# Credit card dataset

## Group 6: wanderers

\* Dre

\* Ian

\* Abdul

\* Carlo

### Import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import random

import warnings

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score, davies\_bouldin\_score, calinski\_harabasz\_score

### Set initial states

rng = 0

np.random.seed(rng)

random.seed(rng)

warnings.filterwarnings("ignore")

%matplotlib inline

### Load dataset

df = pd.read\_csv("cc\_dirty.csv")

### Examine datasest

df.info()

df.head(5)

# print unique values of object columns

for i in df.columns:

if df[i].dtype == "object":

print(df[i].name)

print(df[i].unique())

print("\n")

### Clean data

# clean gender column

def gender\_reencode(x):

if x=="M":

return "Male"

elif x=="Male":

return "Male"

elif x=="F":

return "Female"

elif x=="Female":

return "Female"

else:

return "Error"

df["gender"] = df["gender"].apply(gender\_reencode)

df["gender"].unique()

# clean city\_pop column

df["city\_pop"] = df["city\_pop"].str.replace(",", "")

df["city\_pop"] = df["city\_pop"].str.replace("people", "")

df["city\_pop"] = df["city\_pop"].str.strip()

df["city\_pop"]

df["city\_pop"] = df["city\_pop"].astype(int)

df["city\_pop"]

# clean date of birth column

df["dob"] = pd.to\_datetime(df["dob"], format="%d/%m/%Y")

df["dob"]

# clean unix\_time column

df["unix\_time"] = pd.to\_datetime(df["unix\_time"], unit="s")

df["unix\_time"]

# clean amt column

df["amt"] = df["amt"].str.replace("$", "")

df["amt"] = df["amt"].str.strip()

df["amt"]

df["amt"] = df["amt"].astype(float)

df["amt"]

df.head(5)

df.info()

### Feature engineering

# age from dob

df["age"] = df["dob"].apply(lambda x : (pd.Timestamp.now().year - x.year))

df["age"]

# transaction year, month, day of month, and hour from unix\_time

df["trans\_yr"] = df["unix\_time"].astype(str).apply(lambda x: x[0:4])

df["trans\_yr"] = df["trans\_yr"].astype(int)

df["trans\_mo"] = df["unix\_time"].astype(str).apply(lambda x: x[5:7])

df["trans\_mo"] = df["trans\_mo"].astype(int)

df["trans\_day"] = df["unix\_time"].astype(str).apply(lambda x: x[8:10])

df["trans\_day"] = df["trans\_day"].astype(int)

df["trans\_hr"] = df["unix\_time"].astype(str).apply(lambda x: x[11:13])

df["trans\_hr"] = df["trans\_hr"].astype(int)

# age bracket from age

def age\_bracket(x):

if x < 60:

return "59 and below"

elif x >= 60 and x < 70:

return "60 to 69"

elif x >= 70 and x < 80:

return "70 to 79"

elif x >= 80:

return "80 and above"

else:

return "Error"

df["age\_bracket"] = df["age"].apply(age\_bracket)

df.head(5)

df.info()

# region from city

ref = pd.read\_csv("city-region.csv")

ref.head(5)

df = df.merge(ref, on='city', how='left')

df.head(5)

df.info()

df["region"].head(5)

len(df["acct\_num"].unique())

### RFM

# Convert unix\_time to datetime format

df['unix\_time'] = pd.to\_datetime(df['unix\_time'])

# Calculate the most recent transaction date in the entire dataset

most\_recent\_date = df['unix\_time'].max()

# Calculate RFM metrics for each customer

rfm\_df = df.groupby('acct\_num').agg({

'unix\_time': lambda x: (most\_recent\_date - x.max()).days, # Recency: Number of days since last purchase

'acct\_num': 'count', # Frequency: Number of transactions

'amt': 'sum' # Monetary: Total amount spent

}).rename(columns={

'unix\_time': 'Recency',

'acct\_num': 'Frequency',

'amt': 'Monetary'

