MMA 869: Individual Assignment

- [Muhammad Farrukh, Khan]
- [20200898]
- [2]
- [Game of Thrones]
- [14/08/2021]

Assignment Instructions

This assignment contains four questions. The questions are fully contained in this Google Colab Notebook.

You are to make a copy of this Notebook and edit the copy to provide your answers. You are to complete the assignment entirely within Google Colab. Why?

- It gives you practice using cloud-based interactive notebook environments (which is a popular workflow)
- It is easier for you to manage the environment (e.g., installing packages, etc.)
- Google Colab has nice, beefy machines, so you don't have to worry about running out of memory on your local computer.
- It will be easier for the TA to help you debug your code if you need help
- It will be easier for the TA to mark/run your code

Some parts of this assigment require you to write code. Use Python. You may use standard Python libraries, including `scikit-learn`, `pandas`, `numpy`, and scipy`.

Some parts of this assignment require text responses. In these cases, type your response in the Notebook cell indicated. Use English. Use proper grammar, spelling, and punctuation. Be professional and clear. Be complete, but not overly-verbose. Feel free to use Markdown syntax to format your answer (i.e., add bold, italics, lists, tables).

What to Submit to the Course Portal

- Export your completed Notebook as a PDF file by clicking File->Print->Save as PDF.
- Please do not submit the Notebook file (`.ipynb`) to the course portal.
- Please submit the PDF export of the Notebook.
    - Please name the PDF file `2022_869_FirstnameLastName.pdf`
        - E.g., `2022_869_StephenThomas.pdf`
    - Please make sure you have run all the cells so we can see the output!
    - Best practice: Before exporting to PDF click Runtime->Restart and run all.

Preliminaries: Inspect and Set up environment

No action is required on your part in this section. These cells print out helpful information about the environment, just in case.

```
import datetime
import pandas as pd
import numpy as np
```

```
print(datetime.datetime.now())
```

```
2021-08-15 23:43:06.989761
```

```
!which python
```

```
/usr/local/bin/python
```

```
!python --version
```

```
Python 3.7.11
```

```
!echo $PYTHONPATH
```

```
/env/python
```

```
# TODO: install any packages you need to here. For example:
#pip install unidecode
```

Question 1: Uncle Steve's Diamonds

Instructions

You work at a local jewelry store named *Uncle Steve's Diamonds*. You started as a janitor, but you've recently been promoted to senior data analyst! Congratulations.

Uncle Steve, the store's owner, needs to better understand the store's customers. In particular, he wants to know what kind of customers shop at the store. He wants to know the main types of *customer personas*. Once he knows these, he will contemplate ways to better market to each persona, better satisfy each persona, better cater to each persona, increase the loyalty of each persona, etc. But first, he must know the personas.

You want to help Uncle Steve. Using sneaky magic (and the help of Environics), you've collected four useful features for a subset of the customers: age, income, spending score (i.e., a score based on how much they've spent at the store in total), and savings (i.e., how much money they have in their personal bank account).

Your tasks

1. Pick a clustering algorithm (the `sklearn.cluster` module has many good choices, including `KMeans`, `DBSCAN`, and `AgglomerativeClustering` (aka Hierarchical). (Note that another popular implementation of the hierarchical algorithm can be found in SciPy's `scipy.cluster.hierarchy.linkage`.) Don't spend a lot of time thinking about which algorithm to choose - just pick one. Cluster the customers as best as you can, within reason. That is, try different feature preprocessing steps, hyperparameter values, and/or distance metrics. You don't need to try every single posssible combination, but try a few at least. Measure how good each model configuration is by calculating an internal validation metric (e.g., `calinski_harabasz_score` or `silhouette_score`).
2. You have some doubts - you're not sure if the algorithm you chose in part 1 is the best algorithm for this dataset/problem. Neither is Uncle Steve. So, choose a different algorithm (any!) and do it all again.
3. Which clustering algorithm is "better" in this case? Think about charateristics of the algorithm like quality of results, ease of use, speed, interpretability, etc. Choose a winner and justify to Uncle Steve.
4. Interpret the clusters of the winning model. That is, describe, in words, a *persona* that accurately depicts each cluster. Use statistics (e.g., cluster means/distributions), examples (e.g., exemplar instances from each cluster), and/or visualizations (e.g., relative importance plots, snakeplots) to get started. Human judgement and creativity will be necessary. This is where it all comes together. Be descripive and *help Uncle Steve understand his customers better*. Please!

Marking

The coding parts (i.e., 1 and 2) will be marked based on:

- *Correctness*. Code clearly and fully performs the task specified.
- *Reproducibility*. Code is fully reproducible. I.e., you (and I) are able to run this Notebook again and again, from top to bottom, and get the same results each time.
- *Style*. Code is organized. All parts commented with clear reasoning and rationale. No old code laying around. Code easy to follow.

Parts 3 and 4 will be marked on:

- *Quality*. Response is well-justified and convincing. Responses uses facts and data where possible.
- *Style*. Response uses proper grammar, spelling, and punctuation. Response is clear and professional. Response is complete, but not overly-verbose. Response follows length guidelines.

Tips

- Since clustering is an unsupervised ML technique, you don't need to split the data into training/validation/test or anything like that. Phew!
- On the flip side, since clustering is unsupervised, you will never know the "true" clusters, and so you will never know if a given algorithm is "correct." There really is no notion of "correctness" - only "usefullness."
- Many online clustering tutorials (including some from Uncle Steve) create flashy visualizations of the clusters by plotting the instances on a 2-D graph and coloring each point by the cluster ID. This is really nice and all, but it can only work if your dataset only has exactly two features - no more, no less. This dataset has more than two features, so you cannot use this technique. (But that's OK - you don't need to use this technique.)
- Must you use all four features in the clustering? Not necessarily, no. But "throwing away" quality data, for no reason, is unlikely to improve a model.
- Some people have success applying a dimensionality reduction technique (like `sklearn.decomposition.PCA`) to the features before clustering. You may do this if you wish, although it may not be as helpful in this case because there are only four features to begin with.
- If you apply a transformation (e.g., `MinMaxScaler` or `StandardScaler`) to the features before clustering, you may have difficulty interpretting the means of the clusters (e.g., what is a mean Age of 0.2234??). There are two options to fix this: first, you can always reverse a transformation with the `inverse_transform` method. Second, you can just use the original dataset (i.e., before any prepropoceesing) during the interpreation step.
- You cannot change the distance metric for K-Means. (This is for theoretical reasons: K-Means only works/makes sense with Euclidean distance.

```
#Importing Libraries

#Data Manipulation
import numpy as np # recall that "np" etc. -- are abbreviated names we gave to these packages for notational convenience
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc, roc_auc_score, classification_report, confusion_matrix, make_scorer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

import warnings
warnings.filterwarnings("ignore")

import seaborn as sns

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.metrics import f1_score

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

import itertools

import scipy
```

```
from sklearn.preprocessing import MaxAbsScaler
```

```
from sklearn.preprocessing import OrdinalEncoder
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder, FunctionTransformer
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.decomposition import KernelPCA, PCA, TruncatedSVD
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
```

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity= "all"
```

1.0: Load data

```
# DO NOT MODIFY THIS CELL
df1 = pd.read_csv("https://drive.google.com/uc?export=download&id=1thHDCwQK3GijytoSSZNekAsItN_FGHtm")
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 505 entries, 0 to 504
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Age            505 non-null    int64  
 1   Income         505 non-null    int64  
 2   SpendingScore  505 non-null    float64
 3   Savings        505 non-null    float64
dtypes: float64(2), int64(2)
memory usage: 15.9 KB
```

1.1: Clustering Algorithm #1

EDA

```
df1.head()

df1.tail()
```

	Age	Income	SpendingScore	Savings
0	58	77769	0.791329	6559.829923
1	59	81799	0.791082	5417.661426
2	62	74751	0.702657	9258.992965
3	59	74373	0.765680	7346.334504
4	87	17760	0.348778	16869.507130

	Age	Income	SpendingScore	Savings
500	28	101206	0.387441	14936.775389
501	93	19934	0.203140	17969.693769
502	90	35297	0.355149	16091.401954
503	91	20681	0.354679	18401.088445
504	89	30267	0.289310	14386.351880

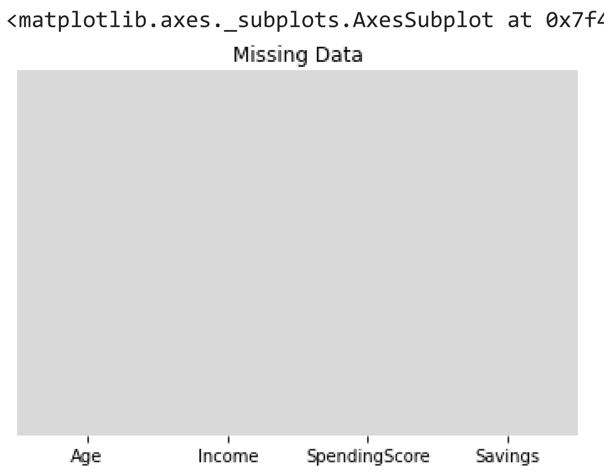
```
df1.describe().transpose()

df1.shape
```

	count	mean	std	min	25%	50%	75%	max
Age	505.0	59.019802	24.140043	17.0	34.000000	59.000000	85.000000	97.0
Income	505.0	75513.291089	35992.922184	12000.0	34529.000000	75078.000000	107100.000000	142000.0
SpendingScore	505.0	0.505083	0.259634	0.0	0.304792	0.368215	0.768279	1.0
Savings	505.0	11862.455867	4949.229253	0.0	6828.709702	14209.932802	16047.268331	20000.0

```
(505, 4)
```

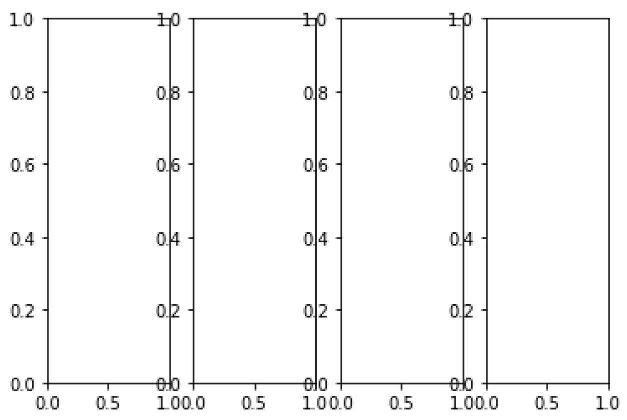
```
#No Missing values found
sns.heatmap(df1.isnull(),yticklabels = False, cbar = False,cmap = 'tab20c_r')
plt.title('Missing Data')
plt.show()
```

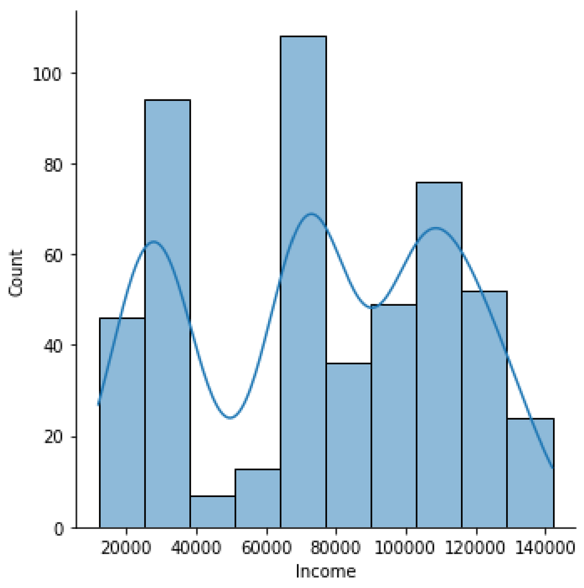
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4869231590>Text(0.5, 1.0, 'Missing Data')
```

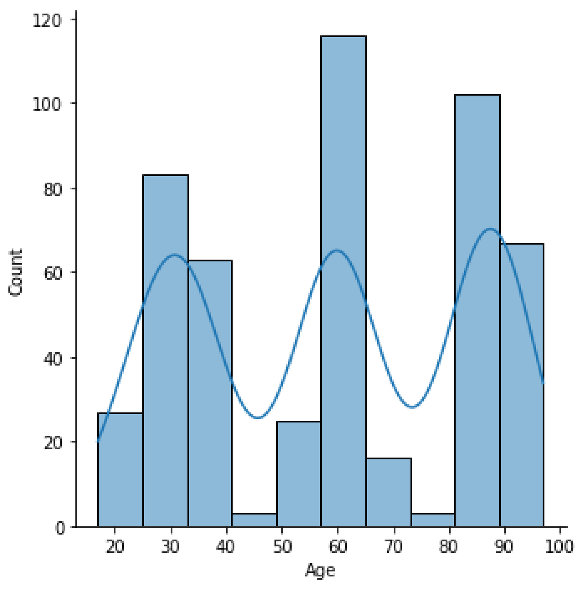


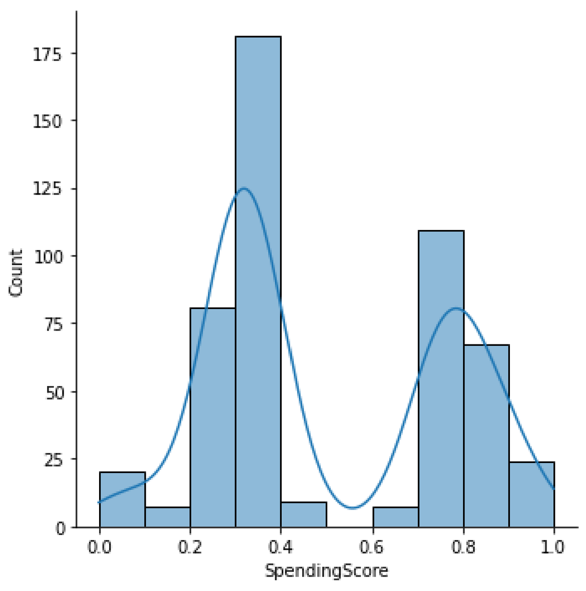

```
f, axes= plt.subplots(ncols=4)

sns.displot(data=df1, x="Income", kde=True, ax=axes[0])
sns.displot(data=df1, x="Age", kde=True, ax=axes[1])
sns.displot(data=df1, x="SpendingScore", kde=True, ax=axes[2])
sns.displot(data=df1, x="Savings", kde=True, ax=axes[3])
```

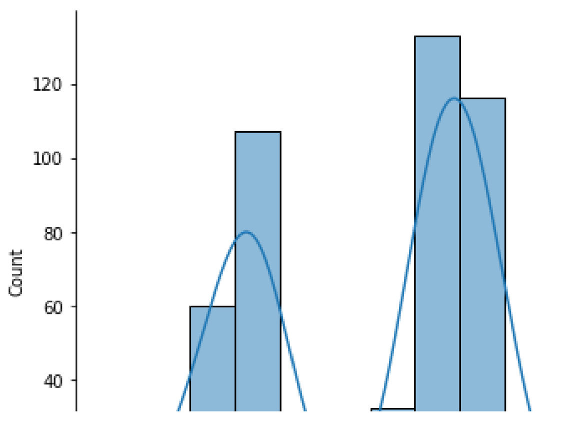
```
<seaborn.axisgrid.FacetGrid at 0x7f485ffe0d10><seaborn.axisgrid.FacetGrid at 0x7f4868a17210><seaborn.axisgrid.FacetGrid at 0x7f485cc352d0><seaborn.axisgrid.FacetGrid at 0x7f485cb1cf50>
```







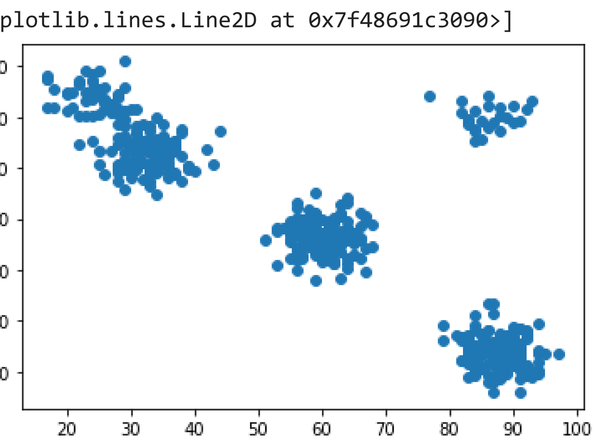




Need to scale the features

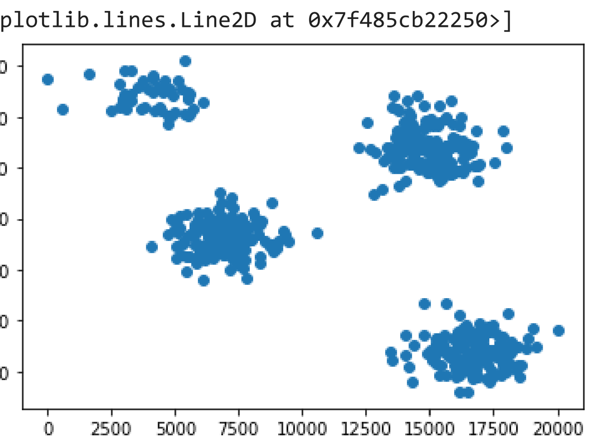
```
plt.plot(df1['Age'],df1['Income'],'o')
plt.set_xlabel='Age'
plt.set_ylabel="Income"
plt.show()
```

```
[<matplotlib.lines.Line2D at 0x7f48691c3090>]
```



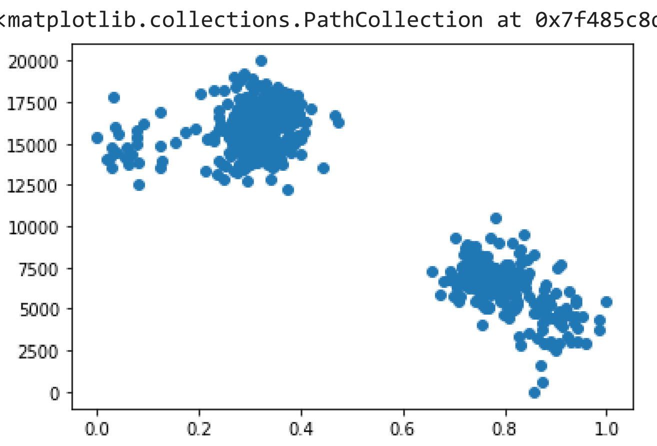
```
plt.plot(df1['Savings'],df1['Income'],'o')
plt.set_xlabel='Savings'
plt.set_ylabel="Income"
plt.show()

#SHOWS NEGATIVE CORRELATION BETWEEN INCOME AND SAVINGS
```

```
[<matplotlib.lines.Line2D at 0x7f485cb22250>]
```



```
plt.scatter(df1['SpendingScore'], df1['Savings'], marker='o')
```

```
<matplotlib.collections.PathCollection at 0x7f485c8d6310>
```



```
messi= df1.corr()

sns.heatmap(messi)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f485c89f290>
```

Negative correlations of Age & Income

```
# Basic Model without tuning & preprocessing

from sklearn.cluster import  KMeans

k_means = KMeans(init="k-means++", n_clusters=4, n_init=10, random_state=42)
k_means.fit(df1)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

```
k_means.inertia_ #POOR RESULTS without preprocessing data
```

```
21452909425.051315
```

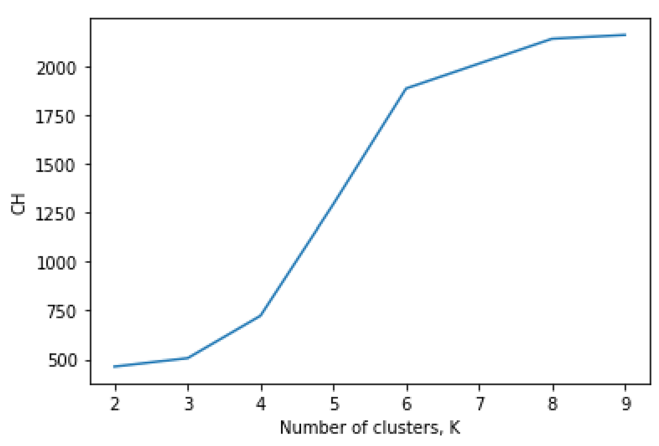
Creating Clusters using all features & Log transformation

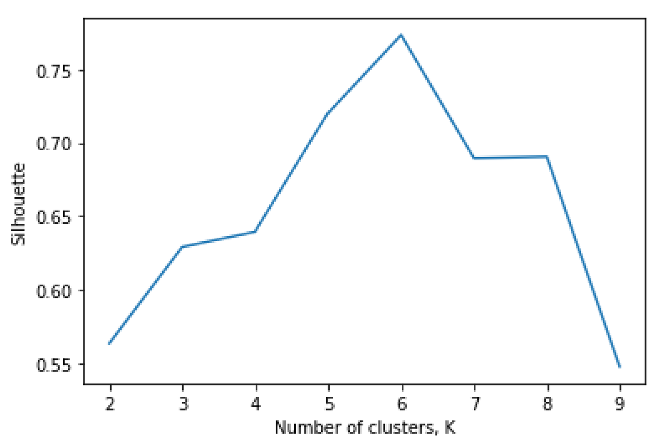
```
# Preprocessing Data

df_scaled=df1.copy()

scaler = preprocessing.FunctionTransformer(np.log1p, validate=True).fit(df_scaled[['Age']])
df_scaled['Age'] = scaler.transform(df_scaled[['Age']])

scaler = preprocessing.FunctionTransformer(np.log1p, validate=True).fit(df_scaled[['Income']])
df_scaled['Income'] = scaler.transform(df_scaled[['Income']])

scaler = preprocessing.FunctionTransformer(np.log1p, validate=True).fit(df_scaled[['SpendingScore']])
df_scaled['SpendingScore'] = scaler.transform(df_scaled[['SpendingScore']])

scaler = preprocessing.FunctionTransformer(np.log1p, validate=True).fit(df_scaled[['Savings']])
df_scaled['Savings'] = scaler.transform(df_scaled[['Savings']])
```

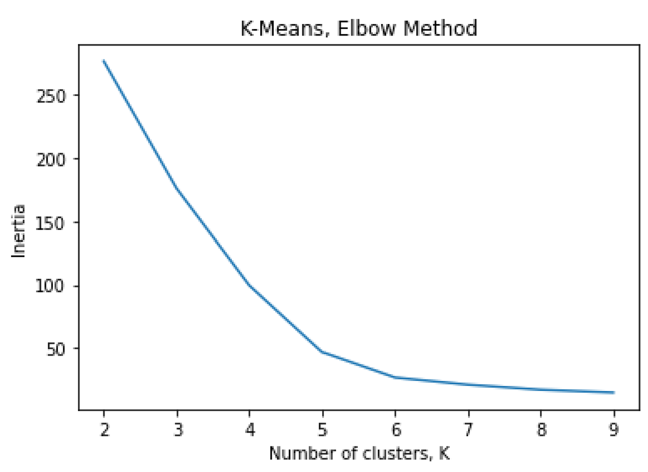
Hyper-Tuning optimal K

```
from sklearn.metrics import silhouette_score, silhouette_samples, calinski_harabasz_score
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
```

