Anuradha Ananth

Recreated what was taught in the webinar as well as the assignment

In [14]:
```
# Importing important libraries
## Basic libraries

import pandas as pd
import numpy as np

## Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [15]:
```
# Importing the dataset
df=pd.read_csv('bank.csv')
df.head()
```

Out[15]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	dura
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	

In [16]:
```
# Checking the shape of the data

print(df.shape)
```

```
(11162, 17)
```

In [17]:
```
# Checking for null values

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

Out[17]:
```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
deposit      0
dtype: int64
```

In [18]:
```
# Dropping the duration column
df=df.drop('duration', axis=1)
df.head()
```

Out[18]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	cam
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	

In [19]:
```
# Printing the shape

print(df.shape)
```

```
(11162, 16)
```

In [20]:
```
# getting proportion of different categories in contact
df['contact'].value_counts()
```

Out[20]:
```
cellular     8042
unknown      2346
telephone     774
Name: contact, dtype: int64
```

In [21]:
```
2346/(8042+2346+774)

# 21% of data points are having missing values
```

Out[21]:
```
0.21017738756495252
```

In [22]:
```
# getting proportion of different categories in contact
df['poutcome'].value_counts()
```

Out[22]:
```
unknown    8326
failure    1228
success    1071
other       537
Name: poutcome, dtype: int64
```

In [23]:
```
8326/11162
```

Out[23]:
```
0.7459236695932628
```

In [24]:
```
# Getting proportion of values in target variable
df['deposit'].value_counts()
```

Out[24]:
```
no     5873
yes    5289
Name: deposit, dtype: int64
```

In [25]:
```
5873/(5873+5289)
```

Out[25]:
```
0.5261601863465328
```

In [31]:
```
## Backup
df_ready = df.copy()
```

In [32]:
```
# Standard scaler



#Importing importing libraries

from sklearn.preprocessing import StandardScaler

## Initialization
scaler = StandardScaler()

num_cols = ['age','balance','day','campaign','pdays','previous']

df_ready[num_cols] = scaler.fit_transform(df_ready[num_cols])
df.ready.head()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_3276\336510201.py in <module>
     13
     14 df_ready[num_cols] = scaler.fit_transform(df_ready[num_cols])
---> 15 df.ready.head()

~\anaconda3\lib\site-packages\pandas\core\generic.py in __getattr__(self, name)
   5573         ):
   5574             return self[name]
-> 5575         return object.__getattribute__(self, name)
   5576
   5577     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'ready'
```

In [35]:
```
# Dealing with categorical columns

## List of categorical columns

cat_cols = ['job','marital','education','default','housing','loan','contact','mon

## Encoding
df_ready = pd.get_dummies(data = df_ready, columns = cat_cols)

df_ready['deposit'] = df_ready['deposit'].apply(lambda x : 1 if x == 'yes' else 0
df_ready.head()
```

Out[35]:

	age	balance	day	campaign	pdays	previous	deposit	job_admin.	job_blue colla
0	1.491505	0.252525	-1.265746	-0.554168	-0.481184	-0.36326	1	1	
1	1.239676	-0.459974	-1.265746	-0.554168	-0.481184	-0.36326	1	1	
2	-0.019470	-0.080160	-1.265746	-0.554168	-0.481184	-0.36326	1	0	
3	1.155733	0.293762	-1.265746	-0.554168	-0.481184	-0.36326	1	0	
4	1.071790	-0.416876	-1.265746	-0.186785	-0.481184	-0.36326	1	1	

5 rows × 51 columns

In [36]:
```
# printing the shape

print(df_ready.shape)
```

```
(11162, 51)
```

In [38]:
```
# Separating input and output features

## Input features
features = df_ready.drop('deposit', axis = 1)

## target
target = df_ready['deposit']
```

In [40]:
```
# Train Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, target,
                                                    test_size = 0.2, random_state
```

In [41]:
```
# Logistic regression
# Model
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(max_iter = 10e6)

# Fitting the model
lr.fit(X_train, y_train)
```

Out[41]:
```
LogisticRegression(max_iter=10000000.0)
```

In [42]:
```
# Predicting from the model
y_pred = lr.predict(X_test)
```

In [43]:
```
# Evaluation
from sklearn.metrics import accuracy_score
print('Accuracy:', accuracy_score(y_test,y_pred))
```

```
Accuracy: 0.7017465293327362
```

In [44]:
```
# Support Vector Machines
```

In [45]:
```
# Model
from sklearn.svm import SVC
svm = SVC(kernel='rbf')

# Fitting the model
svm.fit(X_train, y_train)
```

Out[45]:
```
SVC()
```

In [46]:
```
# Predicting from the model
y_pred = svm.predict(X_test)
```

In [47]:
```
# Evaluation
from sklearn.metrics import accuracy_score
print('Accuracy:', accuracy_score(y_test,y_pred))
```

```
Accuracy: 0.7362292879534259
```

In [48]:
```
# Decision tree classifier
```

In [49]:
```
# Model
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()

# Fitting the model
dt.fit(X_train, y_train)
```

