BAN 210: Workshop 5

Import data

In [146…
```
import pandas as pd
```

In [147…
```
# Loading the Dataset Bank
df = pd.read_csv('bank (1).csv')
df.head() #Checking and printing the first 5 rows
```

Out[147…

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	po
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	u
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	u
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	u

In [148…
```
# Data Inspection
df.info() # Checking the structure and data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        4521 non-null   int64
 1   job        4521 non-null   object
 2   marital    4521 non-null   object
 3   education  4521 non-null   object
 4   default    4521 non-null   object
 5   balance    4521 non-null   int64
 6   housing    4521 non-null   object
 7   loan       4521 non-null   object
 8   contact    4521 non-null   object
 9   day        4521 non-null   int64
 10  month      4521 non-null   object
 11  duration   4521 non-null   int64
 12  campaign   4521 non-null   int64
 13  pdays      4521 non-null   int64
 14  previous   4521 non-null   int64
 15  poutcome   4521 non-null   object
 16  y          4521 non-null   object
dtypes: int64(7), object(10)
memory usage: 600.6+ KB
```

The descriptive statistics of the data

In [150…

```python
# Checking and Printing the Descriptive Statistics of the numerical values
Descriptive_numeric = df.select_dtypes(include=['int64','float64'])

#Calculating the required statistics
statistics = pd.DataFrame({
    'Min': Descriptive_numeric.min(),
    'Q1 (25%)': Descriptive_numeric.quantile(0.25),
    'Median (50%)': Descriptive_numeric.median(),
    'Q3 (75%)': Descriptive_numeric.quantile(0.75),
    'Max': Descriptive_numeric.max(),
    'Mean': Descriptive_numeric.mean()
})

# Printing in Table format the Descriptive Statistics
statistics
```

Out[150…

	Min	Q1 (25%)	Median (50%)	Q3 (75%)	Max	Mean
age	19	33.0	39.0	49.0	87	41.170095
balance	-3313	69.0	444.0	1480.0	71188	1422.657819
day	1	9.0	16.0	21.0	31	15.915284
duration	4	104.0	185.0	329.0	3025	263.961292
campaign	1	1.0	2.0	3.0	50	2.793630
pdays	-1	-1.0	-1.0	-1.0	871	39.766645
previous	0	0.0	0.0	0.0	25	0.542579

In [151…
```
df.columns
```

Out[151…
```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

Splitting the Data-Set into Independent and Dependent Features

In [153…
```
# Independent Features
X = df.drop("y", axis=1)
X.head()
```

Out[153…

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0

In [154…
```
# Dependent Features
y = df["y"]
y.head()
```

Out[154…
```
0    no
1    no
2    no
3    no
4    no
Name: y, dtype: object
```

In [155…
```
print("Shape X:", X.shape)
print("Shape y:", y.shape)
```

```
Shape X: (4521, 16)
Shape y: (4521,)
```

Convert categorical variable into numeric - Using one hot encoding method

In [157…
```
categorical_columns = X.select_dtypes(include="object").columns.tolist()
print(categorical_columns)
```

```
['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome']
```

In [158…
```
X_encoded = pd.get_dummies(X, columns=categorical_columns, drop_first=True)
```

In [159…
```
X_encoded = X_encoded.astype(int)
bool_cols = X_encoded.select_dtypes(include='bool').columns
X_encoded[bool_cols] = X_encoded[bool_cols].astype(int)
X_encoded = pd.get_dummies(X, columns=categorical_columns, drop_first=True, dtype=int)
X_encoded.head()
```

Out[159…

	age	balance	day	duration	campaign	pdays	previous	job_blue-collar	job_entrepreneur	job_housemaid	…	month_jul	month_jun	mo
0	30	1787	19	79	1	-1	0	0	0	0	…	0	0	
1	33	4789	11	220	1	339	4	0	0	0	…	0	0	
2	35	1350	16	185	1	330	1	0	0	0	…	0	0	
3	30	1476	3	199	4	-1	0	0	0	0	…	0	1	
4	59	0	5	226	1	-1	0	1	0	0	…	0	0	

5 rows × 42 columns

In [160…
```
print("Before encoding:", X.shape)
print("After encoding:", X_encoded.shape)
```

```
Before encoding: (4521, 16)
After encoding: (4521, 42)
```

In [161…
```
#The list of dummy variables created by one-hot encoding
print(X_encoded.columns.difference(X.columns))
```

```
Index(['contact_telephone', 'contact_unknown', 'default_yes',
       'education_secondary', 'education_tertiary', 'education_unknown',
       'housing_yes', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
       'job_management', 'job_retired', 'job_self-employed', 'job_services',
       'job_student', 'job_technician', 'job_unemployed', 'job_unknown',
       'loan_yes', 'marital_married', 'marital_single', 'month_aug',
       'month_dec', 'month_feb', 'month_jan', 'month_jul', 'month_jun',
       'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep',
       'poutcome_other', 'poutcome_success', 'poutcome_unknown'],
      dtype='object')
```

Normalize the data set

In [163…
```
# part 5
# feature scaling
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
# Only use numeric columns for scaling
numeric_columns = X_encoded.select_dtypes(include=['int64', 'float64']).columns
X_encoded[numeric_columns] = scaler.fit_transform(X_encoded[numeric_columns])

