Customer Dataset

```
import pandas as pd
dataset_path = '/content/Customer Data.csv'
df = pd.read_csv(dataset_path)
df.head()
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PU
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	

Next steps: Generate code with `df` View recommended plots

1.Perform EDA on the Dataset and draw the insights.

```
missing_values = df.isnull().sum()
missing_values[missing_values > 0]
```

```
CREDIT_LIMIT          1
MINIMUM_PAYMENTS    313
dtype: int64
```

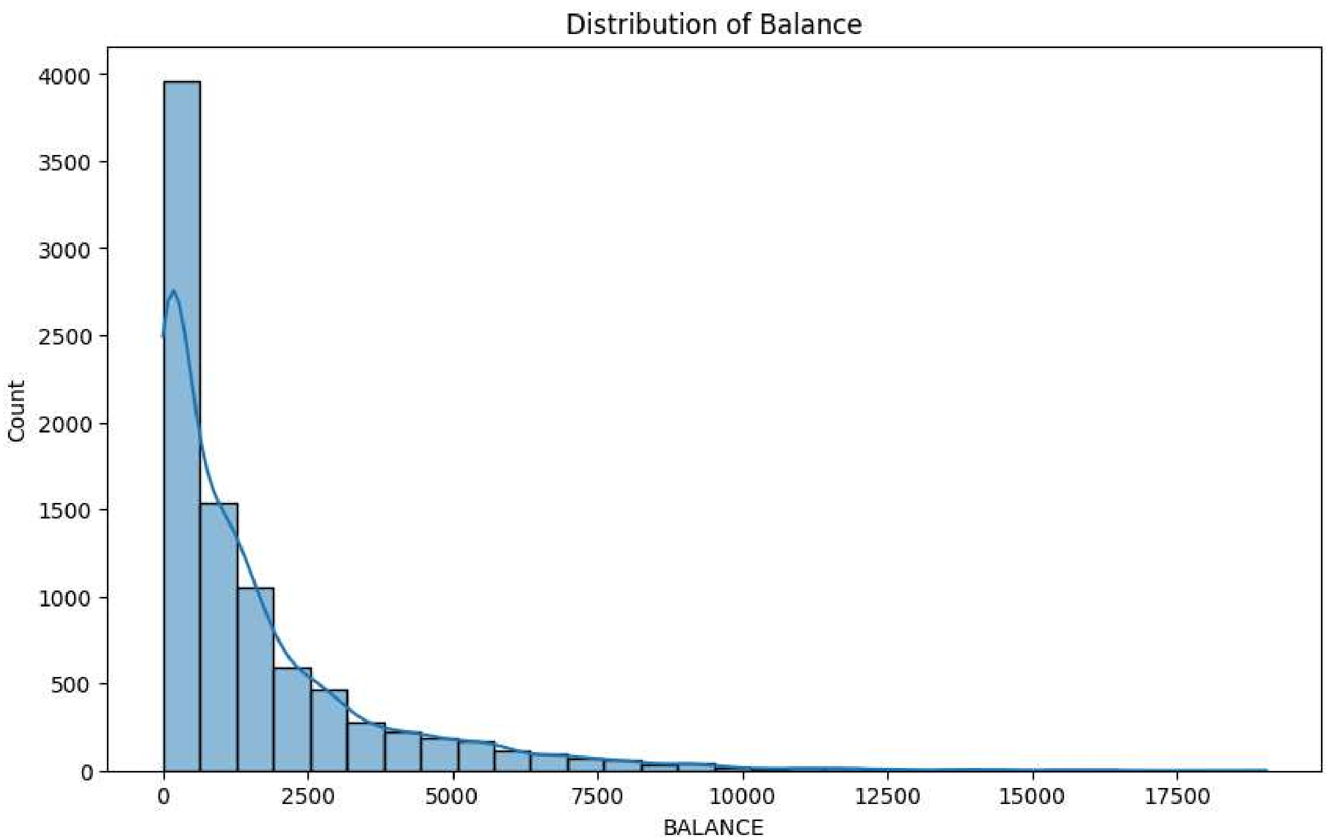
```
summary_stats = df.describe()
summary_stats
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCH
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	0.000000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	