})

rfm\_df.head(5)

def r\_score(x):

if x <= rfm\_df['Recency'].quantile(0.25):

return 4

elif x <= rfm\_df['Recency'].quantile(0.50):

return 3

elif x <= rfm\_df['Recency'].quantile(0.75):

return 2

else:

return 1

def fm\_score(x, column):

if x <= rfm\_df[column].quantile(0.25):

return 1

elif x <= rfm\_df[column].quantile(0.50):

return 2

elif x <= rfm\_df[column].quantile(0.75):

return 3

else:

return 4

rfm\_df['R\_Score'] = rfm\_df['Recency'].apply(r\_score)

rfm\_df['F\_Score'] = rfm\_df['Frequency'].apply(fm\_score, column='Frequency')

rfm\_df['M\_Score'] = rfm\_df['Monetary'].apply(fm\_score, column='Monetary')

rfm\_df['RFM\_Score'] = rfm\_df['R\_Score'] + rfm\_df['F\_Score'] + rfm\_df['M\_Score']

rfm\_df.head(5)

### Clustering

df\_for\_clustering = rfm\_df.drop(columns=["RFM\_Score"])

num\_features = df\_for\_clustering.columns.values.tolist()

preprocessor = ColumnTransformer(

transformers=[

('num', StandardScaler(), num\_features)

])

pipeline = Pipeline([

('preprocessor', preprocessor),

('clusterer', KMeans(n\_clusters=5, random\_state=42))

])

pipeline.fit(df\_for\_clustering)

cluster\_labels = pipeline.predict(df\_for\_clustering)

rfm\_df['Cluster'] = cluster\_labels

rfm\_df.head()

rfm\_df['Cluster'].value\_counts()

# Preprocess the data using the preprocessor defined earlier

X\_preprocessed = preprocessor.fit\_transform(df\_for\_clustering)

# Define the range for k

k\_values = range(2, 15)

inertia\_values = []

silhouette\_scores = []

davies\_bouldin\_scores = []

calinski\_harabasz\_scores = []

# Loop through different values of k

for k in k\_values:

kmeans = KMeans(n\_clusters=k, random\_state=42)

cluster\_labels = kmeans.fit\_predict(X\_preprocessed)

inertia\_values.append(kmeans.inertia\_)

silhouette\_avg = silhouette\_score(X\_preprocessed, cluster\_labels) \* 100

silhouette\_scores.append(silhouette\_avg)

db\_score = davies\_bouldin\_score(X\_preprocessed, cluster\_labels) \* 100

davies\_bouldin\_scores.append(db\_score)

ch\_score = calinski\_harabasz\_score(X\_preprocessed, cluster\_labels)

calinski\_harabasz\_scores.append(ch\_score)

print(f"Number of Clusters: {k}, Inertia: {kmeans.inertia\_}, Silhouette Score: {silhouette\_avg}, Davies-Bouldin Score: {db\_score}, Calinski-Harabasz Score: {ch\_score}")

# Plot metrics

plt.figure(figsize=(16, 9))

plt.plot(k\_values, inertia\_values, marker='o', linestyle='--', label='Inertia')

plt.plot(k\_values, silhouette\_scores, marker='x', linestyle='-', label='Silhouette Score')

plt.plot(k\_values, davies\_bouldin\_scores, marker='+', linestyle='-.', label='Davies-Bouldin Score')

plt.plot(k\_values, calinski\_harabasz\_scores, marker='\*', linestyle=':', label='Calinski-Harabasz Score')

plt.title('Cluster Evaluation Metrics for Optimal k')

plt.xlabel('Number of Clusters')

plt.ylabel('Value')

plt.legend()

plt.grid(True)

plt.show()

optimal\_k = 3

kmeans\_optimal = KMeans(n\_clusters=optimal\_k, random\_state=42)

kmeans\_optimal.fit(X\_preprocessed)

optimal\_cluster\_labels = kmeans\_optimal.predict(X\_preprocessed)

rfm\_df['Optimal\_Cluster'] = optimal\_cluster\_labels

rfm\_df.to\_csv("rfm\_clustered.csv")

rfm\_df["Optimal\_Cluster"].value\_counts()

cluster\_characteristics = rfm\_df.groupby('Optimal\_Cluster').agg({

'Recency': 'mean',

'Frequency': 'mean',

'Monetary': ['mean', 'count']

