In [1]:

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
```

In [2]:

```
data=pd.read_csv(r"C:\Users\Kishore\OneDrive\Desktop\CSV Files\bankloan.csv")
```

In [3]:

```
data
```

Out[3]:

	ID	Age	Experience	Income	ZIP.Code	Family	CCAvg	Education	Mortgage	Personal.Loan	Se
0	1	25	1	49	91107	4	1.6	1	0	0	
1	2	45	19	34	90089	3	1.5	1	0	0	
2	3	39	15	11	94720	1	1.0	1	0	0	
3	4	35	9	100	94112	1	2.7	2	0	0	
4	5	35	8	45	91330	4	1.0	2	0	0	
...	...	...	...	...	...	...	...	...	...	...	
4995	4996	29	3	40	92697	1	1.9	3	0	0	
4996	4997	30	4	15	92037	4	0.4	1	85	0	
4997	4998	63	39	24	93023	2	0.3	3	0	0	
4998	4999	65	40	49	90034	3	0.5	2	0	0	
4999	5000	28	4	83	92612	3	0.8	1	0	0	

5000 rows × 14 columns

In [4]:

```
data.shape
```

Out[4]:

```
(5000, 14)
```

In [5]:
```
data.describe()
```

Out[5]:

	ID	Age	Experience	Income	ZIP.Code	Family	CCAvg	Ec
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000
mean	2500.500000	45.338400	20.104600	73.774200	93152.503000	2.396400	1.937938	1
std	1443.520003	11.463166	11.467954	46.033729	2121.852197	1.147663	1.747659	0
min	1.000000	23.000000	-3.000000	8.000000	9307.000000	1.000000	0.000000	1
25%	1250.750000	35.000000	10.000000	39.000000	91911.000000	1.000000	0.700000	1
50%	2500.500000	45.000000	20.000000	64.000000	93437.000000	2.000000	1.500000	2
75%	3750.250000	55.000000	30.000000	98.000000	94608.000000	3.000000	2.500000	3
max	5000.000000	67.000000	43.000000	224.000000	96651.000000	4.000000	10.000000	3

In [6]:
```
data.head(5)
```

Out[6]:

	ID	Age	Experience	Income	ZIP.Code	Family	CCAvg	Education	Mortgage	Personal.Loan	Securitie
0	1	25	1	49	91107	4	1.6	1	0	0	
1	2	45	19	34	90089	3	1.5	1	0	0	
2	3	39	15	11	94720	1	1.0	1	0	0	
3	4	35	9	100	94112	1	2.7	2	0	0	
4	5	35	8	45	91330	4	1.0	2	0	0	

In [7]:
```
data.tail(5)
```

Out[7]:

	ID	Age	Experience	Income	ZIP.Code	Family	CCAvg	Education	Mortgage	Personal.Loan	Se
4995	4996	29	3	40	92697	1	1.9	3	0	0	
4996	4997	30	4	15	92037	4	0.4	1	85	0	
4997	4998	63	39	24	93023	2	0.3	3	0	0	
4998	4999	65	40	49	90034	3	0.5	2	0	0	
4999	5000	28	4	83	92612	3	0.8	1	0	0	

In [8]:
```
data.shape
```

Out[8]:
```
(5000, 14)
```

In [9]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  5000 non-null   int64  
 1   Age                 5000 non-null   int64  
 2   Experience          5000 non-null   int64  
 3   Income              5000 non-null   int64  
 4   ZIP.Code            5000 non-null   int64  
 5   Family              5000 non-null   int64  
 6   CCAvg               5000 non-null   float64
 7   Education           5000 non-null   int64  
 8   Mortgage            5000 non-null   int64  
 9   Personal.Loan       5000 non-null   int64  
 10  Securities.Account  5000 non-null   int64  
 11  CD.Account          5000 non-null   int64  
 12  Online              5000 non-null   int64  
 13  CreditCard          5000 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 547.0 KB
```

In [10]:

```
data.isnull().sum()
```

Out[10]:

```
ID                    0
Age                   0
Experience            0
Income                0
ZIP.Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal.Loan         0
Securities.Account    0
CD.Account            0
Online                0
CreditCard            0
dtype: int64
```

In [11]:

```
data.min()
```

Out[11]:

```
ID                       1.0
Age                     23.0
Experience              -3.0
Income                   8.0
ZIP.Code              9307.0
Family                   1.0
CCAvg                    0.0
Education                1.0
Mortgage                 0.0
Personal.Loan            0.0
Securities.Account       0.0
CD.Account               0.0
Online                   0.0
CreditCard               0.0
dtype: float64
```