Out[49]:
```
DecisionTreeClassifier()
```

In [50]:
```
# Predicting from the model
y_pred = dt.predict(X_test)
```

In [51]:
```
# Evaluation
from sklearn.metrics import accuracy_score
print('Accuracy:', accuracy_score(y_test,y_pred))
```

```
Accuracy: 0.6287505597850426
```

In [82]:
```
from scipy.stats import loguniform
from pandas import read_csv
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
```

In [54]:
```
dt = DecisionTreeClassifier(random_state=42)
```

In [56]:
```
# Create the parameter grid based on the results of random search
params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'criterion': ["gini", "entropy"]
}
```

In [57]:
```
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=dt,
                           param_grid=params,
                           cv=4, n_jobs=-1, verbose=1, scoring = "accuracy")
```

In [60]:
```
# Create gridsearch instance
#
grid = GridSearchCV(estimator=dt,
                    param_grid=params,
                    cv=10,
                    n_jobs=1,
                    verbose=2)
#
# Fit the model
#
grid.fit(X_train, y_train)
#
# Assess the score
#
grid.best_score_, grid.best_params_
```

```
Fitting 10 folds for each of 50 candidates, totalling 500 fits
[CV] END ....criterion=gini, max_depth=2, min_samples_leaf=5; total time=
0.0s
[CV] END ....criterion=gini, max_depth=2, min_samples_leaf=5; total time=
0.0s
[CV] END ....criterion=gini, max_depth=2, min_samples_leaf=5; total time=
0.0s
[CV] END ....criterion=gini, max_depth=2, min_samples_leaf=5; total time=
0.0s
[CV] END ....criterion=gini, max_depth=2, min_samples_leaf=5; total time=
0.0s
[CV] END ....criterion=gini, max_depth=2, min_samples_leaf=5; total time=
0.0s
[CV] END ....criterion=gini, max_depth=2, min_samples_leaf=5; total time=
0.0s
[CV] END ....criterion=gini, max_depth=2, min_samples_leaf=5; total time=
0.0s
[CV] END ....criterion=gini, max_depth=2, min_samples_leaf=5; total time=
0.0s
```

In [67]:
```
tuned_hyper_model= DecisionTreeRegressor(max_depth=5,max_features='auto',max_lea
```

In [70]:
```
reg_decision_model=DecisionTreeRegressor()
```

In [ ]:

In [71]:
```
# fit independent varaibles to the dependent variables
reg_decision_model.fit(X_train,y_train)
```

Out[71]: DecisionTreeRegressor()

In [73]:
```
reg_decision_model.score(X_train,y_train)
```

Out[73]: 1.0

In [74]:
```
reg_decision_model.score(X_test,y_test)
```

Out[74]:
```
-0.4576281840512153
```

In [75]:
```
tuned_hyper_model.fit(X_train,y_train)
```

Out[75]:
```
DecisionTreeRegressor(max_depth=5, max_features='auto', max_leaf_nodes=50,
                      min_samples_leaf=2, min_weight_fraction_leaf=0.1,
                      splitter='random')
```

In [76]:
```
# prediction

tuned_pred=tuned_hyper_model.predict(X_test)
```

In [77]:
```
# With hyperparameter tuned

from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test,tuned_pred))
print('MSE:', metrics.mean_squared_error(y_test, tuned_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, tuned_pred)))
```

```
MAE: 0.4365479518929114
MSE: 0.21886977383970338
RMSE: 0.4678351994449577
```

In [79]:
```
# without hyperparameter tuning
prediction=reg_decision_model.predict(X_test)
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test,prediction))
print('MSE:', metrics.mean_squared_error(y_test, prediction))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, prediction)))
```

```
MAE: 0.3627407075682938
MSE: 0.3627407075682938
RMSE: 0.6022795925218567
```

In [87]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import randint as sp_randint
from sklearn.ensemble import RandomForestClassifier
```

In [88]:
```
# build a RandomForestClassifier
clf = RandomForestClassifier(n_estimators=50)
```

In [89]:
```
#Specifying the list of parameters and distributions

param_dist = {"max_depth": [3, None],
              "max_features": sp_randint(1, 11),
              "min_samples_split": sp_randint(2, 11),
              "min_samples_leaf": sp_randint(1, 11),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}
```

In [90]:
```
#Defining the sample, distributions and cross-validation

samples = 8  # number of random samples
randomCV = RandomizedSearchCV(clf, param_distributions=param_dist, n_iter=samples
```

In [95]:
```
#All parameters are set and, let's do the fit model

randomCV.fit(X_train, y_train)
print(randomCV.best_params_)
```

```
{'bootstrap': False, 'criterion': 'entropy', 'max_depth': None, 'max_features':
8, 'min_samples_leaf': 7, 'min_samples_split': 10}
```

In [97]:
```
print(randomCV.score(X_test,y_test))
```

```
0.7460815047021944
```

Hence we can see that, by using GridSearchCv() and training

the Decision Tree Classifier using optimal values of the hyper-parameters, the accuracy increases to 74% from 62.87%

In [ ]: