Name :- Ansh Agrawal

Roll NO : 20

Practical 5

Import Libraries

In [20]:

```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,mean_absolute_error,mean_squared_error,r2_s
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
```

Collect datasets

In [22]:

```
df = pd.read_csv("banking.csv");
```

In [23]:

```
df.head()
```

Out[23]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_weel
0	44	blue-collar	married	basic.4y	unknown	yes	no	cellular	aug	thu
1	53	technician	married	unknown	no	no	no	cellular	nov	fr
2	28	management	single	university.degree	no	yes	no	cellular	jun	thu
3	39	services	married	high.school	no	no	no	cellular	apr	fr
4	55	retired	married	basic.4y	no	yes	no	cellular	aug	fr

5 rows × 21 columns

In [24]:

```
df.tail()
```

Out[24]:

	age	job	marital	education	default	housing	loan	contact	month	day_
41183	59	retired	married	high.school	unknown	no	yes	telephone	jun	
41184	31	housemaid	married	basic.4y	unknown	no	no	telephone	may	
41185	42	admin.	single	university.degree	unknown	yes	yes	telephone	may	
41186	48	technician	married	professional.course	no	no	yes	telephone	oct	
41187	25	student	single	high.school	no	no	no	telephone	may	

5 rows × 21 columns

In [25]:

```
df.describe()
```

Out[25]:

	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767

In [26]:

```
df.shape
```

Out[26]:

```
(41188, 21)
```

In [27]:

```
df.dtypes
```

Out[27]:

```
age                 int64
job                object
marital            object
education          object
default            object
housing            object
loan               object
contact            object
month              object
day_of_week        object
duration            int64
campaign            int64
pdays               int64
previous            int64
poutcome           object
emp_var_rate      float64
cons_price_idx    float64
cons_conf_idx     float64
euribor3m         float64
nr_employed       float64
y                   int64
dtype: object
```

In [28]:

```
# Check for missing values
df.isnull().sum()
```

Out[28]:

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp_var_rate      0
cons_price_idx    0
cons_conf_idx     0
euribor3m         0
nr_employed       0
y                 0
dtype: int64
```

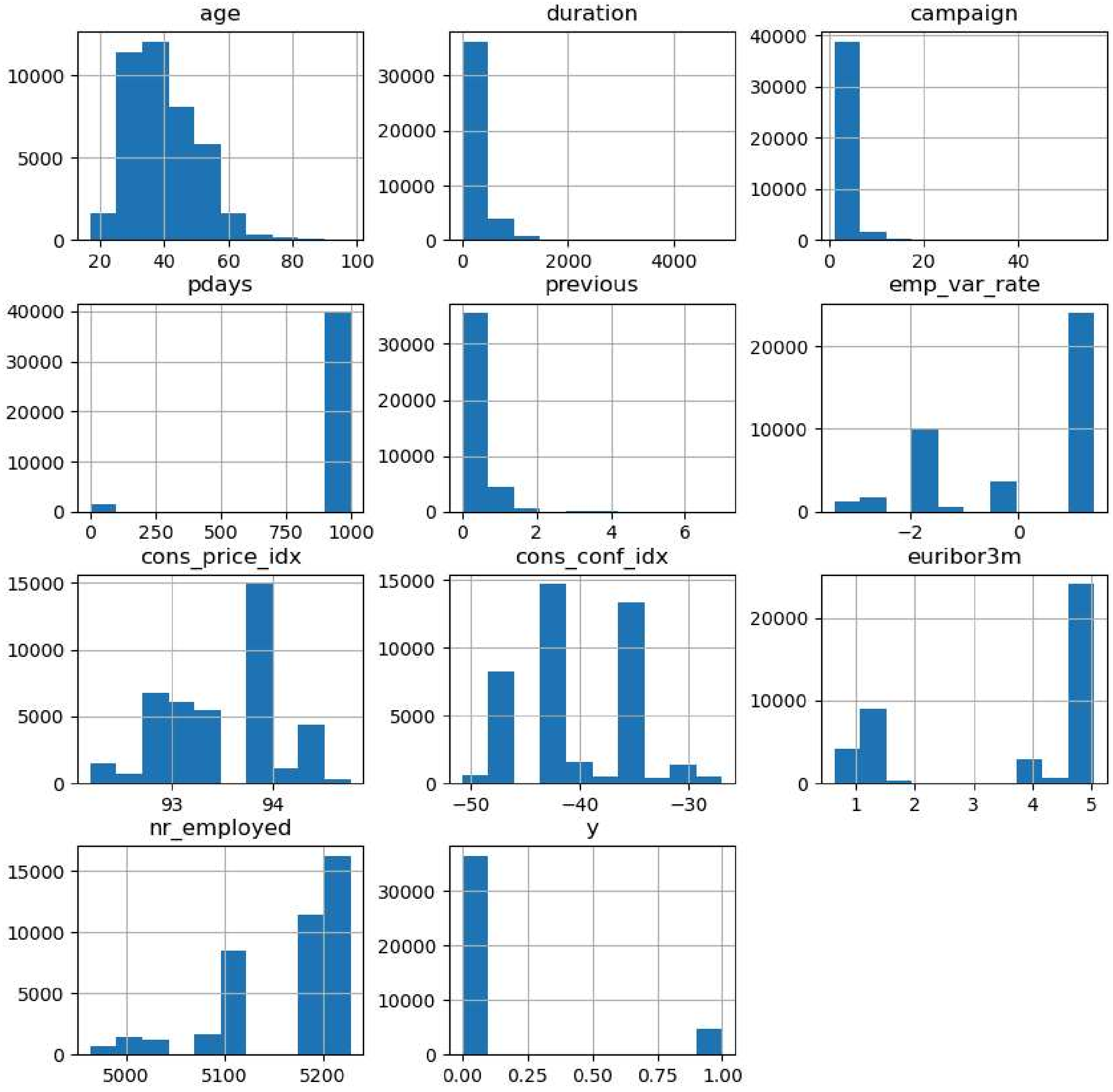
In [29]:

```
print(df.duplicated().sum())
```

```
12
```

In [30]:

```
import matplotlib.pyplot as plt
# Histograms for numerical features
df3.hist(figsize=(10, 10))
plt.show()
```



In [31]:
```
# Split the data into features (X) and target (y)
X = df.drop('loan', axis=1)
y = df['loan']
```

In [32]:
```
# Generate a synthetic dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
```

Random Forest

In [34]:
```
# Build a random forest classifier with 100 trees
rd = RandomForestClassifier(n_estimators=100, random_state=1)

# Train the classifier on the training data
rd.fit(X_train, y_train)
```

Out[34]:

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=1)
```

In [35]:

```
# Make predictions on the test data
y_pred = rd.predict(X_test)

# Evaluate the performance of the classifier
accuracy = rd.score(X_test, y_test)
print("Accuracy: %.2f" % accuracy)
```

```
Accuracy: 0.95
```

In [36]:

```
# Calculate R² score
r2 = r2_score(y_test, y_pred)
print(f"R² score: {r2:.2f}")

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error: {mae:.2f}")
```

```
R² score: 0.80
Mean Absolute Error: 0.05
```

In [37]:

```
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       141
           1       0.99      0.91      0.95       159

    accuracy                           0.95       300
   macro avg       0.95      0.95      0.95       300
weighted avg       0.95      0.95      0.95       300
```

Apply Hyper paraamter tunning

In [39]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
```

In [40]:

```
# Define the hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}
```

In [41]:

```
# Create a Random Forest model
rf = RandomForestClassifier(random_state=42)

# Set up GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1, verbo
# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best parameters found: ", best_params)
```

```
Fitting 5 folds for each of 360 candidates, totalling 1800 fits
Best parameters found:  {'bootstrap': False, 'max_depth': None, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 50}
```

In [42]:

```
# Define the parameter distributions
param_dist = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Set up RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_dist, n_ite

# Fit RandomizedSearchCV
random_search.fit(X_train, y_train)

# Get the best parameters
best_params = random_search.best_params_
print("Best parameters found: ", best_params)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best parameters found:  {'n_estimators': 50, 'min_samples_split': 2, 'min_samples_lea
f': 2, 'max_depth': 30, 'bootstrap': False}
```

In [43]:

```
# Retrieve the best model from RandomizedSearchCV
best_rf = random_search.best_estimator_

# Make predictions
y_pred = best_rf.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Test set accuracy: ", accuracy)
```

```
Test set accuracy:  0.9666666666666667
```

Adaboost Algorithm

In [45]:

```
# Train the AdaBoost classifier
ada = AdaBoostClassifier(n_estimators=50, learning_rate=1.0, random_state=42)
ada.fit(X_train, y_train)

# Make predictions on the test set
y_pred = ada.predict(X_test)

# Calculate accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the AdaBoost classifier: {accuracy *100:.2f}")
```

```
Accuracy of the AdaBoost classifier: 91.33
```

In [46]:

```
# Calculate R² score
r2 = r2_score(y_test, y_pred)
print(f"R² score: {r2:.2f}")

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error: {mae:.2f}")
```

```
R² score: 0.65
Mean Absolute Error: 0.09
```

In [47]:

```
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.86      0.97      0.91       141
           1       0.97      0.86      0.91       159

    accuracy                           0.91       300
   macro avg       0.92      0.92      0.91       300
weighted avg       0.92      0.91      0.91       300
```

Hyper Paramter Tunning

In [49]:

```
# Define the parameter distributions for hyperparameter tuning
param_dist = {
    'n_estimators': [50, 100, 200, 300, 500],  # Number of estimators
    'learning_rate': [0.001, 0.01, 0.1, 0.5, 1.0, 1.5],  # Learning rate
    'algorithm': ['SAMME', 'SAMME.R']  # Algorithm type
}

# Set up RandomizedSearchCV for hyperparameter tuning
random_search = RandomizedSearchCV(
    estimator=ada,
    param_distributions=param_dist,
    n_iter=20,  # Number of parameter settings that are sampled
    cv=5,       # Number of folds for cross-validation
    n_jobs=-1,  # Use all available cores for parallelization
    verbose=2,  # Verbosity level
    random_state=42
)

# Fit RandomizedSearchCV on the training data
random_search.fit(X_train, y_train)

# Get the best parameters from RandomizedSearchCV
best_params = random_search.best_params_
print("Best parameters found: ", best_params)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best parameters found:  {'n_estimators': 200, 'learning_rate': 0.5, 'algorithm': 'SAM
ME'}
```

In [50]:

```
# Retrieve the best AdaBoost model from RandomizedSearchCV
best_ada = random_search.best_estimator_
```

```python
# Make predictions using the best AdaBoost model
y_pred = best_ada.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the tuned AdaBoost classifier: {accuracy * 100:.2f}")
```

```
Accuracy of the tuned AdaBoost classifier: 91.00
```