```
from sklearn.cluster import KMeans
chs = {}
silhouettes = {}
inertias = {}
for k in range(2, 10):
    k_means = KMeans(init='k-means++', n_clusters=k, n_init=10, random_state=42)
    k_means.fit(df_scaled)
    inertias[k] = k_means.inertia_
    sil = silhouette_score(df_scaled, k_means.labels_,  metric='euclidean')
    ch = calinski_harabasz_score(df_scaled, k_means.labels_)
    chs[k] = ch
    silhouettes[k] = sil


plt.figure();
plt.plot(list(chs.keys()), list(chs.values()));
#plt.title('K-Means')
plt.xlabel("Number of clusters, K");
plt.ylabel("CH");
plt.show();


plt.figure();
plt.plot(list(silhouettes.keys()), list(silhouettes.values()));
#plt.title('K-Means, Elbow Method')
plt.xlabel("Number of clusters, K");
plt.ylabel("Silhouette");
plt.show();

plt.figure();
plt.plot(list(inertias.keys()), list(inertias.values()));
plt.title('K-Means, Elbow Method')
plt.xlabel("Number of clusters, K");
plt.ylabel("Inertia");
```







- While using Log scaling on all the features...
- calinski_harabasz_score shows that clusters of 9 are best (2000+)
- silhouette_score shows that cluster of 6 is good (0.75)

```
#Modeling using 6 clusters...

k_means6 = KMeans(init="k-means++", n_clusters=6, n_init=10, random_state=42)
k_means6.fit(df_scaled)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=6, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

```
print(silhouette_score(df_scaled, k_means6.labels_))
print(calinski_harabasz_score(df_scaled, k_means6.labels_))
print(k_means6.inertia_)
```

```
0.7734759854051435
1886.3140696657933
26.673943592895718
```

Using 9 clusters since it improved CH score..

```
#Modeling using 9 clusters...

k_means9 = KMeans(init="k-means++", n_clusters=9, n_init=10, random_state=42)
k_means9.fit(df_scaled)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=9, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

```
print(silhouette_score(df_scaled, k_means9.labels_))
print(calinski_harabasz_score(df_scaled, k_means9.labels_))
print(k_means9.inertia_)
```

```
0.5475930825648443
2160.957336918562
14.805444029684285
```

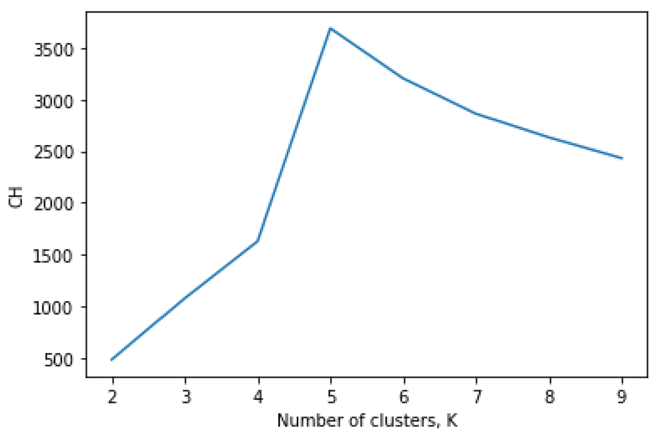
Low Silhouette score when inertia & CH improved significantly

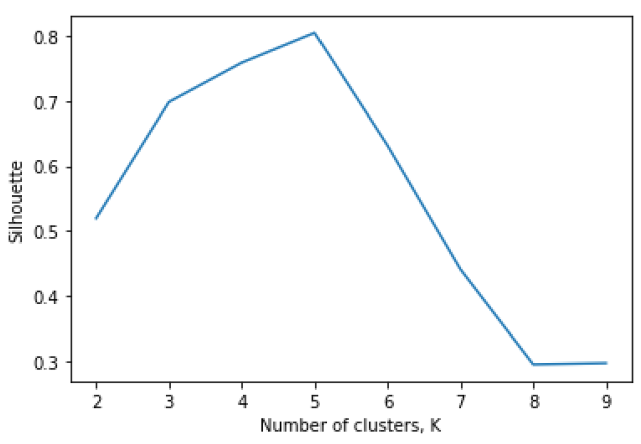
- With All features & log preprocessing, k=6 seems to be the best with Euclidean Distance...

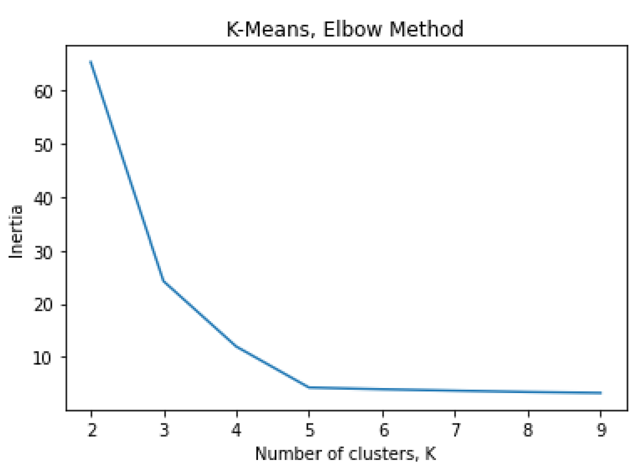
All Features Clustering Using MaxAbsScaler

```
Z = df1.copy()
scaler = MaxAbsScaler()
Z = scaler.fit_transform(Z)
```

```
from sklearn.cluster import KMeans
chs = {}
silhouettes = {}
inertias = {}
for k in range(2, 10):
    k_means = KMeans(init='k-means++', n_clusters=k, n_init=10, random_state=42)
    k_means.fit(Z)
    inertias[k] = k_means.inertia_
    sil = silhouette_score(Z, k_means.labels_,  metric='euclidean')
    ch = calinski_harabasz_score(Z, k_means.labels_)
    chs[k] = ch
    silhouettes[k] = sil


plt.figure();
plt.plot(list(chs.keys()), list(chs.values()));
#plt.title('K-Means')
plt.xlabel("Number of clusters, K");
plt.ylabel("CH");
plt.show();


plt.figure();
plt.plot(list(silhouettes.keys()), list(silhouettes.values()));
#plt.title('K-Means, Elbow Method')
plt.xlabel("Number of clusters, K");
plt.ylabel("Silhouette");
plt.show();

plt.figure();
plt.plot(list(inertias.keys()), list(inertias.values()));
plt.title('K-Means, Elbow Method')
plt.xlabel("Number of clusters, K");
plt.ylabel("Inertia");
```








```
#Using 5 Clusters as shown by the elbow tuning above to be the optimal one...

k_means_messi = KMeans(init="k-means++", n_clusters=5, n_init=10, random_state=42)
k_means_messi.fit(Z)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

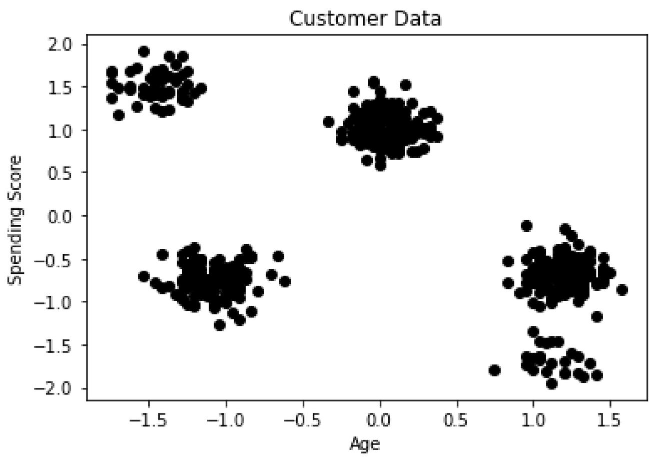
Clustering by Age & Spending Score with Standard Scaler

```
#Using Age and Spending Score to understand the customers, how they spend and what's their age.

X = df1.copy()
X = X.drop(['Income', 'Savings'], axis=1)
X.head(10)
```

	Age	SpendingScore
0	58	0.791329
1	59	0.791082
2	62	0.702657
3	59	0.765680
4	87	0.348778
5	29	0.847034
6	54	0.785198
7	87	0.355290
8	83	0.324719
9	84	0.367063

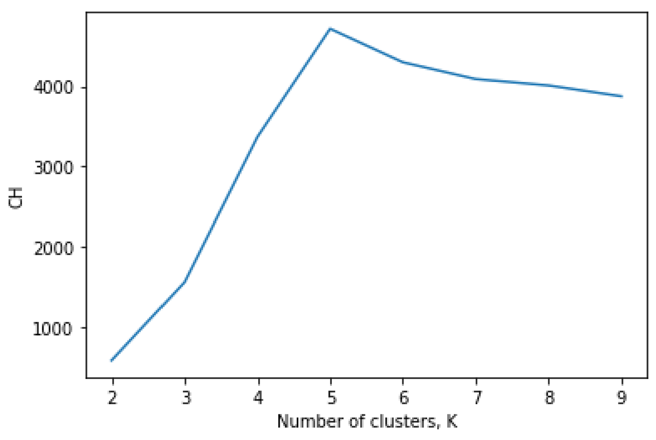
```
#Normalizing Data

scaler = StandardScaler()
features = ['Age', 'SpendingScore']
X[features] = scaler.fit_transform(X[features])
```

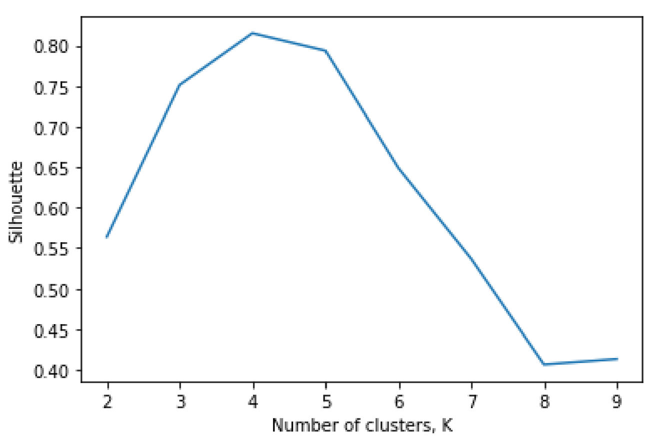
```
#Plotting
plt.figure();

plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c="black");
plt.title("Customer Data");
plt.xlabel('Age');
plt.ylabel('Spending Score');
plt.xticks();
plt.yticks();
```

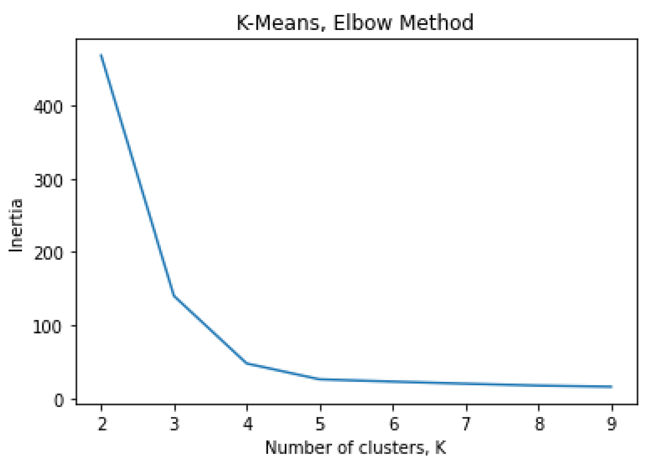


HyperTuning of Age & Spending Score clustering

```
from sklearn.cluster import KMeans
chs = {}
silhouettes = {}
inertias = {}
for k in range(2, 10):
    k_means = KMeans(init='k-means++', n_clusters=k, n_init=10, random_state=42)
    k_means.fit(X)
    inertias[k] = k_means.inertia_
    sil = silhouette_score(X, k_means.labels_,  metric='euclidean')
    ch = calinski_harabasz_score(X, k_means.labels_)
    chs[k] = ch
    silhouettes[k] = sil


plt.figure();
plt.plot(list(chs.keys()), list(chs.values()));
#plt.title('K-Means')
plt.xlabel("Number of clusters, K");
plt.ylabel("CH");
plt.show();


plt.figure();
plt.plot(list(silhouettes.keys()), list(silhouettes.values()));
#plt.title('K-Means, Elbow Method')
plt.xlabel("Number of clusters, K");
plt.ylabel("Silhouette");
plt.show();

plt.figure();
plt.plot(list(inertias.keys()), list(inertias.values()));
plt.title('K-Means, Elbow Method')
plt.xlabel("Number of clusters, K");
plt.ylabel("Inertia");
```








```
#Using 4 clusters as the optimal value from the Elbow Method above
k_means4 = KMeans(init="k-means++", n_clusters=4, n_init=10, random_state=42)
k_means4.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

Clustering Using Income & Spending Score with MaxAbsScaler

```
Y = df1.copy()
Y = Y.drop(['Age', 'Savings'], axis=1)
Y.head(10)
```

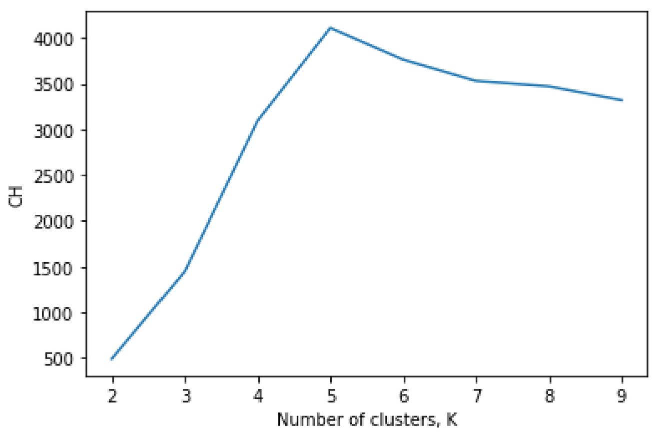
	Income	SpendingScore
0	77769	0.791329
1	81799	0.791082
2	74751	0.702657
3	74373	0.765680
4	17760	0.348778
5	131578	0.847034
6	76500	0.785198
7	42592	0.355290
8	34384	0.324719
9	27693	0.367063

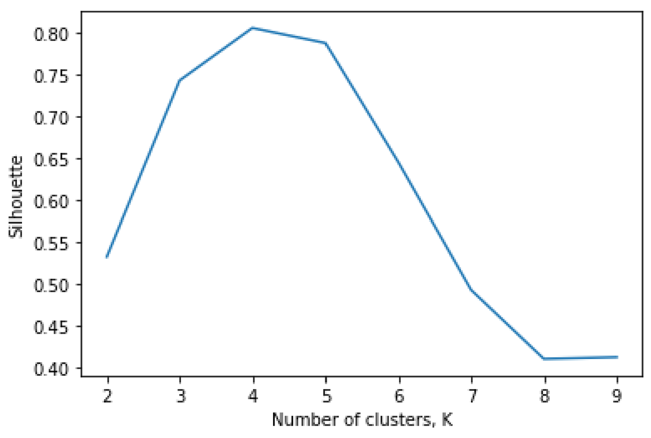
```
scaler = MaxAbsScaler()
features = ['Income', 'SpendingScore']
Y[features] = scaler.fit_transform(Y[features])

Y.describe()
```

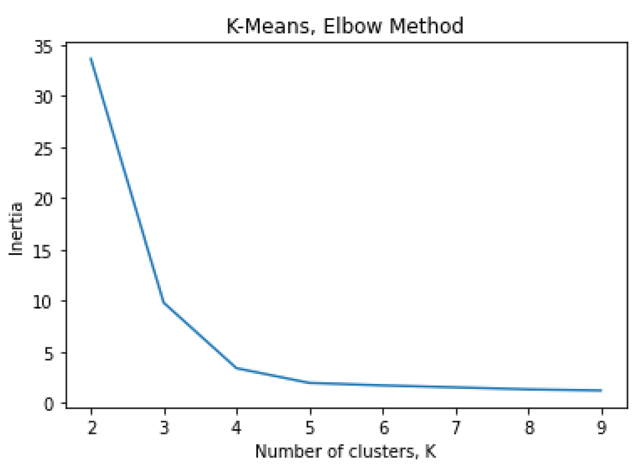
	Income	SpendingScore
count	505.000000	505.000000
mean	0.531784	0.505083
std	0.253471	0.259634
min	0.084507	0.000000
25%	0.243162	0.304792
50%	0.528718	0.368215
75%	0.754225	0.768279
max	1.000000	1.000000

```
from sklearn.cluster import KMeans
chs = {}
silhouettes = {}
inertias = {}
for k in range(2, 10):
    k_means = KMeans(init='k-means++', n_clusters=k, n_init=10, random_state=42)
```

```
    k_means.fit(Y)
    inertias[k] = k_means.inertia_
    sil = silhouette_score(Y, k_means.labels_,  metric='euclidean')
    ch = calinski_harabasz_score(Y, k_means.labels_)
    chs[k] = ch
    silhouettes[k] = sil


plt.figure();
plt.plot(list(chs.keys()), list(chs.values()));
#plt.title('K-Means')
plt.xlabel("Number of clusters, K");
plt.ylabel("CH");
plt.show();


plt.figure();
plt.plot(list(silhouettes.keys()), list(silhouettes.values()));
#plt.title('K-Means, Elbow Method')
plt.xlabel("Number of clusters, K");
plt.ylabel("Silhouette");
plt.show();

plt.figure();
plt.plot(list(inertias.keys()), list(inertias.values()));
plt.title('K-Means, Elbow Method')
plt.xlabel("Number of clusters, K");
plt.ylabel("Inertia");
```








```
#Using 5 clusters as the optimal for Income & Spending Score clustering
k_means5_ISS = KMeans(init="k-means++", n_clusters=5, n_init=10, random_state=42)
k_means5_ISS.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

Evaluation of different models & variations

Scores using optimal K-means clustering numbers derived from Elbow Method

```
print('SIL Score with 9 Clusters (All Features):',silhouette_score(df_scaled, k_means9.labels_))
print('CH Score with 9 Clusters (All Features):',calinski_harabasz_score(df_scaled, k_means9.labels_))
print('Inertia with 5 Clusters (All Feaures):',k_means9.inertia_)
```

```
SIL Score with 9 Clusters (All Features): 0.5475930825648443
CH Score with 9 Clusters (All Features): 2160.957336918562
Inertia with 5 Clusters (All Feaures): 14.805444029684285
```

```
print('SIL Score with 5 Clusters (All Features):',silhouette_score(Z, k_means_messi.labels_)) #Highest Sillhoutte Score
print('CH Score with 5 Clusters (All Features):',calinski_harabasz_score(Z, k_means_messi.labels_)) #Highest CH Score
print('Inertia with 5 Clusters (All Features):',k_means_messi.inertia_) #Lowest Inertia Value
```

```
SIL Score with 5 Clusters (All Features): 0.8048297446929015
CH Score with 5 Clusters (All Features): 3688.532109627289
Inertia with 5 Clusters (All Features): 4.209805843772563
```

```
print('SIL Score with 5 Clusters (Age & Spending Score):',silhouette_score(X, k_means9.labels_))
print('CH Score with 5 Clusters (Age & Spending Score):',calinski_harabasz_score(X, k_means9.labels_))
print('Inertia with 5 Clusters (Age & Spending Score):',k_means9.inertia_)
```

```
SIL Score with 5 Clusters (Age & Spending Score): 0.2288598083360596
CH Score with 5 Clusters (Age & Spending Score): 2358.5609971327053
Inertia with 5 Clusters (Age & Spending Score): 14.805444029684285
```

```
#Best K-Means Model Using Age & Spending Score...

print('SIL Score with 4 Clusters (Age & Spending Score):',silhouette_score(X, k_means4.labels_))
print('CH Score with 4 Clusters (Age & Spending Score):',calinski_harabasz_score(X, k_means4.labels_))
print('Inertia with 4 Clusters (Age & Spending Score):',k_means4.inertia_)
```

```
SIL Score with 4 Clusters (Age & Spending Score): 0.8151026432983561
CH Score with 4 Clusters (Age & Spending Score): 3367.2216515349933
Inertia with 4 Clusters (Age & Spending Score): 47.72479392364724
```

```
print('SIL Score with 6 Clusters (All Features):',silhouette_score(df_scaled, k_means6.labels_))
print('CH Score with 6 Clusters (All Features):',calinski_harabasz_score(df_scaled, k_means6.labels_))
print('Inertia with 6 Clusters (All Feaures):',k_means6.inertia_)
```

```
SIL Score with 6 Clusters (All Features): 0.7734759854051435
CH Score with 6 Clusters (All Features): 1886.3140696657933
Inertia with 6 Clusters (All Feaures): 26.673943592895718
```

```
print('SIL Score with 5 Clusters (Income & Spending Score):',silhouette_score(Y, k_means5_ISS.labels_))
print('CH Score with 5 Clusters (Income & Spending Score):',calinski_harabasz_score(Y, k_means5_ISS.labels_))
print('Inertia with 5 Clusters (Income & Spending Score):',k_means5_ISS.inertia_)
```

```
SIL Score with 5 Clusters (Income & Spending Score): 0.7881880875606122
CH Score with 5 Clusters (Income & Spending Score): 4110.634381763206
Inertia with 5 Clusters (Income & Spending Score): 26.07005589499711
```

1.2: Clustering Algorithm #2

```
#DBSCAN, radius is eps(small values in smaller circles), minpts specify the minimum number of points that need to be in the circle for it to be dense..
#higher values require denser clusters...
#minpts = NumFeatures*2
```

DBSCAN Clustering on All features & Log transformed

```
df1 = pd.read_csv("https://drive.google.com/uc?export=download&id=1thHDCwQK3GijytoSSZNekAsItN_FGHtm")
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 505 entries, 0 to 504
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Age            505 non-null    int64
```

```
 1   Income         505 non-null    int64
 2   SpendingScore  505 non-null    float64
 3   Savings        505 non-null    float64
dtypes: float64(2), int64(2)
memory usage: 15.9 KB
```

```
# Preprocessing Data

df_scaled=df1.copy()

scaler = preprocessing.FunctionTransformer(np.log1p, validate=True).fit(df_scaled[['Age']])
df_scaled['Age'] = scaler.transform(df_scaled[['Age']])

scaler = preprocessing.FunctionTransformer(np.log1p, validate=True).fit(df_scaled[['Income']])
df_scaled['Income'] = scaler.transform(df_scaled[['Income']])

scaler = preprocessing.FunctionTransformer(np.log1p, validate=True).fit(df_scaled[['SpendingScore']])
df_scaled['SpendingScore'] = scaler.transform(df_scaled[['SpendingScore']])

scaler = preprocessing.FunctionTransformer(np.log1p, validate=True).fit(df_scaled[['Savings']])
df_scaled['Savings'] = scaler.transform(df_scaled[['Savings']])
```

```
db1 = DBSCAN(eps=0.3, min_samples=8)
db1.fit(df_scaled)
```

```
DBSCAN(algorithm='auto', eps=0.3, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=8, n_jobs=None, p=None)
```

```
db1.labels_
```