In [12]:

```
data.max()
```

Out[12]:

```
ID                     5000.0
Age                      67.0
Experience               43.0
Income                  224.0
ZIP.Code              96651.0
Family                    4.0
CCAvg                    10.0
Education                 3.0
Mortgage                635.0
Personal.Loan             1.0
Securities.Account        1.0
CD.Account                1.0
Online                    1.0
CreditCard                1.0
dtype: float64
```

In [13]:

```
list(data)
```

Out[13]:

```
['ID',
 'Age',
 'Experience',
 'Income',
 'ZIP.Code',
 'Family',
 'CCAvg',
 'Education',
 'Mortgage',
 'Personal.Loan',
 'Securities.Account',
 'CD.Account',
 'Online',
 'CreditCard']
```

In [14]:

```
data1 = data.drop(['Securities.Account', 'ZIP.Code','Experience','Mortgage'],axis=1)
data1
```

Out[14]:

	ID	Age	Income	Family	CCAvg	Education	Personal.Loan	CD.Account	Online	CreditCard
0	1	25	49	4	1.6	1	0	0	0	0
1	2	45	34	3	1.5	1	0	0	0	0
2	3	39	11	1	1.0	1	0	0	0	0
3	4	35	100	1	2.7	2	0	0	0	0
4	5	35	45	4	1.0	2	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...
4995	4996	29	40	1	1.9	3	0	0	1	0
4996	4997	30	15	4	0.4	1	0	0	1	0
4997	4998	63	24	2	0.3	3	0	0	0	0
4998	4999	65	49	3	0.5	2	0	0	1	0
4999	5000	28	83	3	0.8	1	0	0	1	1

5000 rows × 10 columns

In [15]:

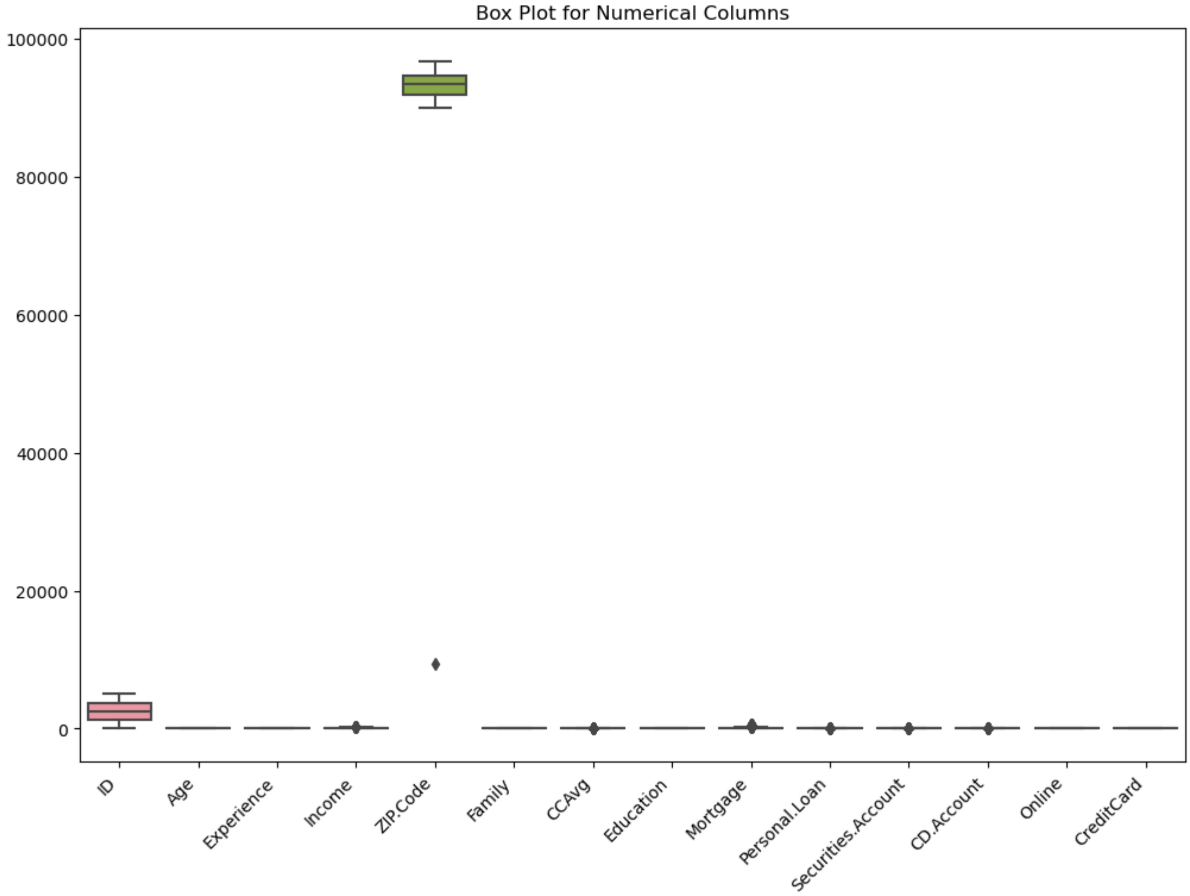
```
numerical_columns = data.select_dtypes(include=['number'])

# Create a box plot for all numerical columns

plt.figure(figsize=(12, 8))
sns.boxplot(data=numerical_columns)
plt.title("Box Plot for Numerical Columns")

# Rotate x-axis labels for better visibility
plt.xticks(rotation=45, ha="right")

# Show the plot
plt.show()
```