}).round(1)

cluster\_characteristics

plt.figure(figsize=(16, 9))

sns.boxplot(y=rfm\_df["Recency"], x=rfm\_df["Optimal\_Cluster"])

plt.show()

plt.figure(figsize=(16, 9))

sns.boxplot(y=rfm\_df["Frequency"], x=rfm\_df["Optimal\_Cluster"])

plt.show()

plt.figure(figsize=(16, 9))

sns.boxplot(y=rfm\_df["Monetary"], x=rfm\_df["Optimal\_Cluster"])

plt.show()

rfm\_df.head(10)

plt.figure(figsize=(16, 9))

opt\_cluster\_count = rfm\_df["Optimal\_Cluster"].value\_counts()

opt\_cluster\_count

r\_ct = pd.crosstab(rfm\_df["R\_Score"], rfm\_df["Optimal\_Cluster"])

r\_ct

r\_ct = pd.crosstab(rfm\_df["F\_Score"], rfm\_df["Optimal\_Cluster"])

r\_ct

r\_ct = pd.crosstab(rfm\_df["M\_Score"], rfm\_df["Optimal\_Cluster"])

r\_ct

plt.figure(figsize=(16, 9))

sns.barplot(y=opt\_cluster\_count.values, x=opt\_cluster\_count.index)

plt.show()

df = df = df.merge(rfm\_df["Optimal\_Cluster"], left\_on='acct\_num', right\_index=True, how='left')

df[["acct\_num", "Optimal\_Cluster"]].head(10)

df[["acct\_num", "Optimal\_Cluster"]].isna().sum()

print(df.head(5))

### Visualizations

#### 1. Categories

acct\_join = pd.merge(df, rfm\_df, on='acct\_num', how='left')

columns\_to\_drop = ['cc\_num', 'gender', 'city', 'city\_pop', 'job', 'dob', 'acct\_num2', 'trans\_num', 'unix\_time', 'age', 'trans\_yr', 'trans\_mo', 'trans\_day', 'trans\_hr', 'age\_bracket']

acct\_join = acct\_join.drop(columns=columns\_to\_drop)

import seaborn as sns

import matplotlib.pyplot as plt

# Calculate the count of each category

category\_counts = acct\_join['category'].value\_counts()

# Sort categories by count in descending order

sorted\_categories = category\_counts.index

# Define a custom color palette

custom\_palette = ['green' if category in sorted\_categories[:5] else 'gray' for category in sorted\_categories]

# Create the count plot with the custom color palette

sns.catplot(data=acct\_join,

y='category',

kind='count',

palette=custom\_palette, # Use the custom color palette

order=sorted\_categories, # Sort by category counts

height=6,

aspect=1.5

)

plt.xlabel('Count')

plt.ylabel('Category')

plt.title('Count of Categories')

plt.show()

acct\_join.head(5)

import seaborn as sns

import matplotlib.pyplot as plt

# Get unique categories excluding null values

categories = acct\_join['Optimal\_Cluster\_y'].unique()

# Create separate count plots for each category

for category in categories:

plt.figure(figsize=(8, 4))

subset = acct\_join[acct\_join['Optimal\_Cluster\_y'] == category]

# Calculate the sum of 'amt' for each category and sort by sum

category\_amts = subset.groupby('category')['amt'].sum().reset\_index()

category\_amts = category\_amts.sort\_values(by='amt', ascending=False)

# Define a custom color palette

custom\_palette = ['green' if i < 3 else 'gray' for i in range(len(category\_amts))]

sns.barplot(data=category\_amts, y='category', x='amt', palette=custom\_palette)

plt.xlabel('Total Amount')

plt.ylabel('Category')

plt.title(f'Sum of Amount in Category for Cluster: {category}')