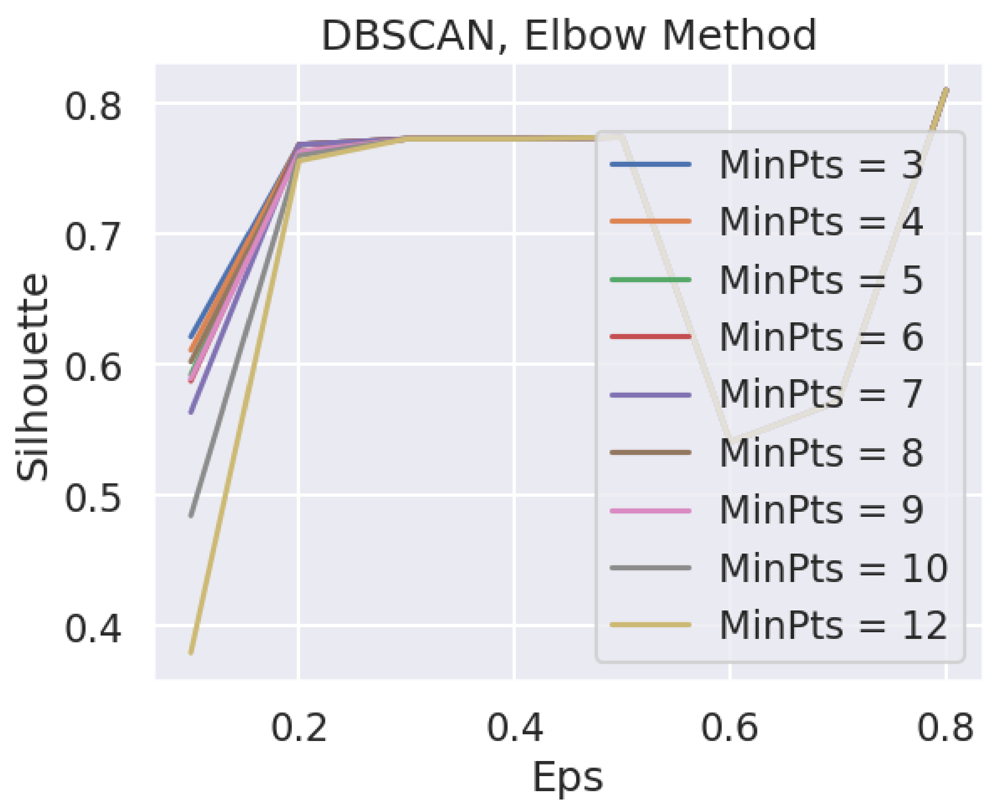
```
array([ 0,  0,  0,  0,  1,  2,  0,  1,  1,  1,  3,  4,  4,  4,  0,  3,  0,
        1,  1,  3,  4,  3,  0,  1,  3,  2,  2,  0,  1,  0,  0,  0,  1,  0,
        4,  0,  1,  0,  4,  4,  0,  1,  2,  0,  0,  2,  3,  2,  4,  0,  1,
        4,  2,  1,  0,  1,  4,  1,  1,  1,  1,  1,  1,  0,  0,  0,  1,  2,
        2,  0,  0,  4,  1,  1,  4,  0,  4,  0,  0,  2,  0,  3,  0,  1,  1,
        4,  4,  0,  0,  0,  0,  0, -1,  1,  0,  2,  4,  0,  0,  4,  4,  2,
        4,  2,  4,  1,  1,  4,  4,  0,  1,  4,  0,  4,  4,  4,  0,  4,  2,
        1,  4,  1,  1,  1,  4,  3,  0,  1,  0,  0,  4,  0,  4,  0,  2,  1,
        3,  2,  0,  4,  1,  0,  0,  2,  0,  2,  0,  3,  0,  1,  0,  4,  1,
        0,  1,  3,  4,  0,  1,  0,  4,  1,  0,  4,  1,  4,  1,  0,  4,  0,
        1,  0,  4,  0,  1,  0,  0,  1,  1,  1,  0,  0,  4,  2,  2,  1,  4,
        0,  0,  0,  4,  1,  0,  0,  0,  1, -1,  4,  1,  1,  1,  4,  0,  4,
        0,  4,  0,  1,  2,  0,  1,  1,  1,  1,  0,  1,  4,  2,  4,  1,  0,
        0,  4,  1,  4,  2,  4,  0,  0,  1,  0,  0,  1,  0,  0,  0,  4,  0,
        4,  4,  2,  0,  0,  0,  3,  0,  4,  1,  2,  2,  2,  1,  0,  4,  1,
        1,  4,  4,  0,  0,  3,  0,  0,  1,  0,  1,  4,  2,  0,  4,  1,  0,
        1,  1,  4,  0,  1,  1,  0,  0,  1,  4,  4,  3,  4,  1,  1,  4,  3,
        1,  1,  2,  0,  4,  4,  0,  4,  1,  1,  4,  0,  4,  1,  0,  4,  2,
        0,  1,  4,  4,  1,  3,  3,  0,  1,  4,  0,  0,  1,  4,  1,  4,  1,
        4,  1,  1,  4,  1,  4,  0,  4,  0,  0,  4,  4,  1,  1, -1,  2,  4,
        1,  2,  1,  3,  0,  0,  0,  1,  1,  4,  4,  3,  4,  3,  0,  4,  2,
        1,  0,  1,  2,  0,  1,  1,  0,  1,  4,  0,  3,  4,  4,  0,  1,  1,
        1,  1,  1,  4,  4,  1,  0,  4,  0,  3,  1,  1,  0,  1,  1,  0,  1,
        1,  4,  1,  0,  4,  2,  4,  0,  1,  4,  0,  0,  0,  1,  4,  2,  4,
        0,  4,  0,  0,  2,  4,  3,  2,  4,  4,  1,  0,  1,  4,  1,  4,  0,
        0,  1,  3,  4,  1,  4,  0,  0,  4,  0,  1,  0,  4,  0,  1,  0,  1,
        2,  4,  4,  2,  4,  4,  1,  1,  0,  2,  0,  4,  4,  3,  2,  4,  2,
        1,  4,  4,  0,  4,  1,  1,  2,  0,  4,  0,  0,  4,  0,  0,  0,  1,
        4,  4,  0,  0,  0,  1,  4,  1,  1,  1,  1,  2,  1,  1,  1,  0,  0,
        1,  1,  0,  2,  1,  4,  2,  4,  1,  1,  1,  1])
```

```
silhouette_score(df_scaled, db1.labels_)
```

```
0.7730490088903714
```

Tuning Using Elbow Method

```
silhouettes = {}

epss = np.arange(0.1, 0.9, 0.1)
minss = [3, 4, 5, 6, 7, 8, 9, 10, 12]

ss = np.zeros((len(epss), len(minss)))

for i, eps in enumerate(epss):
    for j, mins in enumerate(minss):
        db1 = DBSCAN(eps=eps, min_samples=mins).fit(df_scaled)
        if len(set(db1.labels_)) == 1:
            ss[i, j] = -1
        else:
            ss[i, j] = silhouette_score(df_scaled, db1.labels_, metric='euclidean')


plt.figure();
#plt.plot(list(silhouettes.keys()), list(silhouettes.values()));
for i in range(len(minss)):
    plt.plot(epss, ss[:, i], label="MinPts = {}".format(minss[i]));
#plt.plot(epss, ss[:, 1]);
plt.title('DBSCAN, Elbow Method')
plt.xlabel("Eps");
plt.ylabel("Silhouette");
plt.legend();
```




```
#Using Optimal Tuning as shown in above plot
db2 = DBSCAN(eps=0.8, min_samples=12)
db2.fit(df_scaled)
```

```
DBSCAN(algorithm='auto', eps=0.8, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=12, n_jobs=None, p=None)
```

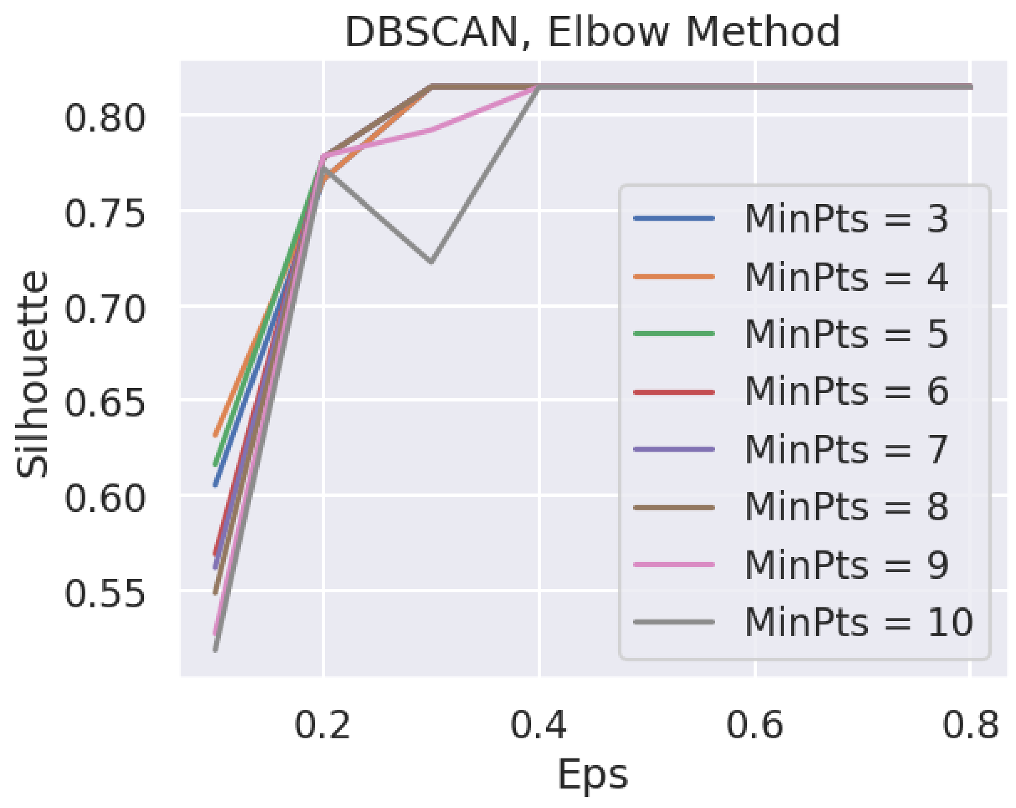
```
print(silhouette_score(df_scaled, db2.labels_))
print(calinski_harabasz_score(df_scaled,db2.labels_))
```

```
0.8102680974713122
83.88761899147732
```

Clustering & Tuning on Age & Spending Score (DBSCAN)

```
silhouettes = {}

epss = np.arange(0.1, 0.9, 0.1)
minss = [3, 4, 5, 6, 7, 8, 9, 10]

ss = np.zeros((len(epss), len(minss)))

for i, eps in enumerate(epss):
    for j, mins in enumerate(minss):
        db = DBSCAN(eps=eps, min_samples=mins).fit(X)
        if len(set(db.labels_)) == 1:
            ss[i, j] = -1
        else:
            ss[i, j] = silhouette_score(X, db.labels_, metric='euclidean')


plt.figure();
#plt.plot(list(silhouettes.keys()), list(silhouettes.values()));
for i in range(len(minss)):
    plt.plot(epss, ss[:, i], label="MinPts = {}".format(minss[i]));
#plt.plot(epss, ss[:, 1]);
plt.title('DBSCAN, Elbow Method')
plt.xlabel("Eps");
plt.ylabel("Silhouette");
plt.legend();
```

```
#plt.savefig('out/simple_dbscan_elbow');
```



Tuning with 0.5 EPS

```
scaler = StandardScaler()
features = ['Age', 'SpendingScore']
X[features] = scaler.fit_transform(X[features])

X.describe()

#Normalizing Data

scaler = StandardScaler()
features = ['Age', 'SpendingScore']
X[features] = scaler.fit_transform(X[features])
```

```
#Using Optimal Tuning as shown in above plot
db3 = DBSCAN(eps=0.5, min_samples=8)
db3.fit(X)
```

```
    DBSCAN(algorithm='auto', eps=0.5, leaf_size=30, metric='euclidean',
           metric_params=None, min_samples=8, n_jobs=None, p=None)
```

```
db3.labels_
```