In [16]:

```
##  as we look at Box plots we have outliers in Zip_code column. Thus We Won't use it
## then we will drop it
```

In [17]:

```
# Assuming 'ZIP_Code' and 'Personal_Loan' are columns in your DataFrame
x = data.drop(['ZIP.Code', 'Personal.Loan', 'ID'], axis=1)  # Drop the specified colu
y = data['Personal.Loan']  # Set the target variable
```

In [18]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state
```

In [19]:

```
# Create a pipeline
pipeline_rf = Pipeline([
    ('classifier', RandomForestClassifier())
])
```

In [20]:
```
# Fit the pipeline
pipeline_rf.fit(x_train, y_train)
```

Out[20]:
```
▸ Pipeline
  ▸ RandomForestClassifier
```

In [21]:
```
# Make predictions
y_pred_rf = pipeline_rf.predict(x_test)

# Evaluate the performance
accuracy_rf = accuracy_score(y_test, y_pred_rf)

print("Random Forest Accuracy:", accuracy_rf)
```

```
Random Forest Accuracy: 0.99
```

In [ ]:

In [22]:
```
from sklearn.linear_model import LogisticRegression

pipeline_lr = Pipeline([
    ('classifier', LogisticRegression())
])
```

In [23]:
```
pipeline_lr.fit(x_train, y_train)
```

Out[23]:
```
▸ Pipeline
  ▸ LogisticRegression
```

In [24]:
```
y_pred_lr = pipeline_lr.predict(x_test)

accuracy_lr = accuracy_score(y_test, y_pred_lr)

print("Logistic Regression Accuracy:", accuracy_lr)
```

```
Logistic Regression Accuracy: 0.952
```

In [25]:
```
##  we will use GridSearchCV and will print Best parameters can get Higher Performanc
```

In [26]:
```
grid_rf = {
    'n_estimators': [10, 50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
}
```

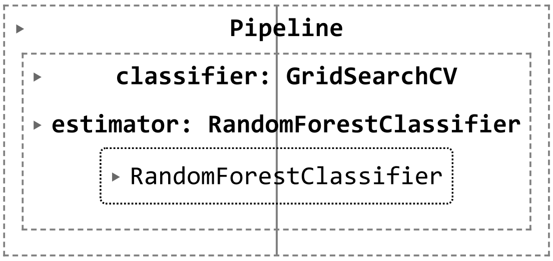
In [27]:

```
# Create a pipeline with GridSearchCV for RandomForestClassifier
pipeline_rf_cv = Pipeline([
    ('classifier', GridSearchCV(RandomForestClassifier(), grid_rf, cv=5))
])
```

In [29]:

```
#Fit the pipeline with cross-validation and hyperparameter tuning
pipeline_rf_cv.fit(x_train, y_train)
```

Out[29]:



```
▸ Pipeline
  ▸ classifier: GridSearchCV
  ▸ estimator: RandomForestClassifier
      ▸ RandomForestClassifier
```

In [32]:

```
y_pred_rf_cv = pipeline_rf_cv.predict(x_test)

accuracy_rf_cv = accuracy_score(y_test, y_pred_rf_cv)
print("Random Forest Accuracy (with CV):", accuracy_rf_cv)
```

```
Random Forest Accuracy (with CV): 0.99
```

In [33]:

```
best_params_rf = pipeline_rf_cv.named_steps['classifier'].best_params_
print("\nBest Hyperparameters for RandomForestClassifier:")
print(best_params_rf)
```

```

Best Hyperparameters for RandomForestClassifier:
{'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 50}
```

In [34]:

```
# Fit the pipeline on the training data
pipeline_rf_cv.fit(x_train, y_train)

# Make predictions on the training set
y_pred_train = pipeline_rf_cv.predict(x_train)

# Make predictions on the test set
y_pred_test = pipeline_rf_cv.predict(x_test)

# Calculate accuracy for training set
accuracy_train = accuracy_score(y_train, y_pred_train)
print("Training Set Accuracy:", accuracy_train)

# Calculate accuracy for test set
accuracy_test = accuracy_score(y_test, y_pred_test)
print("Test Set Accuracy:", accuracy_test)
```

```
Training Set Accuracy: 0.99825
Test Set Accuracy: 0.989
```

In [ ]: