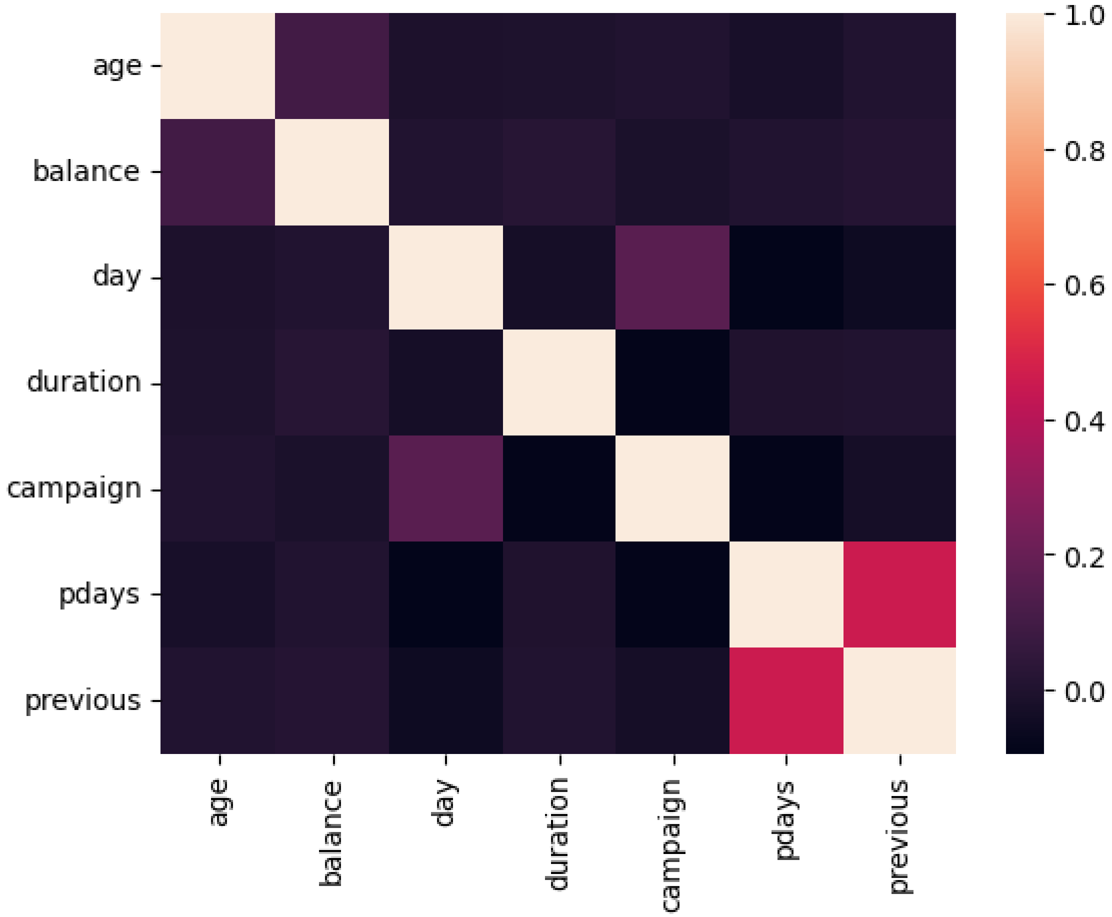
Plot the heatmap

In [10]:

```
sns.heatmap(df.corr(numeric_only = True))
```

Out[10]:

```
<Axes: >
```



Convert categorical data into numerical

In [11]:

```
reset = {"yes": 1, "no": 0, "unknown": None}
df = df.replace({"housing": reset})
df = df.replace({"loan": reset})
df = df.replace({"education": {"primary": 1, "secondary": 2, "tertiary": 3, "unknown": N
df = df.replace({"marital": {"single": 1, "married": 2, "divorced": 3, "unknown": None}}
df = df.replace({"contact": {"cellular": 1, "telephone": 0, "unknown": None}})
df = df.replace({"y": {"yes": 1, "no": 0, "unknown": None}})
df = df.dropna()
df
```

Out[11]:

	age	job	marital	education	default	balance	housing	loan	contact	day
12657	27	management	1	2.0	no	35	0	0	1.0	4
12658	54	blue-collar	2	1.0	no	466	0	0	1.0	4
12659	43	blue-collar	2	2.0	no	105	0	1	1.0	4
12660	31	technician	1	2.0	no	19	0	0	0.0	4
12661	27	technician	1	2.0	no	126	1	1	1.0	4
...	...	...	...	...	...	...	...	...	...	...
45206	51	technician	2	3.0	no	825	0	0	1.0	17
45207	71	retired	3	1.0	no	1729	0	0	1.0	17
45208	72	retired	2	2.0	no	5715	0	0	1.0	17
45209	57	blue-collar	2	2.0	no	668	0	0	0.0	17
45210	37	entrepreneur	2	2.0	no	2971	0	0	1.0	17

31011 rows × 17 columns

Next step is to select features and labels

In [12]:

```

feature_cols = ['marital', 'education', 'balance', 'duration', 'housing', 'loan', 'conta
X = df[feature_cols] # Features
y = df.y # Target variable
```

Drop "poutcome"

In [13]:

```
del df['poutcome']
df
```

Out[13]:

	age	job	marital	education	default	balance	housing	loan	contact	day
12657	27	management	1	2.0	no	35	0	0	1.0	4
12658	54	blue-collar	2	1.0	no	466	0	0	1.0	4
12659	43	blue-collar	2	2.0	no	105	0	1	1.0	4
12660	31	technician	1	2.0	no	19	0	0	0.0	4
12661	27	technician	1	2.0	no	126	1	1	1.0	4
...	...	...	...	...	...	...	...	...	...	...
45206	51	technician	2	3.0	no	825	0	0	1.0	17
45207	71	retired	3	1.0	no	1729	0	0	1.0	17
45208	72	retired	2	2.0	no	5715	0	0	1.0	17
45209	57	blue-collar	2	2.0	no	668	0	0	0.0	17
45210	37	entrepreneur	2	2.0	no	2971	0	0	1.0	17

31011 rows × 16 columns

Split the data into train and test

In [14]:

```
import pandas as pd


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

Applying Decision Tree Classifier:

Next, I created a pipeline of StandardScaler (standardize the features) and DT Classifier (see a note below regarding Standardization of features). We can import DT classifier as from sklearn.tree import DecisionTreeClassifier from Scikit-Learn. To determine the best parameters (criterion of split and maximum tree depth) for DT classifier, I also used Grid Search Cross Validation. The code snippet below is self-explanatory.

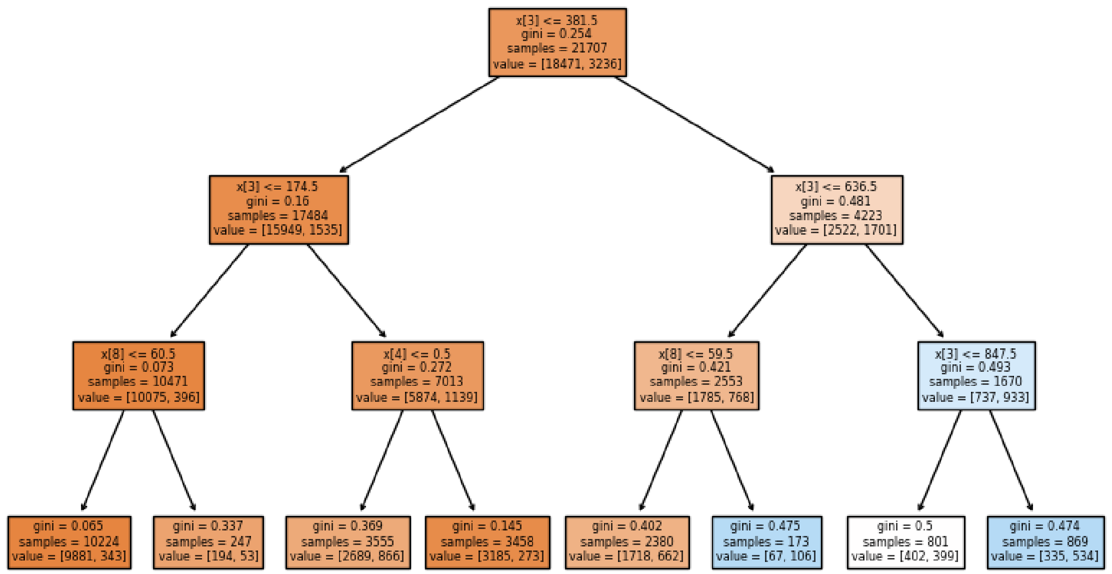
In [15]:

```
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics, tree

# Create Decision Tree classifer object
clf = DecisionTreeClassifier(max_depth = 3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

To display

In [16]:

```
plt.figure(figsize=(10, 6))
tree.plot_tree(clf, filled= True)
plt.show()
```



The number of nodes and the maximum depth

In [17]:

```
print(clf.tree_.node_count, clf.tree_.max_depth)
```

```
15 3
```

Accuracy measurement

In [18]:

```
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.869518486672399
```

Prediction

In [19]:

```
y_pred
```

Out[19]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Grid Search

In [20]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

pipe = Pipeline(steps=[('dec_tree', clf)])
criterion = ['gini', 'entropy']
max_depth = [2,4,6,8,10,12]

parameters = dict(dec_tree__criterion = criterion, dec_tree__max_depth = max_depth)

grid_search = GridSearchCV(pipe, parameters)

grid_search.fit(X_train, y_train)
```

Out[20]:

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ DecisionTreeClassifier
```

Display the best features

In [21]:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=100)
classifier.fit(X_train, y_train)

feature_importances_df = pd.DataFrame(
    {"feature": list(X.columns), "importance": classifier.feature_importances_}
).sort_values("importance", ascending=False)

# Display
feature_importances_df
```

Out[21]:

	feature	importance
3	duration	0.403927
2	balance	0.237821
8	age	0.187885
7	campaign	0.054657
1	education	0.033936
0	marital	0.030489
4	housing	0.023989
5	loan	0.015022
6	contact	0.012273

Run DecisionTreeClassifier using the obtained features

In [22]:

```
optimized = DecisionTreeClassifier(criterion='gini', max_depth=2)
optimized.fit(X_train, y_train)
```

Out[22]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2)
```

Concat train test results

In [31]:

```
y_pred_train = optimized.predict(X_train)
y_pred_test = optimized.predict(X_test)
y_train_np = y_train.to_numpy().reshape(len(y_train), 1)
y_test_np = y_test.to_numpy().reshape(len(y_test), 1)

result_train = np.concatenate((y_pred_train.reshape(-1, 1), y_train_np), axis=1)
result_test = np.concatenate((y_pred_test.reshape(-1, 1), y_test_np), axis=1)

print("Training:")
print(result_train)

print("Test:")
print(result_test)
```

```
Training:
[[0 0]
 [0 0]
 [1 1]
 ...
 [0 0]
 [0 0]
 [0 0]]
Test:
[[0 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

Section 2

1. Read "petrol_consumption.csv" file

In [33]:

```
df = pd.read_csv("./petrol_consumption.csv")
df
```

Out[33]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consu
0	9.00	3571	1976	0.525	
1	9.00	4092	1250	0.572	
2	9.00	3865	1586	0.580	
3	7.50	4870	2351	0.529	
4	8.00	4399	431	0.544	
5	10.00	5342	1333	0.571	
6	8.00	5319	11868	0.451	
7	8.00	5126	2138	0.553	
8	8.00	4447	8577	0.529	
9	7.00	4512	8507	0.552	
10	8.00	4391	5939	0.530	
11	7.50	5126	14186	0.525	
12	7.00	4817	6930	0.574	
13	7.00	4207	6580	0.545	
14	7.00	4332	8159	0.608	
15	7.00	4318	10340	0.586	
16	7.00	4206	8508	0.572	
17	7.00	3718	4725	0.540	
18	7.00	4716	5915	0.724	
19	8.50	4341	6010	0.677	
20	7.00	4593	7834	0.663	
21	8.00	4983	602	0.602	
22	9.00	4897	2449	0.511	
23	9.00	4258	4686	0.517	
24	8.50	4574	2619	0.551	
25	9.00	3721	4746	0.544	
26	8.00	3448	5399	0.548	
27	7.50	3846	9061	0.579	
28	8.00	4188	5975	0.563	
29	9.00	3601	4650	0.493	
30	7.00	3640	6905	0.518	
31	7.00	3333	6594	0.513	
32	8.00	3063	6524	0.578	
33	7.50	3357	4121	0.547	
34	8.00	3528	3495	0.487	
35	6.58	3802	7834	0.629	
36	5.00	4045	17782	0.566	

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consu
37	7.00	3897	6385	0.586	
38	8.50	3635	3274	0.663	
39	7.00	4345	3905	0.672	
40	7.00	4449	4639	0.626	
41	7.00	3656	3985	0.563	
42	7.00	4300	3635	0.603	
43	7.00	3745	2611	0.508	
44	6.00	5215	2302	0.672	
45	9.00	4476	3942	0.571	
46	7.00	4296	4083	0.623	
47	7.00	5002	9794	0.593	

2. Display the first 5 records

In [34]:

```
df.head()
```

Out[34]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consun
0	9.0	3571	1976	0.525	
1	9.0	4092	1250	0.572	
2	9.0	3865	1586	0.580	
3	7.5	4870	2351	0.529	
4	8.0	4399	431	0.544	

4. Identify the label (Petrol_Consumption)

In [35]:

```
y = df.Petrol_Consumption
y
```

Out[35]:

```
0     541
1     524
2     561
3     414
4     410
5     457
6     344
7     467
8     464
9     498
10    580
11    471
12    525
13    508
14    566
15    635
16    603
17    714
18    865
19    640
20    649
21    540
22    464
23    547
24    460
25    566
26    577
27    631
28    574
29    534
30    571
31    554
32    577
33    628
34    487
35    644
36    640
37    704
38    648
39    968
40    587
41    699
42    632
43    591
44    782
45    510
46    610
47    524
Name: Petrol_Consumption, dtype: int64
```

5. Identify the features.

In [36]:

```
X = df.drop('Petrol_Consumption', axis=1)
X
```

Out[36]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)
0	9.00	3571	1976	0.525
1	9.00	4092	1250	0.572
2	9.00	3865	1586	0.580
3	7.50	4870	2351	0.529
4	8.00	4399	431	0.544
5	10.00	5342	1333	0.571
6	8.00	5319	11868	0.451
7	8.00	5126	2138	0.553
8	8.00	4447	8577	0.529
9	7.00	4512	8507	0.552
10	8.00	4391	5939	0.530
11	7.50	5126	14186	0.525
12	7.00	4817	6930	0.574
13	7.00	4207	6580	0.545
14	7.00	4332	8159	0.608
15	7.00	4318	10340	0.586
16	7.00	4206	8508	0.572
17	7.00	3718	4725	0.540
18	7.00	4716	5915	0.724
19	8.50	4341	6010	0.677
20	7.00	4593	7834	0.663
21	8.00	4983	602	0.602
22	9.00	4897	2449	0.511
23	9.00	4258	4686	0.517
24	8.50	4574	2619	0.551
25	9.00	3721	4746	0.544
26	8.00	3448	5399	0.548
27	7.50	3846	9061	0.579
28	8.00	4188	5975	0.563
29	9.00	3601	4650	0.493
30	7.00	3640	6905	0.518
31	7.00	3333	6594	0.513
32	8.00	3063	6524	0.578
33	7.50	3357	4121	0.547
34	8.00	3528	3495	0.487
35	6.58	3802	7834	0.629
36	5.00	4045	17782	0.566

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)
37	7.00	3897	6385	0.586
38	8.50	3635	3274	0.663
39	7.00	4345	3905	0.672
40	7.00	4449	4639	0.626
41	7.00	3656	3985	0.563
42	7.00	4300	3635	0.603
43	7.00	3745	2611	0.508
44	6.00	5215	2302	0.672
45	9.00	4476	3942	0.571
46	7.00	4296	4083	0.623
47	7.00	5002	9794	0.593

6. Use of describe method to describe the dataset.

In [37]:

```
df.describe()
```

Out[37]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Co
count	48.000000	48.000000	48.000000	48.000000	
mean	7.668333	4241.833333	5565.416667	0.570333	
std	0.950770	573.623768	3491.507166	0.055470	
min	5.000000	3063.000000	431.000000	0.451000	
25%	7.000000	3739.000000	3110.250000	0.529750	
50%	7.500000	4298.000000	4735.500000	0.564500	
75%	8.125000	4578.750000	7156.000000	0.595250	
max	10.000000	5342.000000	17782.000000	0.724000	

7. Display the first 5 records of the features

In [38]:

```
X.head()
```

Out[38]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)
0	9.0	3571	1976	0.525
1	9.0	4092	1250	0.572
2	9.0	3865	1586	0.580
3	7.5	4870	2351	0.529
4	8.0	4399	431	0.544

8. Split the data into training (80%) and testing (20%) sets.

In [39]:

```
import pandas as pd

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

9. Build your model and train the training data

In [40]:

```
from sklearn.tree import DecisionTreeRegressor

clf = DecisionTreeRegressor(max_depth = 3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

10. Prediction using the testing set

In [41]:

```
y_pred
```

Out[41]:

```
array([547.875, 547.875, 547.875, 572.3  , 547.875, 458.7  , 572.3  ,
       547.875, 968.   , 547.875])
```

11. Display Actual and Predictied price side by side in df

In [42]:

```
results_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}).reset_index(drop = Tr
results_df
```

Out[42]:

	Actual	Predicted
0	628	547.875
1	547	547.875
2	648	547.875
3	640	572.300
4	561	547.875
5	414	458.700
6	554	572.300
7	577	547.875
8	782	968.000
9	631	547.875

12. Evaluate the model using mean_absulate_error

In [43]:

```
from sklearn.metrics import accuracy_score, mean_absolute_error
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)
```

```
Mean Absolute Error: 62.32000000000001
```

13. Display the predicted output using first 5 features.

In [46]:

```
results_df['Predicted'].head()
```

Out[46]:

```
0    547.875
1    547.875
2    547.875
3    572.300
4    547.875
Name: Predicted, dtype: float64
```

In [ ]:

```

```