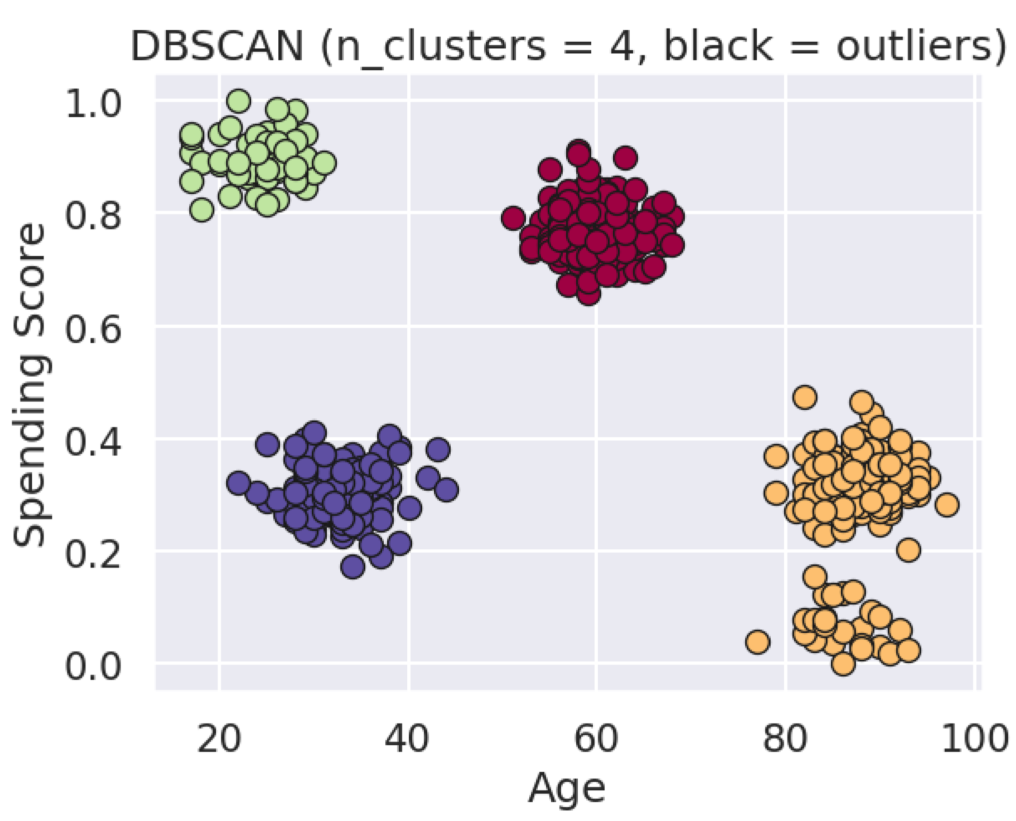
```
    array([0, 0, 0, 0, 1, 2, 0, 1, 1, 1, 1, 3, 3, 3, 0, 1, 0, 1, 1, 1, 3, 1,
           0, 1, 1, 2, 2, 0, 1, 0, 0, 0, 1, 0, 3, 0, 1, 0, 3, 3, 0, 1, 2, 0,
           0, 2, 1, 2, 3, 0, 1, 3, 2, 1, 0, 1, 3, 1, 1, 1, 1, 1, 1, 0, 0, 0,
           1, 2, 2, 0, 0, 3, 1, 1, 3, 0, 3, 0, 0, 2, 0, 1, 0, 1, 1, 3, 3, 0,
           0, 0, 0, 0, 2, 1, 0, 2, 3, 0, 0, 3, 3, 2, 3, 2, 3, 1, 1, 3, 3, 0,
           1, 3, 0, 3, 3, 3, 0, 3, 2, 1, 3, 1, 1, 1, 3, 1, 0, 1, 0, 0, 3, 0,
           3, 0, 2, 1, 1, 2, 0, 3, 1, 0, 0, 2, 0, 2, 0, 1, 0, 1, 0, 3, 1, 0,
           1, 1, 3, 0, 1, 0, 3, 1, 0, 3, 1, 3, 1, 0, 3, 0, 1, 0, 3, 0, 1, 0,
           0, 1, 1, 1, 0, 0, 3, 2, 2, 1, 3, 0, 0, 0, 3, 1, 0, 0, 0, 1, 2, 3,
           1, 1, 1, 3, 0, 3, 0, 3, 0, 1, 2, 0, 1, 1, 1, 1, 0, 1, 3, 2, 3, 1,
           0, 0, 3, 1, 3, 2, 3, 0, 0, 1, 0, 0, 1, 0, 0, 0, 3, 0, 3, 3, 2, 0,
           0, 0, 1, 0, 3, 1, 2, 2, 2, 1, 0, 3, 1, 1, 3, 3, 0, 0, 1, 0, 0, 1,
           0, 1, 3, 2, 0, 3, 1, 0, 1, 1, 3, 0, 1, 1, 0, 0, 1, 3, 3, 1, 3, 1,
           1, 3, 1, 1, 1, 2, 0, 3, 3, 0, 3, 1, 1, 3, 0, 3, 1, 0, 3, 2, 0, 1,
           3, 3, 1, 1, 1, 0, 1, 3, 0, 0, 1, 3, 1, 3, 1, 3, 1, 1, 3, 1, 3, 0,
           3, 0, 0, 3, 3, 1, 1, 2, 2, 3, 1, 2, 1, 1, 0, 0, 0, 1, 1, 3, 3, 1,
           3, 1, 0, 3, 2, 1, 0, 1, 2, 0, 1, 1, 0, 1, 3, 0, 1, 3, 3, 0, 1, 1,
           1, 1, 1, 3, 3, 1, 0, 3, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 3, 1, 0, 3,
           2, 3, 0, 1, 3, 0, 0, 0, 1, 3, 2, 3, 0, 3, 0, 0, 2, 3, 1, 2, 3, 3,
           1, 0, 1, 3, 1, 3, 0, 0, 1, 1, 3, 1, 3, 0, 0, 3, 0, 1, 0, 3, 0, 1,
           0, 1, 2, 3, 3, 2, 3, 3, 1, 1, 0, 2, 0, 3, 3, 1, 2, 3, 2, 1, 3, 3,
           0, 3, 1, 1, 2, 0, 3, 0, 0, 3, 0, 0, 0, 1, 3, 3, 0, 0, 0, 1, 3, 1,
           1, 1, 1, 2, 1, 1, 1, 0, 0, 1, 1, 0, 2, 1, 3, 2, 3, 1, 1, 1, 1])
```

```
print(silhouette_score(X, db3.labels_))
print(calinski_harabasz_score(X, db3.labels_))
```

```
    0.8569132370837265
    5709.487987910597
```

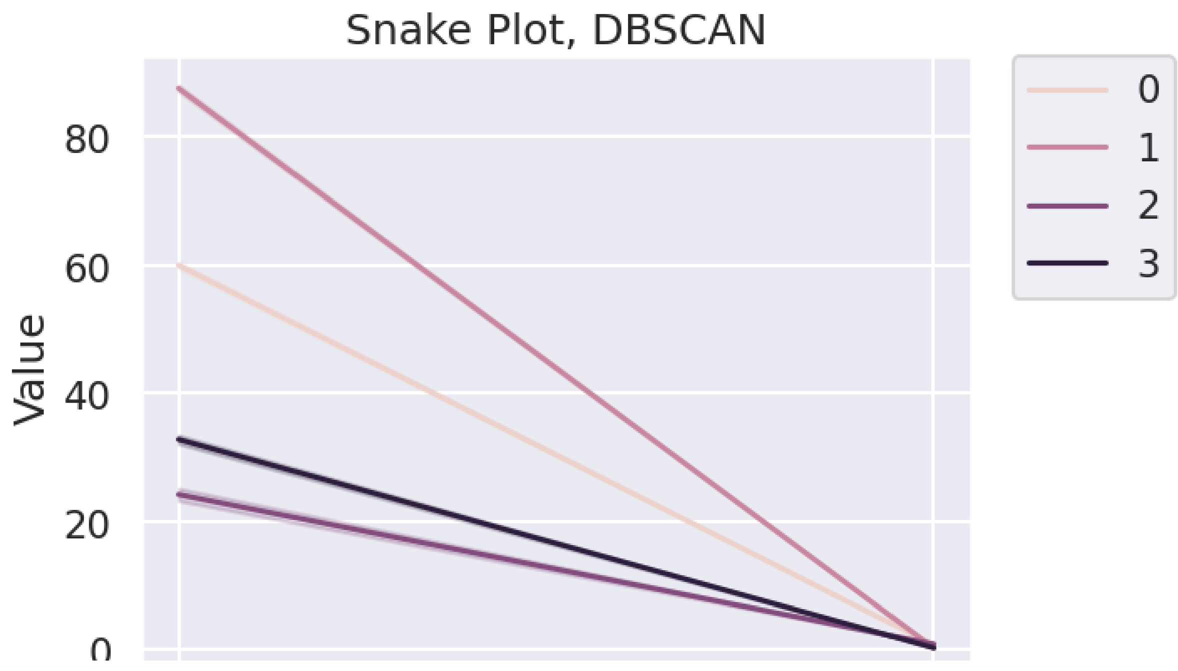
SIL and CH score higher when only using Age & Spending Score...

```
plt.figure();

unique_labels = set(db3.labels_)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))];

for k in unique_labels:
    if k == -1:        # Black used for noise.
        col = [0, 0, 0, 1]
    else:
        col = colors[k]

    xy = F[db3.labels_ == k]
    plt.plot(xy.iloc[:, 0], xy.iloc[:, 1], 'o', markerfacecolor=tuple(col), markeredgecolor='k', markersize=10);


plt.title('');
plt.title("DBSCAN (n_clusters = {:d}, black = outliers)".format(len(unique_labels)));
plt.xlabel('Age');
plt.ylabel('Spending Score');
```



When we use only Age & Spending Score, the SIL score is better.

```
import seaborn as sns

X_df = pd.DataFrame(F)
X_df['Cluster'] = db3.labels_
X_df.head()

X_df_melt = pd.melt(X_df,
    id_vars=['Cluster'],
    value_vars=['Age', 'SpendingScore'],
    var_name='Feature',
    value_name='Value')

plt.title('Snake Plot, DBSCAN');
sns.set(style="darkgrid");
sns.set_context("talk");
sns.lineplot(x="Feature", y="Value", hue='Cluster', data=X_df_melt, legend="full");
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.);
#plt.savefig('out/mall_heatmap.png', bbox_inches = "tight")
```



1.3 Model Comparison

K-Means

-SIL Score with 5 Clusters (Age & Spending Score): 0.8151026432983561

-CH Score with 5 Clusters (Age & Spending Score): 3367.2216515349933

-Inertia with 5 Clusters (Age & Spending Score): 47.72479392364724

DBSCAN (Age & Spending Score)

SIL SCORE: 0.8569132370837265

CH SCORE: 5709.487987910597

After testing multiple pre-processing, inter-changing features for clustering, & using different parameters for tuning using the Elbow method - The best modification of K-Means clustering was with 4 clusters when we performed a Standard Scaling while using Age & Spending Score to cluster with a Silhouette Score of 0.815.

While, for the 2nd Algorithm, we used DBSCAN clustering. Using the same steps & modifications used in the 1st algorithm, the best modification was also when we only used Age & Spending Score features to cluster the customers in 4 clusters. This resulted in a Silhouette score of 0.856.

The quality of the DBSCAN results are better with a higher SIL score of 0.856, that means a good amount of clusters were assigned to their correct cluster. As higher values mean that an instance is closer to its own cluster than other clusters.

Both K-Means and DBSCAN were easy to interpret when using only 2 features or reducing the number of clusters but DBSCAN gave high score in the end. Both are easy to use and and the speed for creating the models was fast for both of them. But in quality - DBSCAN has won in evaluation.

1.4 Personas

Please note: The code, visualizations & numbers are below this answer as evidence.

The personas described below are based on the clusters statistics and their respective features. We have 4 clusters from our DBSCAN model

Cluster 0: "About to Hang Boots" (Red Cluster)

This cluster has an average of people who are in their late 50s, average around 60, it can be assumed that they are either nearing their retirement or already retired. Their cluster is depicted by the color **red** in the graph below. They have the second highest average score in the Spending Score criteria -0.77. They have spent a lot on the store and they seem to spend a good amount on jewellery. They dont have a lot of income and savings on average (third highest income cluster), but they seem stable in their life and now want to live it to the fullest by buying what they desire.