Next steps: Generate code with `summary_stats` View recommended plots

```
df['CREDIT_LIMIT'].fillna(df['CREDIT_LIMIT'].mean(), inplace=True)
df['MINIMUM_PAYMENTS'].fillna(df['MINIMUM_PAYMENTS'].mean(), inplace=True)
```

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10, 6))
sns.histplot(df['BALANCE'], bins=30, kde=True)
plt.title('Distribution of Balance')
plt.show()
```



2. Prepare the dataset for Machine Learning

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Prepare the data
X = df.drop(['CUST_ID', 'TENURE'], axis=1)
y = df['TENURE']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

3. Apply Decision Tree, Random Forest and Naïve Bayes to classify the customers based on the tenures

```
# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train_scaled, y_train)
y_pred_dt = dt.predict(X_test_scaled)
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))

# Random Forest
rf = RandomForestClassifier()
rf.fit(X_train_scaled, y_train)
y_pred_rf = rf.predict(X_test_scaled)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))

# Naïve Bayes
nb = GaussianNB()
nb.fit(X_train_scaled, y_train)
y_pred_nb = nb.predict(X_test_scaled)
print("Naïve Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
```

```
Decision Tree Accuracy: 0.888268156424581
Random Forest Accuracy: 0.9055865921787709
Naïve Bayes Accuracy: 0.3268156424581006
```

6. Clustering

```
from sklearn.cluster import KMeans

X_clustering = df.drop(['CUST_ID', 'TENURE'], axis=1)

# Apply KMeans
kmeans = KMeans(n_clusters=4, random_state=42)
clusters = kmeans.fit_predict(X_clustering)
df['Cluster'] = clusters
cluster_names = {0: 'Low Balance', 1: 'Moderate Users', 2: 'High Spenders', 3: 'Cash Advance Users'}
df['Cluster Name'] = df['Cluster'].map(cluster_names)
df[['CUST_ID', 'Cluster', 'Cluster Name']].head()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_ini
  warnings.warn(
```

	CUST_ID	Cluster	Cluster Name
0	C10001	1	Moderate Users
1	C10002	0	Low Balance
2	C10003	0	Low Balance
3	C10004	0	Low Balance
4	C10005	1	Moderate Users

4. Optimization

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_clustering)
```

```
# Apply PCA
pca = PCA(n_components=0.95)  # Retain 95% of the variance
X_pca = pca.fit_transform(X_scaled)

explained_variance = pca.explained_variance_ratio_.sum()
print(f"Explained variance by PCA components: {explained_variance:.2f}")

X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train_pca, y_train)
y_pred_dt = dt.predict(X_test_pca)
print("Decision Tree Accuracy after PCA:", accuracy_score(y_test, y_pred_dt))

# Random Forest
rf = RandomForestClassifier()
rf.fit(X_train_pca, y_train)
y_pred_rf = rf.predict(X_test_pca)
print("Random Forest Accuracy after PCA:", accuracy_score(y_test, y_pred_rf))

# Naïve Bayes
nb = GaussianNB()
nb.fit(X_train_pca, y_train)
y_pred_nb = nb.predict(X_test_pca)
print("Naïve Bayes Accuracy after PCA:", accuracy_score(y_test, y_pred_nb))
```

```
Explained variance by PCA components: 0.96
Decision Tree Accuracy after PCA: 0.7458100558659218
Random Forest Accuracy after PCA: 0.846927374301676
Naïve Bayes Accuracy after PCA: 0.8430167597765363
```

7. Applying DBSCAN and Hierarchical Clustering

```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_clustering)

dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_clusters = dbscan.fit_predict(X_scaled)
df['DBSCAN_Cluster'] = dbscan_clusters

Z = linkage(X_scaled, method='ward')
hierarchical_clusters = fcluster(Z, 4, criterion='maxclust')
df['Hierarchical_Cluster'] = hierarchical_clusters

dbscan_unique_clusters = len(set(dbscan_clusters))
hierarchical_unique_clusters = len(set(hierarchical_clusters))

print(f"DBSCAN clusters: {dbscan_unique_clusters}, Hierarchical clusters: {hierarchical_unique_clusters}")
```

```
DBSCAN clusters: 35, Hierarchical clusters: 4
```

8. Optimizing the algorithms to yield better clusters

```
from sklearn.metrics import silhouette_score

# KMeans optimization using Silhouette Score
silhouette_scores = []
for n_clusters in range(2, 10):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = kmeans.fit_predict(X_scaled)
    silhouette_avg = silhouette_score(X_scaled, cluster_labels)
    silhouette_scores.append((n_clusters, silhouette_avg))

optimal_clusters = max(silhouette_scores, key=lambda x: x[1])
print(f"Optimal number of clusters: {optimal_clusters[0]} with Silhouette Score: {optimal_clusters[1]:.2f}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_ini
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_ini
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_ini
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_ini
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_ini
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_ini
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_ini
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_ini
  warnings.warn(
Optimal number of clusters: 3 with Silhouette Score: 0.27
```

9. Applying Deep Learning models

```
from keras.models import Sequential
from keras.layers import Dense

X_deep = X_clustering.values
y_deep = df['TENURE'].values

from sklearn.model_selection import train_test_split
X_train_deep, X_test_deep, y_train_deep, y_test_deep = train_test_split(X_deep, y_deep, test_size=0.2, random_state=42

scaler = StandardScaler()
X_train_deep_scaled = scaler.fit_transform(X_train_deep)
X_test_deep_scaled = scaler.transform(X_test_deep)

model1 = Sequential()
model1.add(Dense(64, input_dim=X_train_deep_scaled.shape[1], activation='relu'))
model1.add(Dense(32, activation='relu'))
model1.add(Dense(1, activation='linear'))
model1.compile(optimizer='adam', loss='mse', metrics=['mae'])
model1.fit(X_train_deep_scaled, y_train_deep, epochs=50, batch_size=32, validation_split=0.2)

model2 = Sequential()
model2.add(Dense(128, input_dim=X_train_deep_scaled.shape[1], activation='relu'))
model2.add(Dense(64, activation='relu'))
model2.add(Dense(32, activation='relu'))
model2.add(Dense(1, activation='linear'))
model2.compile(optimizer='adam', loss='mse', metrics=['mae'])
model2.fit(X_train_deep_scaled, y_train_deep, epochs=50, batch_size=32, validation_split=0.2)
```

```
Epoch 23/50
179/179 [==============================] - 1s 3ms/step - loss: 0.8939 - mae: 0.5900 - val_loss: 1.1205 - val_mae
Epoch 24/50
179/179 [==============================] - 1s 4ms/step - loss: 0.9124 - mae: 0.6110 - val_loss: 1.1880 - val_mae
Epoch 25/50
179/179 [==============================] - 1s 4ms/step - loss: 0.9090 - mae: 0.6164 - val_loss: 0.9757 - val_mae
Epoch 26/50
179/179 [==============================] - 1s 4ms/step - loss: 0.8297 - mae: 0.5589 - val_loss: 1.0368 - val_mae
Epoch 27/50
179/179 [==============================] - 1s 3ms/step - loss: 0.8434 - mae: 0.5783 - val_loss: 0.9873 - val_mae
Epoch 28/50
179/179 [==============================] - 1s 3ms/step - loss: 0.8375 - mae: 0.5709 - val_loss: 1.0062 - val_mae
Epoch 29/50
179/179 [==============================] - 1s 3ms/step - loss: 0.8179 - mae: 0.5632 - val_loss: 0.9851 - val_mae
Epoch 30/50
179/179 [==============================] - 1s 3ms/step - loss: 0.7958 - mae: 0.5557 - val_loss: 1.0267 - val_mae
Epoch 31/50
179/179 [==============================] - 1s 4ms/step - loss: 0.8216 - mae: 0.5676 - val_loss: 0.9702 - val_mae
Epoch 32/50
179/179 [==============================] - 1s 5ms/step - loss: 0.7655 - mae: 0.5424 - val_loss: 0.9731 - val_mae
Epoch 33/50
179/179 [==============================] - 1s 5ms/step - loss: 0.8216 - mae: 0.5743 - val_loss: 0.9854 - val_mae
Epoch 34/50
179/179 [==============================] - 1s 5ms/step - loss: 0.7430 - mae: 0.5293 - val_loss: 1.0024 - val_mae
Epoch 35/50
179/179 [==============================] - 1s 4ms/step - loss: 0.8002 - mae: 0.5648 - val_loss: 1.1383 - val_mae
Epoch 36/50
179/179 [==============================] - 1s 3ms/step - loss: 0.7618 - mae: 0.5418 - val_loss: 0.9704 - val_mae
Epoch 37/50
179/179 [==============================] - 1s 3ms/step - loss: 0.7636 - mae: 0.5419 - val_loss: 1.0304 - val_mae
Epoch 38/50
179/179 [==============================] - 1s 3ms/step - loss: 0.7466 - mae: 0.5365 - val_loss: 1.0392 - val_mae
Epoch 39/50
179/179 [==============================] - 1s 4ms/step - loss: 0.7010 - mae: 0.5135 - val_loss: 0.9657 - val_mae
Epoch 40/50
179/179 [==============================] - 1s 4ms/step - loss: 0.7540 - mae: 0.5481 - val_loss: 1.0382 - val_mae
Epoch 41/50
179/179 [==============================] - 1s 3ms/step - loss: 0.7194 - mae: 0.5270 - val_loss: 0.9432 - val_mae
Epoch 42/50
179/179 [==============================] - 1s 3ms/step - loss: 0.6984 - mae: 0.5110 - val_loss: 1.0116 - val_mae
Epoch 43/50
179/179 [==============================] - 1s 4ms/step - loss: 0.7140 - mae: 0.5251 - val_loss: 0.9657 - val_mae
Epoch 44/50
179/179 [==============================] - 1s 4ms/step - loss: 0.6808 - mae: 0.5075 - val_loss: 0.9725 - val_mae
Epoch 45/50
179/179 [==============================] - 1s 4ms/step - loss: 0.6830 - mae: 0.5022 - val_loss: 1.0582 - val_mae
Epoch 46/50
179/179 [==============================] - 1s 4ms/step - loss: 0.7208 - mae: 0.5352 - val_loss: 1.1139 - val_mae
Epoch 47/50
179/179 [==============================] - 1s 4ms/step - loss: 0.6480 - mae: 0.4867 - val_loss: 1.0876 - val_mae
Epoch 48/50
179/179 [==============================] - 1s 4ms/step - loss: 0.6794 - mae: 0.5063 - val_loss: 1.0066 - val_mae
Epoch 49/50
179/179 [==============================] - 1s 4ms/step - loss: 0.6523 - mae: 0.4910 - val_loss: 0.9832 - val_mae
Epoch 50/50
179/179 [==============================] - 1s 4ms/step - loss: 0.6439 - mae: 0.4832 - val_loss: 1.0005 - val_mae
<keras.src.callbacks.History at 0x7cfce81c57e0>
```

10. Optimizing the deep learning model

```
from keras.callbacks import EarlyStopping, ReduceLROnPlateau

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5)

model1.fit(X_train_deep_scaled, y_train_deep, epochs=100, batch_size=32, validation_split=0.2, callbacks=[early_stoppi
model2.fit(X_train_deep_scaled, y_train_deep, epochs=100, batch_size=32, validation_split=0.2, callbacks=[early_stoppi
```

```
Epoch 28/100
179/179 [==============================] - 1s 3ms/step - loss: 0.7721 - mae: 0.5117 - val_loss: 0.9286 - val_mae
Epoch 1/100
179/179 [==============================] - 1s 4ms/step - loss: 0.6512 - mae: 0.4963 - val_loss: 1.0045 - val_mae
Epoch 2/100
179/179 [==============================] - 1s 5ms/step - loss: 0.6383 - mae: 0.4863 - val_loss: 1.0212 - val_mae
Epoch 3/100
179/179 [==============================] - 1s 5ms/step - loss: 0.6485 - mae: 0.4936 - val_loss: 0.9862 - val_mae
Epoch 4/100
179/179 [==============================] - 1s 5ms/step - loss: 0.6379 - mae: 0.4859 - val_loss: 1.2059 - val_mae
Epoch 5/100
179/179 [==============================] - 1s 3ms/step - loss: 0.6679 - mae: 0.5146 - val_loss: 1.0610 - val_mae
Epoch 6/100
179/179 [==============================] - 1s 3ms/step - loss: 0.6547 - mae: 0.5135 - val_loss: 1.2213 - val_mae
Epoch 7/100
179/179 [==============================] - 1s 3ms/step - loss: 0.6526 - mae: 0.4996 - val_loss: 1.0999 - val_mae
Epoch 8/100
179/179 [==============================] - 1s 3ms/step - loss: 0.6393 - mae: 0.4942 - val_loss: 1.0972 - val_mae
Epoch 9/100
179/179 [==============================] - 1s 4ms/step - loss: 0.5169 - mae: 0.4130 - val_loss: 0.9445 - val_mae
Epoch 10/100
179/179 [==============================] - 1s 4ms/step - loss: 0.4912 - mae: 0.3934 - val_loss: 0.9221 - val_mae
Epoch 11/100
179/179 [==============================] - 1s 3ms/step - loss: 0.4822 - mae: 0.3887 - val_loss: 0.9268 - val_mae
Epoch 12/100
179/179 [==============================] - 1s 4ms/step - loss: 0.4820 - mae: 0.3898 - val_loss: 0.9357 - val_mae
Epoch 13/100
179/179 [==============================] - 1s 3ms/step - loss: 0.4770 - mae: 0.3874 - val_loss: 0.9579 - val_mae
Epoch 14/100
179/179 [==============================] - 1s 3ms/step - loss: 0.4754 - mae: 0.3856 - val_loss: 0.9505 - val_mae
Epoch 15/100
179/179 [==============================] - 1s 3ms/step - loss: 0.4742 - mae: 0.3872 - val_loss: 0.9340 - val_mae
Epoch 16/100
179/179 [==============================] - 1s 3ms/step - loss: 0.4531 - mae: 0.3740 - val_loss: 0.9209 - val_mae
Epoch 17/100
179/179 [==============================] - 1s 4ms/step - loss: 0.4515 - mae: 0.3714 - val_loss: 0.9181 - val_mae
Epoch 18/100
179/179 [==============================] - 1s 4ms/step - loss: 0.4521 - mae: 0.3717 - val_loss: 0.9189 - val_mae
Epoch 19/100
179/179 [==============================] - 1s 4ms/step - loss: 0.4496 - mae: 0.3701 - val_loss: 0.9211 - val_mae
Epoch 20/100
179/179 [==============================] - 1s 4ms/step - loss: 0.4489 - mae: 0.3696 - val_loss: 0.9203 - val_mae
Epoch 21/100
179/179 [==============================] - 1s 5ms/step - loss: 0.4468 - mae: 0.3709 - val_loss: 0.9229 - val_mae
Epoch 22/100
179/179 [==============================] - 1s 5ms/step - loss: 0.4476 - mae: 0.3698 - val_loss: 0.9190 - val_mae
Epoch 23/100
179/179 [==============================] - 1s 5ms/step - loss: 0.4424 - mae: 0.3693 - val_loss: 0.9198 - val_mae
Epoch 24/100
179/179 [==============================] - 1s 4ms/step - loss: 0.4419 - mae: 0.3655 - val_loss: 0.9194 - val_mae
Epoch 25/100
179/179 [==============================] - 1s 4ms/step - loss: 0.4420 - mae: 0.3665 - val_loss: 0.9193 - val_mae
Epoch 26/100
179/179 [==============================] - 1s 4ms/step - loss: 0.4415 - mae: 0.3663 - val_loss: 0.9197 - val_mae
Epoch 27/100
179/179 [==============================] - 1s 3ms/step - loss: 0.4418 - mae: 0.3683 - val_loss: 0.9210 - val_mae
<keras.src.callbacks.History at 0x7cfc9b964f40>
```

```
# Check for non-numeric columns in X_set_b
non_numeric_columns_b = X_set_b.select_dtypes(include=['object']).columns
print("Non-numeric columns in X_set_b:", non_numeric_columns_b)

# Check for non-numeric columns in X_set_a
non_numeric_columns = X_set_a.select_dtypes(include=['object']).columns
print("Non-numeric columns in X_set_a:", non_numeric_columns)
```

```
Non-numeric columns in X_set_b: Index(['Cluster Name_A'], dtype='object')
Non-numeric columns in X_set_a: Index([], dtype='object')
```

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

dataset_path = '/content/Customer Data.csv'
df = pd.read_csv(dataset_path)

df = df.drop(['CUST_ID'], axis=1)

df.fillna(df.mean(), inplace=True)

set_a, set_b = train_test_split(df, test_size=0.5, random_state=42)

X_set_a = set_a.drop(['TENURE'], axis=1)

non_numeric_columns_a = X_set_a.select_dtypes(include=['object']).columns
print("Non-numeric columns in X_set_a:", non_numeric_columns_a)

X_set_a = X_set_a.drop(non_numeric_columns_a, axis=1)

kmeans = KMeans(n_clusters=4, random_state=42)
clusters_a = kmeans.fit_predict(X_set_a)
set_a['Cluster_A'] = clusters_a

X_set_b = set_b.drop(['TENURE'], axis=1)

non_numeric_columns_b = X_set_b.select_dtypes(include=['object']).columns
print("Non-numeric columns in X_set_b:", non_numeric_columns_b)

X_set_b = X_set_b.drop(non_numeric_columns_b, axis=1)

set_b['Cluster_A'] = kmeans.predict(X_set_b)

cluster_names_a = {0: 'Segment 1', 1: 'Segment 2', 2: 'Segment 3', 3: 'Segment 4'}
set_a['Cluster Name_A'] = set_a['Cluster_A'].map(cluster_names_a)
set_b['Cluster Name_A'] = set_b['Cluster_A'].map(cluster_names_a)

X_set_b = set_b.drop(['Cluster_A', 'Cluster Name_A'], axis=1)
y_set_b = set_b['Cluster_A']

non_numeric_columns_b = X_set_b.select_dtypes(include=['object']).columns
print("Non-numeric columns in X_set_b:", non_numeric_columns_b)


X_set_b = X_set_b.drop(non_numeric_columns_b, axis=1)
X_train_b, X_test_b, y_train_b, y_test_b = train_test_split(X_set_b, y_set_b, test_size=0.2, random_state=42)

scaler_b = StandardScaler()
X_train_b_scaled = scaler_b.fit_transform(X_train_b)
X_test_b_scaled = scaler_b.transform(X_test_b)
```

```
X_test_b_scaled = scaler_b.transform(X_test_b)

rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train_b_scaled, y_train_b)
y_pred_b = rf_classifier.predict(X_test_b_scaled)
print("Random Forest Classification Accuracy on Set B:", accuracy_score(y_test_b, y_pred_b))
```

```
Non-numeric columns in X_set_a: Index([], dtype='object')
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_ini
  warnings.warn(
Non-numeric columns in X_set_b: Index([], dtype='object')
Non-numeric columns in X_set_b: Index([], dtype='object')
Random Forest Classification Accuracy on Set B: 0.9843575418994414
```

Start coding or generate with AI.