Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Exe19 - Decision Tree Exercise 1

Name: Chong Mun Chen

IC Number: 960327-07-5097

Date : 18/7/2023

Introduction : Practising on supervised machine learning with decision tree classification and regression.

Conclusion : Achieved a proper accuracy score with the training and testing sets.

Section 1

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

Decision Tree

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import plotly.express as px
```

Read "bank-full.csv"

In [2]:
```
df = pd.read_csv('../data_samples2/bank-full.csv', delimiter=';')
```

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

Out[3]:

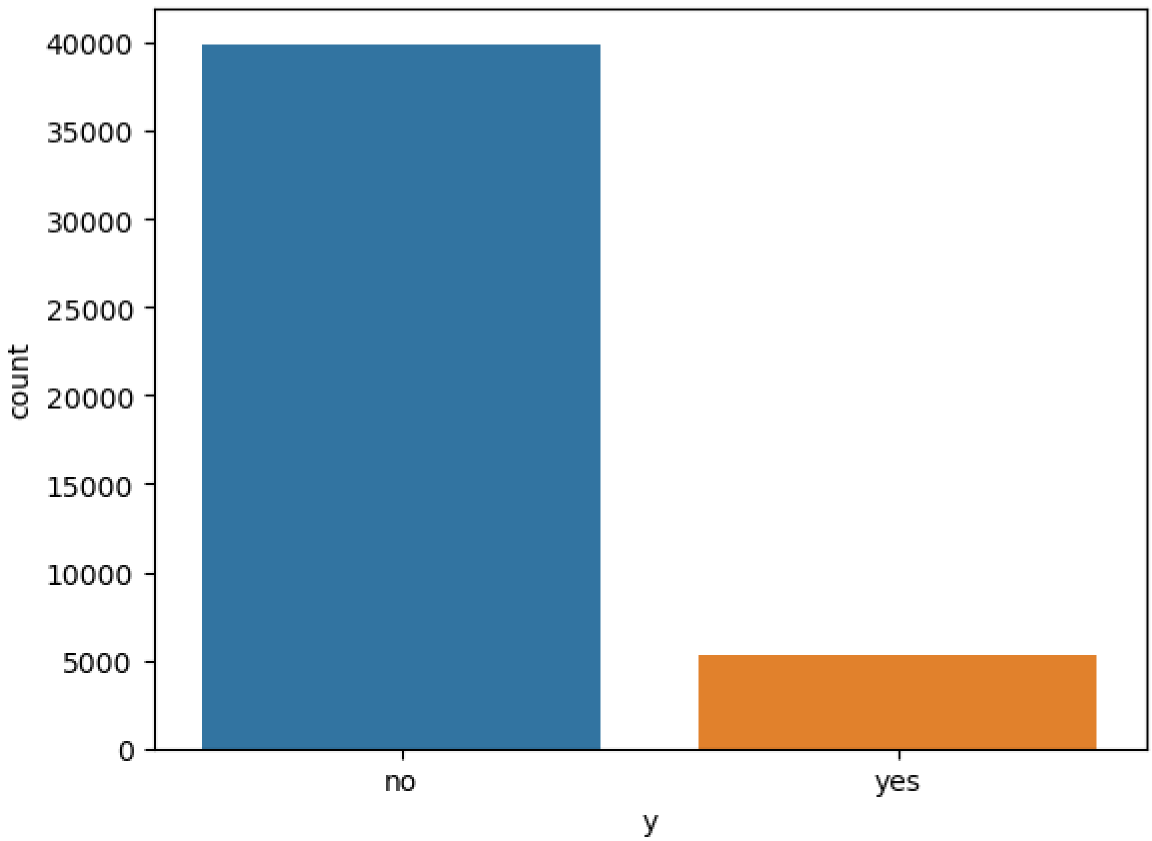
	age	job	marital	education	default	balance	housing	loan	contact	day	month
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may

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

In [4]:
```
sns.countplot(x=df['y'])

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

Out[4]:
```
no     39922
yes     5289
Name: y, dtype: int64
```



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

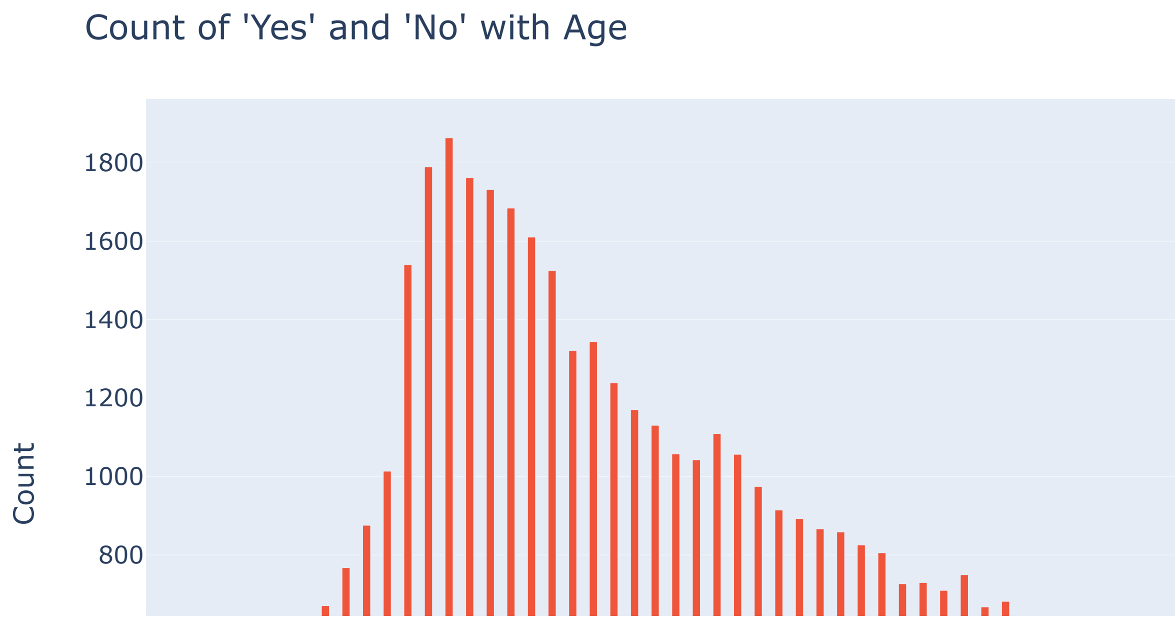
In [5]:
```
age_df = df.groupby('age')['y'].value_counts()
age_df = age_df.rename('count').reset_index()
print(age_df)

# Plot using seaborn barplot
# plt.figure(figsize=(10, 6))
# sns.barplot(data=age_df, x='age', y='count', hue='y')
# plt.xlabel('Age')
# plt.ylabel('Count')
# plt.legend(title='Response')
# plt.title("Count of 'Yes' and 'No' with Age")
# plt.show()


# Plot using plotly barplot
fig = px.bar(age_df, x='age', y='count', color='y', barmode='group')
fig.update_layout(
    title="Count of 'Yes' and 'No' with Age",
    xaxis_title="Age",
    yaxis_title="Count",
    legend_title="Response"
)
fig.show()
```

```
     age    y  count
0     18  yes      7
1     18   no      5
2     19   no     24
3     19  yes     11
4     20   no     35
..   ...  ...    ...
143   92  yes      2
144   93  yes      2
145   94   no      1
146   95   no      1
147   95  yes      1

[148 rows x 3 columns]
```



Correlation between the data

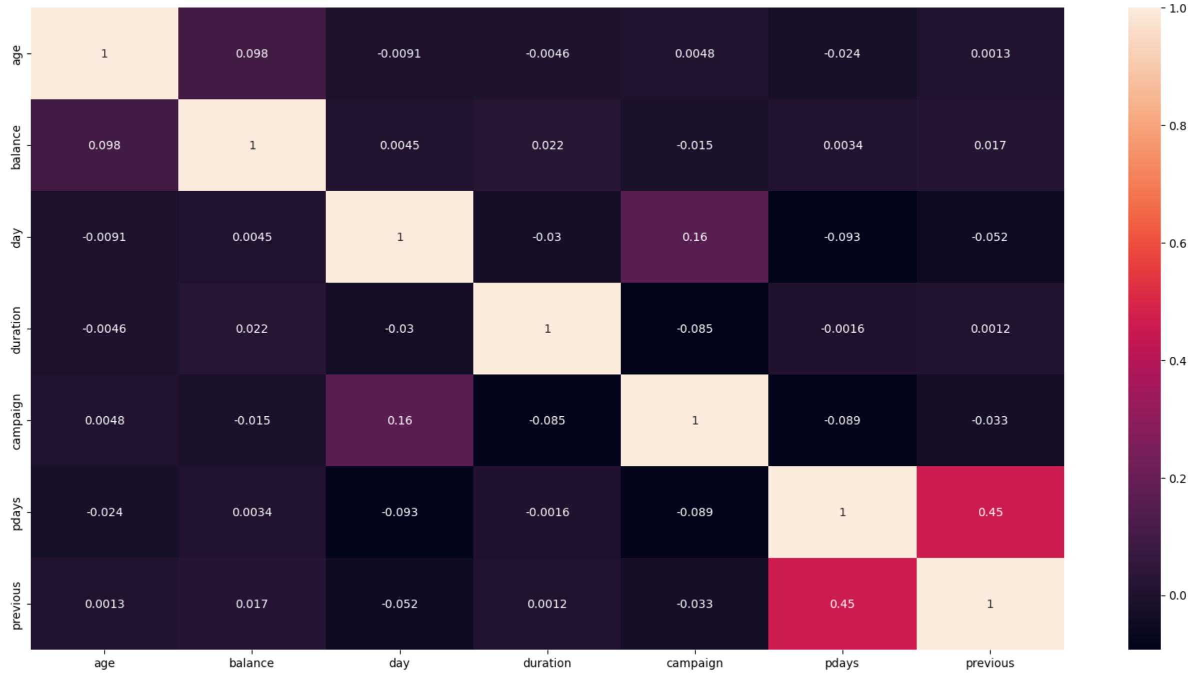
In [6]:
```
df.corr(numeric_only=True)
```

Out[6]:

	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 [7]:
```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(numeric_only=True), annot=True)
plt.show()
```



Convert categorical data into numerical

In [8]:
```
replace_response = {'yes': 1, 'no': 0}
df = df.replace({'default': replace_response, 'housing': replace_response, 'lo
                 'y': replace_response,})

# replace_marital = {'single': 1, 'married': 2, 'divorced': 3}
# df['marital'] = df['marital'].replace(replace_marital)

# replace_education = {'primary': 1, 'secondary': 2, 'tertiary': 3, 'unknown':
# df['education'] = df['education'].replace(replace_education)

# replace_contact = {'telephone': 1, 'cellular': 2, 'unknown': None}
# df['contact'] = df['contact'].replace(replace_contact)

df
```

Out[8]:

	age	job	marital	education	default	balance	housing	loan	contact	day	m
0	58	management	married	tertiary	0	2143	1	0	unknown	5	
1	44	technician	single	secondary	0	29	1	0	unknown	5	
2	33	entrepreneur	married	secondary	0	2	1	1	unknown	5	
3	47	blue-collar	married	unknown	0	1506	1	0	unknown	5	
4	33	unknown	single	unknown	0	1	0	0	unknown	5	
...	...	...	...	...	...	...	...	...	...	...	
45206	51	technician	married	tertiary	0	825	0	0	cellular	17	
45207	71	retired	divorced	primary	0	1729	0	0	cellular	17	
45208	72	retired	married	secondary	0	5715	0	0	cellular	17	
45209	57	blue-collar	married	secondary	0	668	0	0	telephone	17	
45210	37	entrepreneur	married	secondary	0	2971	0	0	cellular	17	

45211 rows × 17 columns

Next step is to select features and labels

In [11]:
```
# da = df.dropna(subset=['education', 'contact'])
# X = da.drop(['job', 'day', 'month', 'poutcome', 'y'], axis=1)
X = df[['default', 'housing', 'loan']]
y = df.y
```

Drop "poutcome"

In [12]:
```
new_df = df.drop('poutcome', axis=1)
new_df
```

Out[12]:

	age	job	marital	education	default	balance	housing	loan	contact	day	m
0	58	management	married	tertiary	0	2143	1	0	unknown	5	
1	44	technician	single	secondary	0	29	1	0	unknown	5	
2	33	entrepreneur	married	secondary	0	2	1	1	unknown	5	
3	47	blue-collar	married	unknown	0	1506	1	0	unknown	5	
4	33	unknown	single	unknown	0	1	0	0	unknown	5	
...	...	...	...	...	...	...	...	...	...	...	
45206	51	technician	married	tertiary	0	825	0	0	cellular	17	
45207	71	retired	divorced	primary	0	1729	0	0	cellular	17	
45208	72	retired	married	secondary	0	5715	0	0	cellular	17	
45209	57	blue-collar	married	secondary	0	668	0	0	telephone	17	
45210	37	entrepreneur	married	secondary	0	2971	0	0	cellular	17	

45211 rows × 16 columns

Split the data into train and test

In [13]:
```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import metrics, tree

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
```

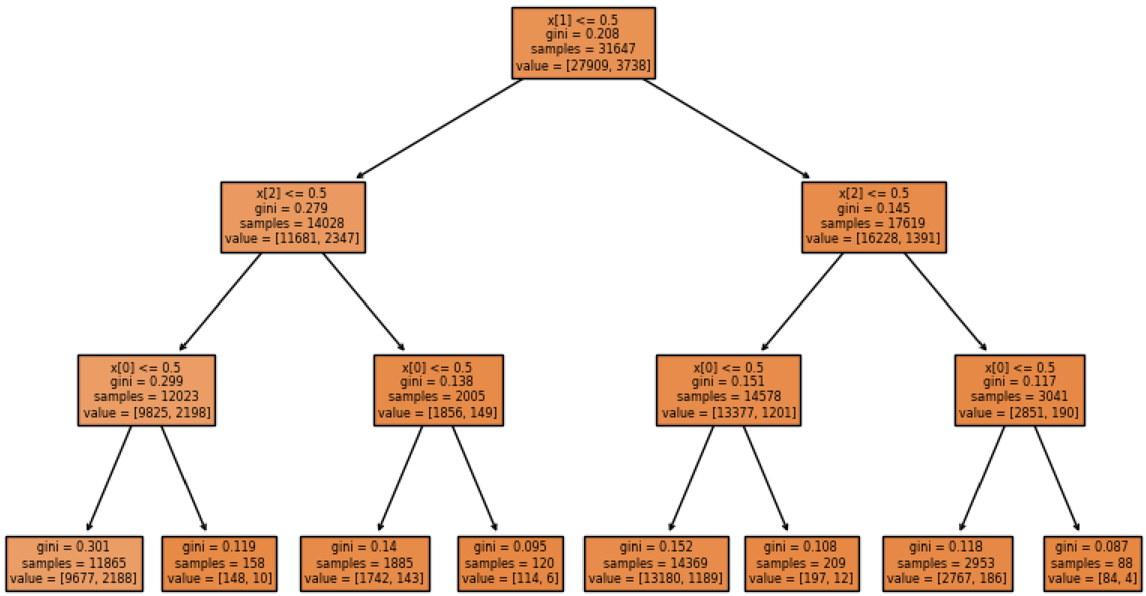
Applying Decision Tree Classifier:

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 [14]:
```
clf = DecisionTreeClassifier()
clf = clf.fit(X_train,y_train)
```

To display

In [15]:
```
plt.figure(figsize=(10, 6))
tree.plot_tree(clf, filled=True)
plt.show()
```



The number of nodes and the maximum depth

In [16]:
```
print(clf.tree_.node_count, clf.tree_.max_depth)
```

```
15 3
```

Prediction

In [17]:
```
y_pred = clf.predict(X_test)
pd.DataFrame({'Predicted':y_pred})
```

Out[17]:

	Predicted
0	0
1	0
2	0
3	0
4	0
...	...
13559	0
13560	0
13561	0
13562	0
13563	0

13564 rows × 1 columns

Accuracy measurement

In [18]:
```
metrics.accuracy_score(y_test, y_pred)
```

Out[18]:
```
0.8856531996461221
```

Grid Search

In [19]:
```
from sklearn.pipeline import Pipeline

# param_grid = {
#     'criterion': ['gini', 'entropy'],
#     'max_depth': [2, 4, 6, 8, 10, 12]
# }

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(clf, param_grid, cv=5)
grid_search = GridSearchCV(pipe, parameters)
grid_search.fit(X_train, y_train)
```

Out[19]:
```
▸ GridSearchCV
▸ estimator: Pipeline
▸ DecisionTreeClassifier
```

Display the best features

In [20]:
```
features = X.columns
importances = clf.feature_importances_

best_features_df = pd.DataFrame({'Features': features, 'Importance': importanc
best_features_df
```

Out[20]:

	Features	Importance
0	default	0.029758
1	housing	0.719692
2	loan	0.250550

Run DecisionTreeClassifier using the obtained features

In [21]:
```
criterion = grid_search.best_estimator_.get_params()['dec_tree__criterion']
# criterion = grid_search.best_params_['criterion']

max_depth = grid_search.best_estimator_.get_params()['dec_tree__max_depth']
# max_depth = grid_search.best_params_['max_depth']

X = df[['housing', 'loan']]
y = df.y

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando

optimized_clf = DecisionTreeClassifier(criterion = criterion, max_depth = max_
optimized_clf.fit(X_train, y_train)
```

Out[21]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2)
```

Concat train test results

In [22]:
```
import numpy as np

y_pred_train = optimized_clf.predict(X_train)
y_pred_test = optimized_clf.predict(X_test)

y_pred_train = y_pred_train.reshape(len(y_pred_train), 1)
y_pred_test = y_pred_test.reshape(len(y_pred_test), 1)

print('Train Result')
print(np.concatenate((y_pred_train, y_train.to_numpy().reshape(len(y_train), 1

print('Test Result')
print(np.concatenate((y_pred_test, y_test.to_numpy().reshape(len(y_test), 1)),
```

```
Train Result
[[0 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
Test Result
[[0 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 1]]
```

Section 2

1. Read "petrol_consumption.csv" file

In [23]:
```
petrol_df = pd.read_csv('../data_samples2/petrol_consumption.csv')
```

2. Display the first 5 records

In [24]:
```
petrol_df.head()
```

Out[24]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumptio
0	9.0	3571	1976	0.525	54
1	9.0	4092	1250	0.572	52
2	9.0	3865	1586	0.580	56
3	7.5	4870	2351	0.529	41
4	8.0	4399	431	0.544	41

4. Identify the label (Petrol_Consumption)

In [25]:
```
y_petrol = petrol_df['Petrol_Consumption']
```

5. Identify the features.

In [26]:
```
X_petrol = petrol_df.drop('Petrol_Consumption', axis=1)
```

6. Use of describe method to describe the dataset.

In [27]:
```
petrol_df.describe()
```

Out[27]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consun
count	48.000000	48.000000	48.000000	48.000000	48.0
mean	7.668333	4241.833333	5565.416667	0.570333	576.7
std	0.950770	573.623768	3491.507166	0.055470	111.8
min	5.000000	3063.000000	431.000000	0.451000	344.0
25%	7.000000	3739.000000	3110.250000	0.529750	509.5
50%	7.500000	4298.000000	4735.500000	0.564500	568.5
75%	8.125000	4578.750000	7156.000000	0.595250	632.7
max	10.000000	5342.000000	17782.000000	0.724000	968.0

7. Display the first 5 records of the features

In [28]:
```
X_petrol.head()
```

Out[28]:

	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 [29]:
```
X_petrol_train, X_petrol_test, y_petrol_train, y_petrol_test = train_test_spli
```

9. Build your model and train the training data

In [30]:
```
reg_petrol = DecisionTreeRegressor()
reg_petrol = reg_petrol.fit(X_petrol_train,y_petrol_train)
reg_petrol
```

Out[30]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

10. Prediction using the testing set

In [31]:
```
y_petrol_pred = reg_petrol.predict(X_petrol_test)
y_petrol_pred
```

Out[31]:
```
array([487., 524., 534., 635., 524., 467., 571., 580., 968., 574.])
```

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

In [32]:
```
compare_df = pd.DataFrame({'Actual': y_petrol_test, 'Predicted': y_petrol_pred
compare_df
```

Out[32]:

	Actual	Predicted
0	628	487.0
1	547	524.0
2	648	534.0
3	640	635.0
4	561	524.0
5	414	467.0
6	554	571.0
7	577	580.0
8	782	968.0
9	631	574.0

12. Evaluate the model using mean_absulate_error

In [33]:
```
from sklearn.metrics import mean_absolute_error

mean_absolute_error(y_petrol_test, y_petrol_pred)
```

Out[33]: 63.6

13. Display the predicted output using first 5 features.

In [34]:
```
compare_df[['Predicted']].head()
```

Out[34]:

	Predicted
0	487.0
1	524.0
2	534.0
3	635.0
4	524.0