Cluster 1: "Legends" (Yellow Cluster)

The oldest age group out of all clusters has an average spending score 0f 0.29, which is the lowest out of all clusters. These customers are not regular, they only shop for some special items as they have the lowest income, maybe due to retirement, although the Std. Deviation for their income is highest, means there are quite a few who are rich & still earning. They have a lot of savings, even though they don't need a lot of jewellery to wear but they may make some purchases on special events from their savings.

Cluster 2: "Customers who live by this slang - YOLO " (Green Cluster)

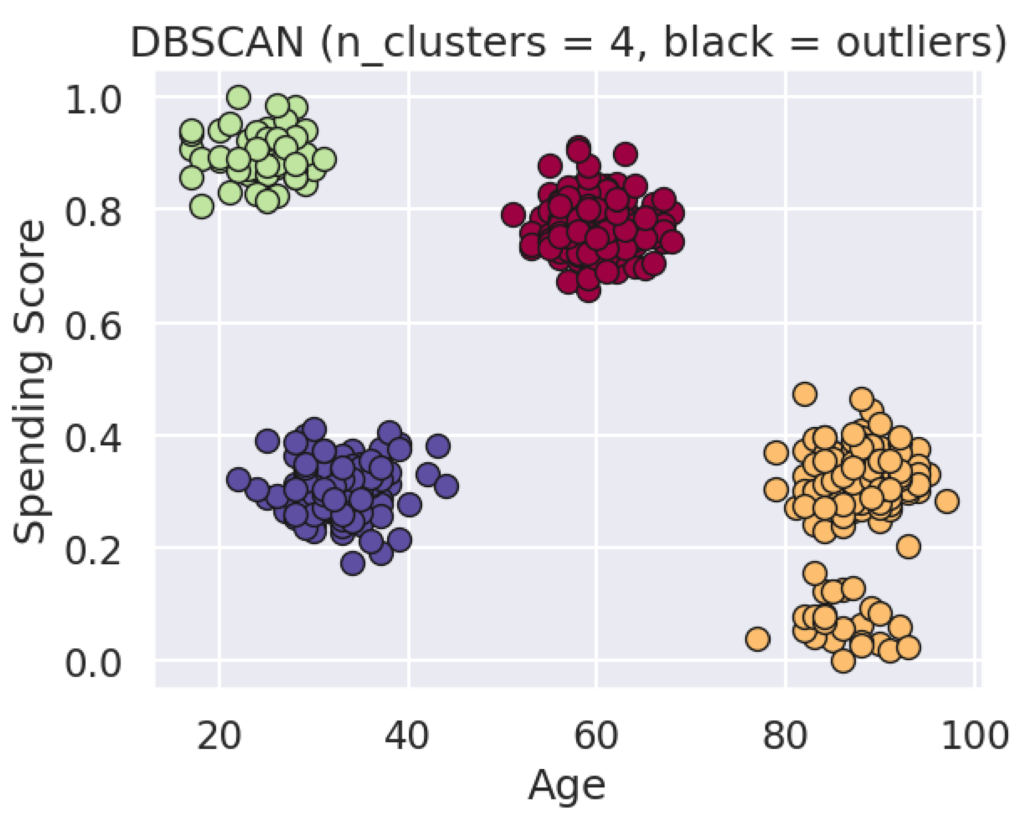
The youngest cluster of customers love buying from Uncle Steve. They are his shop's regular customers and bring a good amount of business to him. They have the highest income from all the clusters but they don't save a lot. They fulfill their daily changing fashion needs by buying new jewellery regularly. Since they have a 'you only live once' mindset, they don't care about savings. Attractive marketing makes them buy new items to wear trendy items. Retaining them should be a priority.

Cluster 3: "Puny Spenders" (Purple cluster)

With the second lowest Spending Score, these customers are the 2nd youngest, earns the 2nd highest income and save a lot. Flashy marketing does not attract them and they want to save for the future. They only buy when its really necessary. Even though, they have a good inflow of cash, they would rather save a good chunk of it than spend on their fashionble needs. It's a difficult task to convince them.They might spend on a budget, so its a good option to offer them items under their budget, else they love saving that money! They are in their 30's so maybe they are saving for the future, or if they are married they might have kids, they might be saving for their college tuitions or maybe planning to buy some expensive asset.

Coding Interpretations

```
plt.figure();

unique_labels = set(db3.labels_)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))];

for k in unique_labels:
    if k == -1:        # Black used for noise.
        col = [0, 0, 0, 1]
    else:
        col = colors[k]

    xy = F[db3.labels_ == k]
    plt.plot(xy.iloc[:, 0], xy.iloc[:, 1], 'o', markerfacecolor=tuple(col), markeredgecolor='k', markersize=10);


plt.title('');
plt.title("DBSCAN (n_clusters = {:d}, black = outliers)".format(len(unique_labels)));
plt.xlabel('Age');
plt.ylabel('Spending Score');
```




```
# Let's look at some example rows in each.

for label in set(db3.labels_):
    print('\nCluster {}:'.format(label))
    print(df1[db3.labels_==label].head())
```

```
Cluster 0:
   Age  Income  SpendingScore      Savings
0   58   77769       0.791329  6559.829923
1   59   81799       0.791082  5417.661426
```

```
2   62   74751       0.702657  9258.992965
3   59   74373       0.765680  7346.334504
6   54   76500       0.785198  6878.884249

Cluster 1:
    Age  Income  SpendingScore       Savings
4    87   17760       0.348778  16869.507130
7    87   42592       0.355290  18086.287158
8    83   34384       0.324719  14783.379086
9    84   27693       0.367063  17879.558906
10   85  111389       0.036795  16009.237763

Cluster 2:
    Age  Income  SpendingScore      Savings
5    29  131578       0.847034  3535.514352
25   30  122788       0.872872  5706.149573
26   17  134966       0.907242  4128.044796
42   20  129142       0.887052  5603.121028
45   18  130813       0.890891  5256.434560

Cluster 3:
    Age  Income  SpendingScore       Savings
11   36   99780       0.265433  16398.401333
12   30   99949       0.344679  13621.639726
13   31  107963       0.290509  13407.081391
20   30  101073       0.314387  14324.555977
34   33  101058       0.315082  14911.868398
```

Using Panda's group-by function to group Stats of different clusters...

```
from scipy import stats

import pandas as pd


X_df = pd.DataFrame(df1)
X_df['Cluster'] = db3.labels_
X_df.head()

cl_group = X_df.groupby(['Cluster']).agg('describe')


cl_group['Age']
print('Spending Score')
cl_group['SpendingScore']
print('Income')
cl_group['Income']
print('Savings')
cl_group['Savings']
```

	Age	Income	SpendingScore	Savings	Cluster
0	58	77769	0.791329	6559.829923	0
1	59	81799	0.791082	5417.661426	0
2	62	74751	0.702657	9258.992965	0
3	59	74373	0.765680	7346.334504	0
4	87	17760	0.348778	16869.507130	1

	count	mean	std	min	25%	50%	75%	max
Cluster								
0	157.0	59.955414	3.376662	51.0	58.0	59.0	62.0	68.0
1	172.0	87.517442	3.576195	77.0	85.0	88.0	90.0	97.0
2	50.0	24.180000	3.662775	17.0	22.0	24.5	27.0	31.0
3	126.0	32.777778	3.792390	22.0	30.0	33.0	35.0	44.0

Spending Score

	count	mean	std	min	25%	50%	75%	max
Cluster								
0	157.0	0.771518	0.046058	0.657314	0.740367	0.766720	0.800598	0.910417
1	172.0	0.290948	0.102186	0.000000	0.279134	0.316756	0.353532	0.473550
2	50.0	0.896892	0.043466	0.806553	0.871957	0.890676	0.926473	1.000000
3	126.0	0.309926	0.045513	0.174120	0.281237	0.309479	0.341096	0.411112

Income

	count	mean	std	min	25%	50%	75%	max
Cluster								
0	157.0	72448.063694	6240.260008	56321.0	68463.00	72027.0	76594.00	90422.0
1	172.0	41249.523256	33140.527666	12000.0	24387.25	28915.5	34864.25	128596.0
2	50.0	128029.120000	5688.904656	117108.0	123042.00	128162.0	131435.75	142000.0
3	126.0	105265.809524	6080.621753	89598.0	100760.25	106002.5	108858.75	119877.0

Savings

	count	mean	std	min	25%	50%	75%	max
Cluster								
0	157.0	6889.972190	1052.276354	4077.658657	6225.376082	6845.056822	7497.231607	10547.775368
1	172.0	16390.282135	1346.532447	12554.692742	15470.915224	16509.338762	17324.577737	20000.000000
2	50.0	4087.520309	1277.754801	0.000000	3275.320193	4361.967019	4986.863329	6089.478323
3	126.0	14962.778066	1061.734017	12207.526078	14223.787562	14976.943192	15682.288845	17968.553929

Examplars

```
from scipy import stats
from statistics import mean
```

```
K=4
```

```
np.set_printoptions(precision=2)
np.set_printoptions(suppress=True)

means = np.zeros((K,df1.shape[1]))

for i, label in enumerate(set(db3.labels_)):
    means[i,:] = df1[db3.labels_==label].mean(axis=0)
    print('\nCluster {} (n={}):'.format(label, sum(db3.labels_==label)))
    print((means[i,:]))

means
```

```
Cluster 0 (n=157):
[   59.96 72448.06     0.77  6889.97     0.  ]

Cluster 1 (n=172):
[   87.52 41249.52     0.29 16390.28     1.  ]

Cluster 2 (n=50):
[    24.18 128029.12      0.9    4087.52      2.  ]

Cluster 3 (n=126):
[    32.78 105265.81      0.31  14962.78      3.  ]
array([[    59.96,  72448.06,      0.77,   6889.97,      0.  ],
       [    87.52,  41249.52,      0.29,  16390.28,      1.  ],
       [    24.18, 128029.12,      0.9 ,   4087.52,      2.  ],
       [    32.78, 105265.81,      0.31,  14962.78,      3.  ]])
```

```
from scipy.spatial import distance

for i, label in enumerate(set(db3.labels_)):
    X_tmp= df1
    exemplar_idx = distance.cdist([means[i]], df1).argmin()

    print('\nCluster {}:'.format(label))
    #print("  Examplar ID: {}".format(exemplar_idx))
    #print("  Label: {}".format(labels[exemplar_idx]))
    #print("  Features:")
    display(df1.iloc[[exemplar_idx]])
```

Cluster 0:

	Age	Income	SpendingScore	Savings	Cluster
419	51	72086	0.791115	6732.096069	0

Cluster 1:

	Age	Income	SpendingScore	Savings	Cluster
122	84	42018	0.297994	16148.370454	1

Question 2: Uncle Steve's Fine Foods

Instructions

Uncle Steve runs a small, local grocery store in Ontario. The store sells all the normal food staples (e.g., bread, milk, cheese, eggs, more cheese, fruits, vegatables, meat, fish, waffles, ice cream, pasta, cereals, drinks), personal care products (e.g., toothpaste, shampoo, hair goo), medicine, and cakes. There's even a little section with flowers and greeting cards! Normal people shop here, and buy normal things in the normal way.

Business is OK but Uncle Steve wants more. He's thus on the hunt for customer insights. Given your success at the jewelry store, he has asked you to help him out.

He has given you a few years' worth of customer transactions, i.e., sets of items that customers have purchased. You have applied an association rules learning algorithm (like Apriori) to the data, and the algorithm has generated a large set of association rules of the form `{X} -> {Y}`, where `{X}` and `{Y}` are item-sets.

Now comes a thought experiment. For each of the following scenarios, state what one of the discovered association rules might be that would meet the stated condition. (Just make up the rule, using your human experience and intuition.) Also, describe whether and why each rule would be considered interesting or uninteresting for Uncle Steve (i.e., is this insight new to him? Would he be able to use it somehow?).

Keep each answer to 600 characters or less (including spaces).

To get those brain juices going, an example condition and answer is provided below:

> Condition: A rule that has high support.