# Check the result
X_encoded[numeric_columns].describe()
```

Out[163…

	age	balance	day	duration	campaign	pdays	previous
count	4.521000e+03	4.521000e+03	4.521000e+03	4.521000e+03	4.521000e+03	4.521000e+03	4.521000e+03
mean	-1.178737e-16	-1.571649e-17	2.357474e-18	-7.622500e-17	-6.286598e-18	-3.889832e-17	3.929124e-18
std	1.000111e+00	1.000111e+00	1.000111e+00	1.000111e+00	1.000111e+00	1.000111e+00	1.000111e+00
min	-2.096455e+00	-1.573671e+00	-1.808625e+00	-1.000513e+00	-5.768295e-01	-4.072183e-01	-3.204128e-01
25%	-7.725828e-01	-4.498240e-01	-8.385461e-01	-6.156433e-01	-5.768295e-01	-4.072183e-01	-3.204128e-01
50%	-2.052091e-01	-3.252105e-01	1.027262e-02	-3.038984e-01	-2.552305e-01	-4.072183e-01	-3.204128e-01
75%	7.404137e-01	1.905496e-02	6.165717e-01	2.503146e-01	6.636847e-02	-4.072183e-01	-3.204128e-01
max	4.333780e+00	2.318321e+01	1.829170e+00	1.062641e+01	1.518152e+01	8.303196e+00	1.444300e+01

Divide the dataset to training and test sets

In [165…

```
# Part 6
from sklearn.model_selection import train_test_split

# Converting target variable to binary
y_binary = y.map({'yes': 1, 'no': 0})

# Spliting the dataset: 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y_binary, test_size=0.2, random_state=42)

# Checking the shape
print("Shape - Training Set:", X_train.shape)
print("Shape - Test Set:", X_test.shape)
```

```
Shape - Training Set: (3616, 42)
Shape - Test Set: (905, 42)
```

Use the KNN, Decision Tree and Logistic Regression algorithm to predict the test set out values

In [167…
```
# Import Libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [168…
```
# Use cross-validation to find the best value of n_neighbors
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

# Define the parameter grid: try n_neighbors from 1 to 20
param_grid = {'n_neighbors': list(range(1, 21))}
# Initialize the KNN model
knn = KNeighborsClassifier()
# Set up GridSearch with 5-fold cross-validation
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')

# Fit the model on the training data
grid_search.fit(X_train, y_train)
# Print the best value of n_neighbors
print("Best n_neighbors:", grid_search.best_params_['n_neighbors'])
# Print the best cross-validated accuracy score
print("Best cross-validation accuracy:", grid_search.best_score_)
```

```
Best n_neighbors: 9
Best cross-validation accuracy: 0.8921459847321245
```

In [169…
```
# KNN Model
knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)

print("KNN Model")
print("Confusion Matrix:\n", confusion_matrix(y_test, knn_pred))
print("Classification Report:\n", classification_report(y_test, knn_pred))
```

```
KNN Model
Confusion Matrix:
 [[792  15]
 [ 83  15]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.98      0.94       807
           1       0.50      0.15      0.23        98

    accuracy                           0.89       905
   macro avg       0.70      0.57      0.59       905
weighted avg       0.86      0.89      0.87       905
```

In [170…

```python
# Decision Tree Model
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
dt_pred = dt.predict(X_test)

print("Decision Tree Model")
print("Confusion Matrix:\n", confusion_matrix(y_test, dt_pred))
print("Classification Report:\n", classification_report(y_test, dt_pred))
```

```
Decision Tree Model
Confusion Matrix:
 [[749  58]
 [ 49  49]]
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.93      0.93       807
           1       0.46      0.50      0.48        98

    accuracy                           0.88       905
   macro avg       0.70      0.71      0.71       905
weighted avg       0.89      0.88      0.88       905
```

In [171…

```python
# Logistic Regression Model
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)

print("Logistic Regression Model")
print("Confusion Matrix:\n", confusion_matrix(y_test, lr_pred))
print("Classification Report:\n", classification_report(y_test, lr_pred))
```

```
Logistic Regression Model
Confusion Matrix:
 [[789  18]
 [ 71  27]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95       807
           1       0.60      0.28      0.38        98

    accuracy                           0.90       905
   macro avg       0.76      0.63      0.66       905
weighted avg       0.88      0.90      0.88       905
```

Evaluate the model performance by computing Accuracy.

In [193…

```
print("KNN Accuracy:", accuracy_score(y_test, knn_pred))
print("Decision Tree Accuracy:", accuracy_score(y_test, dt_pred))
print("Logistic Regression Accuracy:", accuracy_score(y_test, lr_pred))
```

```
KNN Accuracy: 0.8917127071823204
Decision Tree Accuracy: 0.881767955801105
Logistic Regression Accuracy: 0.901657458563536
```

Comments: All three models perform well (over 88% accuracy): the bank data is well-prepared and has good predictive power.

Firts, Logistic Regression performs best:

The problem is linearly separable (Logistic Regression is good at finding linear boundaries). The features are well scaled (using StandardScaler), and that benefits Logistic Regression the most. Data is not highly complex or non-linear.

Second, KNN is close to Logistic Regression:

The dataset is clean, and n_neighbors is well-chosen. However, KNN can become slower on large datasets and is more sensitive to irrelevant features or noise.

Decision Tree is slightly behind:

This model is more prone to overfitting (especially without pruning or tuning). It's good at capturing non-linear patterns, but might not generalize well on unseen data.