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Calculate the sum of 'amt' for each category

category\_sums = acct\_join.groupby('category')['amt'].sum().reset\_index()

category\_sums = category\_sums.sort\_values(by='amt', ascending=False)

# Define a custom color palette

n\_colors = len(category\_sums)

custom\_palette = ['green' if i < 5 else 'gray' for i in range(n\_colors)]

# Create the bar plot with sorted categories and custom colors

plt.figure(figsize=(12, 6))

sns.barplot(data=category\_sums, x='category', y='amt', palette=custom\_palette)

plt.xlabel('Category')

plt.ylabel('Total Amount')

plt.title('Total Amount by Category')

plt.xticks(rotation=30)

plt.show()

#### Gender

sns.catplot(data=df,

x='gender',

kind='count')

plt.xlabel('Gender')

plt.ylabel('Count')

plt.title('Gender Count')

plt.show()

sns.barplot(data=df,

x='gender',

y="amt")

plt.xlabel('Gender')

plt.ylabel('Average Amount')

plt.title('Gender vs Average Amount Spend')

plt.show()

gender\_amount\_cluster\_df = df.groupby(['gender', 'Optimal\_Cluster']).agg({

'amt': 'sum'

}).reset\_index()

gender\_amount\_cluster\_df = gender\_amount\_cluster\_df.sort\_values(by="gender", ascending=False)

gender\_amount\_cluster\_df

sns.barplot(data=df,

x='Optimal\_Cluster',

y="amt",

hue='gender')

plt.xlabel('Cluster')

plt.ylabel('Average Amount')

plt.title('Cluster vs Average Amount Spend for Gender')

plt.legend(title='Gender', bbox\_to\_anchor=(1.05, 1), loc='upper left')

plt.show()

sns.scatterplot(data=gender\_amount\_cluster\_df,

x='gender',

y='amt',

hue='Optimal\_Cluster',

sizes='Optimal\_Cluster',

palette='viridis')

# Add labels and title

plt.xlabel('Gender')

plt.ylabel('Amount Spend')

plt.title('Scatterplot of Gender vs. Amount Spend for Cluster')

# Show the plot

plt.legend(title='Cluster', bbox\_to\_anchor=(1.05, 1), loc='upper left')

plt.show()

sns.boxplot(

data=df,

x="gender",

y="amt",

sym="")

plt.xlabel('Gender')

plt.ylabel('Average Amount Spend')

plt.title('Boxplot of Gender vs. Average Amount Spend')

plt.show()

sns.boxplot(

data=df[df['Optimal\_Cluster']==0],

x="gender",

y="amt",

sym="")

plt.xlabel('Gender')

plt.ylabel('Average Amount Spend')

plt.title('Boxplot of Gender vs. Average Amount Spend for Cluster 0')

plt.show()

sns.boxplot(

data=df[df['Optimal\_Cluster']==1],

x="gender",

y="amt",

sym="")

plt.xlabel('Gender')

plt.ylabel('Average Amount Spend')

plt.title('Boxplot of Gender vs. Average Amount Spend for Cluster 1')

plt.show()

sns.boxplot(

data=df[df['Optimal\_Cluster']==2],

x="gender",

y="amt",

sym="")

plt.xlabel('Gender')

plt.ylabel('Average Amount Spend')

plt.title('Boxplot of Gender vs. Average Amount Spend for Cluster 2')

plt.show()

#### 3. Time-based analysis

year\_filter = df[["trans\_yr", "amt", "Optimal\_Cluster"]]

year\_filter

year\_filter["Year-Cluster"] = year\_filter["trans\_yr"].astype(str) + "-" + year\_filter["Optimal\_Cluster"].astype(str)

year\_filter = year\_filter.sort\_values("Year-Cluster")

year\_filter

plt.figure(figsize=(16, 9))

sns.boxplot(x=year\_filter["Year-Cluster"], y=year\_filter["amt"])

plt.show()

year\_cluster\_count = year\_filter.groupby("Year-Cluster").count()

year\_cluster\_count

time\_summ = df.groupby(["trans\_mo", "Optimal\_Cluster"])["Optimal\_Cluster"].count()

time\_summ

time\_summ = pd.DataFrame(time\_summ)

time\_summ

temp = time\_summ["Optimal\_Cluster"].values

time\_summ["trans\_mo"] = [x for x in time\_summ.index.get\_level\_values(0)]

time\_summ["Optimal\_Cluster"] = [x for x in time\_summ.index.get\_level\_values(1)]

time\_summ["Count"] = temp

time\_summ = time\_summ.reset\_index(drop=True)

time\_summ.head(6)

plt.figure(figsize=(14, 9))

sns.lineplot(time\_summ, x="trans\_mo", y="Count", hue="Optimal\_Cluster")

plt.title("Transaction counts trend per month")

plt.xlabel("Month")

plt.ylabel("Transactions")

plt.show()

time\_summ = df.groupby(["trans\_mo", "Optimal\_Cluster"])["amt"].sum()

time\_summ = pd.DataFrame(time\_summ)

time\_summ

temp = time\_summ["amt"].values

time\_summ["trans\_mo"] = [x for x in time\_summ.index.get\_level\_values(0)]

time\_summ["Optimal\_Cluster"] = [x for x in time\_summ.index.get\_level\_values(1)]

time\_summ["amt"] = temp

time\_summ = time\_summ.reset\_index(drop=True)

time\_summ.head(6)

plt.figure(figsize=(14, 9))

sns.lineplot(time\_summ, x="trans\_mo", y="amt", hue="Optimal\_Cluster")

plt.title("Total amount trend per month")

plt.xlabel("Month")

plt.ylabel("Amount")

plt.show()

# drill down to June, July, August

df\_6\_7\_8 = df[df["trans\_mo"].isin([6, 7, 8])]

df\_6\_7\_8

df\_6\_7\_8.columns

df\_6\_7\_8\_ct = pd.crosstab(df\_6\_7\_8["category"], df\_6\_7\_8["Optimal\_Cluster"])

df\_6\_7\_8\_ct.style.background\_gradient(cmap ='RdYlBu').set\_properties(\*\*{'font-size': '20px'})

#### Region

# Make a new data frame that contains only the cluster, amt and region

pie\_df=df.groupby(['region','Optimal\_Cluster']).size().unstack()

print(pie\_df)

# Get the top 5 regions by count

top\_regions = df['region'].value\_counts().index[:5]

# Create a custom palette for all unique regions

unique\_regions = df['region'].unique()

# Sort unique regions based on their frequency (count)

unique\_regions = sorted(unique\_regions, key=lambda x: top\_regions.get\_loc(x) if x in top\_regions else len(top\_regions))

# Create a custom palette with blue for the top 5 regions and grey for the rest

custom\_palette\_loc = ['blue' if region in top\_regions else 'grey' for region in unique\_regions]

# Create a count plot of 'region' with the custom palette

plt.figure(figsize=(10, 6))

sns.countplot(data=df, x='region', palette=custom\_palette\_loc, order=unique\_regions)

# Set plot labels and title

plt.xlabel('Region')

plt.ylabel('Count')

plt.title('Cardholders: Regional Distribution (Sorted)')

# Display the plot

plt.grid(False)

plt.xticks(rotation=45)

plt.show()

# Get unique optimal clusters using number of columns

unique\_clusters = pie\_df.columns

# Create a 1x3 grid of subplots for the pie charts

fig, axes = plt.subplots(1, 3, figsize=(15, 5)) # 1 row, 3 columns

# Define a color map where percentages < 10% are grey

colors = plt.cm.tab20c(range(20)) # Choose a color map with 20 distinct colors

colors[10:] = (0.8, 0.8, 0.8, 1.0) # Set colors beyond the 10th index to grey

# Create and display a pie chart for each cluster in a subplot

for i, cluster in enumerate(unique\_clusters):

cluster\_data = pie\_df[cluster].dropna() # Drop NaN values

# Sort the data in descending orderf

cluster\_data = cluster\_data.sort\_values(ascending=False)

labels = cluster\_data.index

sizes = cluster\_data.values

# Calculate the percentages

percentages = sizes / sizes.sum() \* 100

# Create a list of colors for pie slices based on percentages

slice\_colors = [colors[i] if percentage >= 10 else colors[-1] for i, percentage in enumerate(percentages)]

# Create a pie chart in the current subplot with custom colors

axes[i].pie(sizes, labels=labels, colors=slice\_colors, autopct='%1.1f%%', startangle=90, counterclock=False)

axes[i].set\_title(f'Optimal Cluster {cluster} - Region Distribution')

axes[i].axis('equal') # Equal aspect ratio ensures a circular pie chart

# Adjust layout spacing

plt.tight\_layout()

# Display the subplots

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Sample DataFrame (replace with your actual DataFrame)

# df\_2 = ...

# Create a barplot sorted by 'city\_pop' in descending order

sns.barplot(data=df,

x='region',

y='city\_pop',

order=df.groupby('region')['city\_pop'].median().sort\_values(ascending=False).index, ci=None)

# Set plot labels and title

plt.xlabel('Region')

plt.ylabel('City Population')

plt.title('Barplot of City Population by Region (Sorted by City Population)')

# Display the plot

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Filter the DataFrame for Cluster 1 and Cluster 2

cluster1\_df = df[df['Optimal\_Cluster'] == 0]

cluster2\_df = df[df['Optimal\_Cluster'] == 1]

# Calculate mean spending per category within each cluster

cluster1\_means = cluster1\_df.groupby(['Optimal\_Cluster', 'category'])['amt'].mean().reset\_index()

cluster2\_means = cluster2\_df.groupby(['Optimal\_Cluster', 'category'])['amt'].mean().reset\_index()

# Sort the categories by mean spending in descending order for both clusters

cluster1\_means = cluster1\_means.sort\_values(by=['Optimal\_Cluster', 'amt'], ascending=[True, False])

cluster2\_means = cluster2\_means.sort\_values(by=['Optimal\_Cluster', 'amt'], ascending=[True, False])

# Create subplots for Cluster 1 and Cluster 2

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 8))

# Plot Cluster 1

sns.barplot(

data=cluster1\_means,

x='Optimal\_Cluster',

y='amt',

hue='category',

ci=None,

palette='tab10',

ax=axes[0]

)

axes[0].set\_xlabel('Cluster 0')

axes[0].set\_ylabel('Amount Spent')

axes[0].set\_title('Cluster 0 Spending per Category')

axes[0].get\_legend().remove()

# Plot Cluster 2

sns.barplot(

data=cluster2\_means,

x='Optimal\_Cluster',

y='amt',

hue='category',

ci=None,

palette='tab10',

ax=axes[1]

)

axes[1].set\_xlabel('Cluster 1')

axes[1].set\_ylabel('Amount Spent')

axes[1].set\_title('Cluster 1 Spending per Category')

axes[1].legend(loc='upper right', bbox\_to\_anchor=(1.0, 1.0))

plt.tight\_layout()

plt.show()

# Plot Cluster 3

cluster3\_df=df[df['Optimal\_Cluster']==2]

cluster3\_means=cluster3\_df.groupby(['Optimal\_Cluster','category'])['amt'].mean().reset\_index()

cluster3\_means=cluster3\_means.sort\_values(by=['Optimal\_Cluster','amt'], ascending=[True,False])

sns.barplot(

data=cluster3\_means,

x='Optimal\_Cluster',

y='amt',

hue='category',

ci=None,

palette='tab10',

)

plt.xlabel('Cluster')

plt.ylabel('Average Spending')

plt.title('Category Spending of Cluster 2')

plt.legend(loc='upper right', bbox\_to\_anchor=(1.35, 1.0))

print(cluster1\_df['acct\_num'].value\_counts())