> Answer: The rule `{milk} -> {bread}` would have high support, since milk and bread are household staples and a high percentage of transactions would include both `{milk}` and `{bread}`. Uncle Steve would likely not find this rule interesting, because these items are so common, he would have surely already noticed that so many transactions contain them.

Marking

Your responses will be marked as follows:

- *Correctness*. Rule meets the specificed condition, and seems plausible in an Ontario grocery store.
- *Justification of interestness*. Response clearly describes whether and why the rule would be considered interesting to Uncle Steve.

Tips

- There is no actual data for this question. This question is just a thought exercise. You need to use your intuition, creatitivty, and understanding of the real world. I assume you are familiar with what happens inside of normal grocery stores. We are not using actual data and you do not need to create/generate/find any data. I repeat: there is no data for this question.
- The reason this question is having you do a thought experiment, rather than writing and running code to find actual association rules on an actual dataset, is because writing code to find association rules is actually pretty easy. But using your brain to come up with rules that meet certain criteria, on the other hand, is a true test of whether you understand how the algorithm works, what support and confidence mean, and the applicability of rules. The question uses the grocery store context because most, if not all, students should be familiar from personal experience.

2.1: A rule that might have high support and high confidence.

Cereal -> Milk.

One could easily infer that if someone is buying Cereal, they need Milk hence this association rule should have a high confidence as milk is required with cereal for normal people. This rule should have high support too because both items are usually bought together for breakfast.

Uncle Steve will not find this rule interesting as he has probably noticed that people buy these items together.

2.2: A rule that might have reasonably high support but low confidence.

Soft Drinks -> Chips

The rule has high support because when kids are to buying junk foods, they buy both these items and they must show up together a lot in transactions. Although this rule has low confidence because one doesn't necessarily need to buy chips with a soft drink, a lot of people might just buy the soft drink only.

This rule should be interesting for Uncle Steve when he finds out all those kids buying these two items together, just because they like junk food.

2.3: A rule that might have low support and low confidence.

Toilet Paper -> Vegetables

Well, this type of grocerer wanted all toilet paper at home in the pandemic for some odd reason. Apparently he is vegan and he needs to buy fresh vegetables daily for his nutrition needs and for the last year he has been buying toilet paper whenever he goes to buy his veggies. Eating healthy doesn't stop Covid-19 affecting your lifestyle, it seems.

Uncle Steve should definitely find this rule interesting and increase the prices of toilet paper whenever vegetable prices increase to earn some double revenue. Atleast until vaccine passports are here.

2.4: A rule that might have low support and high confidence.

Frozen Sausages -> Mustard

You always need some mustard to make those hot dogs and people do need mustard. So, probably mustard will be required when buying sausages. But it is not necessary they occur together in every transaction as mustard is used for other purposes too and it lasts long to be bought every time, hence low support.

Should not be an interesting rule for Uncle Steve, as he probably has knowledge of this happening.

Question 3: Uncle Steve's Credit Union

Instructions

Uncle Steve has recently opened a new credit union in Kingston, named *Uncle Steve's Credit Union*. He plans to disrupt the local market by instaneously providing credit to customers.

The first step in Uncle Steve's master plan is to create a model to predict whether an application has *good risk* or *bad risk*. He has outsourced the creation of this model to you.

You are to create a classification model to predict whether a loan applicant has good risk or bad risk. You will use data that Uncle Steve bought from another credit union (somewhere in Europe, he thinks?) that has around 6000 instances and a number of demographics features (e.g., `Sex`, `DateOfBirth`, `Married`), loan details (e.g., `Amount`, `Purpose`), credit history (e.g., number of loans), as well as an indicator (called `BadCredit` in the dataset) as to whether that person was a bad risk.

Your tasks

To examine the effects of the various ML stages, you are to create the model several times, each time adding more sophistication, and measuring how much the model improved (or not). In particular, you will:

0. Split the data in training and testing. Don't touch the testing data again, for any reason, until step 5. We are pretending that the testing data is "future, unseen data that our model won't see until production." I'm serious, don't touch it. I'm watching you!
1. Build a baseline model - no feature engineering, no feature selection, no hyperparameter tuning (just use the default settings), nothing fancy. (You may need to do some basic feature transformations, e.g., encoding of categorical features, or dropping of features you do not think will help or do not want to deal with yet.) Measure the performance using K-fold cross validation (recommended: `sklearn.model_selection.cross_val_score`) on the training data. Use at least 5 folds, but more are better. Choose a `scoring` parameter (i.e., classification metric) that you feel is appropriate for this task. Don't use accuracy. Print the mean score of your model.
2. Add a bit of feature engineering. The `sklearn.preprocessing` module contains many useful transformations. Engineer at least three new features. They don't need to be especially ground-breaking or complicated. Dimensionality reduction techniques like `sklearn.decomposition.PCA` are fair game but not required. (If you do use dimensionality reduction techniques, it would only count as "one" new feature for the purposes of this assignment, even though I realize that PCA creates many new "features" (i.e., principal componentns).) Re-train your baseline model. Measure performance. Compare to step 1.
3. Add feature selection. The `sklearn.feature_selection` has some algorithms for you to choose from. After selecting features, re-train your model, measure performance, and compare to step 2.

4. Add hyperparameter tuning. Make reasonable choices and try to find the best (or at least, better) hyperparameters for your estimator and/or transformers. It's probably a good idea to stop using `cross_val_score` at this point and start using `sklearn.model_selection.GridSearchCV` as it is specifically built for this purpose and is more convienient to use. Measure performance and compare to step 3.
5. Finally, estimate how well your model will work in production. Use the testing data (our "future, unseen data") from step 0. Transform the data as appropriate (easy if you've built a pipeline, a little more difficult if not), use the model from step 4 to get predictions, and measure the performance. How well did we do?

Marking

Each part will be marked for:

- *Correctness*. Code clearly and fully performs the task specified.
- *Reproducibility*. Code is fully reproducible. I.e., you (and I) should be able to run this Notebook again and again, from top to bottom, and get the same results each and every time.
- *Style*. Code is organized. All parts commented with clear reasoning and rationale. No old code laying around. Code easy to follow.

Tips

- The origins of the dataset are a bit of a mystery. Assume the data set is recent (circa 2021) and up-to-date. Assume that column names are correct and accurate.
- You don't need to experiment with more than one algorithm/estimator. Just choose one (e.g., `sklearn.tree.DecisionTreeClassifier`, `sklearn.ensemble.RandomForestClassifier`, `sklearn.linear_model.LogisticRegression`, `sklearn.svm.LinearSVC`, whatever) and stick with it for this question.
- There is no minimum accuracy/precision/recall for this question. I.e., your mark will not be based on how good your model is. Rather, you mark will be based on good your process is.
- Watch out for data leakage and overfitting. In particular, be sure to `fit()` any estimators and transformers (collectively, *objects*) only to the training data, and then use the objects' `transform()` methods on both the training and testing data. Data School has a helpful video about this. Pipelines are very helpful here and make your code shorter and more robust (at the expense of making it harder to understand), and I recommend using them, but they are not required for this assignment.
- Create as many code cells as you need. In general, each cell should do one "thing."
- Don't print large volumes of output. E.g., don't do: `df.head(100)`

3.0: Load data and split

```
# DO NOT MODIFY THIS CELL

# First, we'll read the provided labeled training data
df3 = pd.read_csv("https://drive.google.com/uc?export=download&id=1wOhyCnvGeY4jplxI8lZ-bbYN3zLtickf")
df3.info()

from sklearn.model_selection import train_test_split

X = df3.drop('BadCredit', axis=1) #.select_dtypes(['number'])
y = df3['BadCredit']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   UserID             6000 non-null   object
 1   Sex                6000 non-null   object
 2   PreviousDefault    6000 non-null   int64 
 3   FirstName          6000 non-null   object
 4   LastName           6000 non-null   object
 5   NumberPets         6000 non-null   int64 
 6   PreviousAccounts   6000 non-null   int64 
 7   ResidenceDuration  6000 non-null   int64 
 8   Street             6000 non-null   object
 9   LicensePlate       6000 non-null   object
 10  BadCredit          6000 non-null   int64 
 11  Amount             6000 non-null   int64 
 12  Married            6000 non-null   int64 
 13  Duration           6000 non-null   int64 
 14  City               6000 non-null   object
 15  Purpose            6000 non-null   object
 16  DateOfBirth        6000 non-null   object
dtypes: int64(8), object(9)
memory usage: 797.0+ KB
```

```
df3.head()
```

	UserID	Sex	PreviousDefault	FirstName	LastName	NumberPets	PreviousAccounts	ResidenceDuration	Street	LicensePlate	BadCredit	Amount	Married	Duration	City	Purpose	DateOfBirth
0	218-84-8180	F	0	Debra	Schaefer	2	3	1	503 Linda Locks	395C	0	3907	0	24	Port Keith	Vacation	1964-04-07
1	395-49-9764	M	0	Derek	Wright	0	1	1	969 Cox Dam Suite 101	UFZ 691	0	3235	0	12	Lake Debra	NewCar	1978-06-02
2	892-81-4890	F	0	Shannon	Smith	0	0	2	845 Kelly Estate	48A•281	0	3108	1	30	North Judithbury	NewCar	1972-03-18
3	081-11-7963	F	0	Christina	Brooks	2	1	3	809 Burns Creek	30Z J39	1	4014	1	36	Lake Chad	Other	1985-02-26
4	347-03-9639	M	0	Ralph	Anderson	1	5	1	248 Brandt Plains Apt. 465	71-Q331	1	3823	0	18	North Judithbury	Vacation	1983-08-08

```
sns.heatmap(df3.isnull(),yticklabels = False, cbar = False,cmap = 'tab20c_r')
plt.title('Missing Data')
plt.show()
```

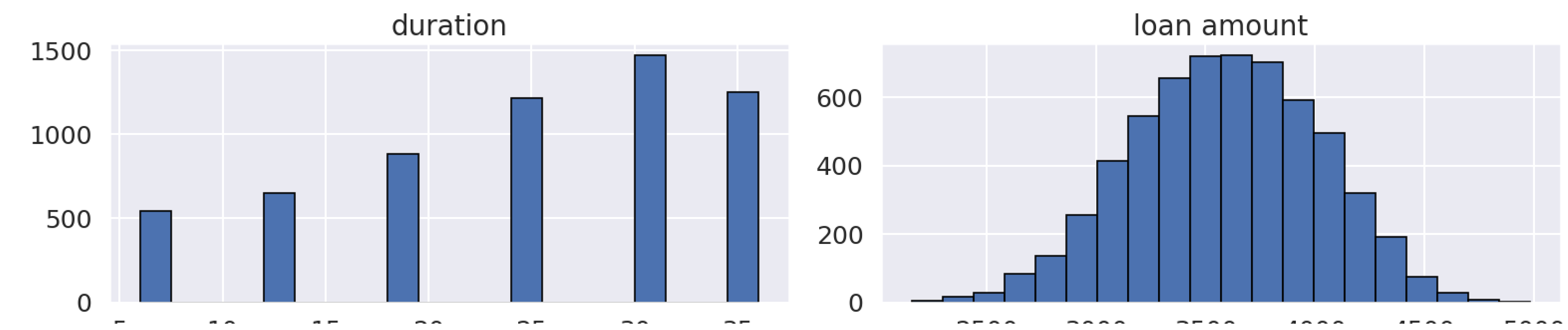
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4859415990>Text(0.5, 1.0, 'Missing Data')
```

3.1: Baseline model

```
X_train.head()
```

	UserID	Sex	PreviousDefault	FirstName	LastName	NumberPets	PreviousAccounts	ResidenceDuration	Street	LicensePlate	Amount	Married	Duration	City	Purpose	DateOfBirth
3897	236-22-6766	M	0	Jerry	Black	2	0	2	0466 Brown Wall	3-U8282	3329	0	12	New Roberttown	Household	1970-04-22
5628	766-20-5986	F	0	Julia	Jones	0	2	2	6095 Larson Causeway	LWO 912	2996	0	36	Ericmouth	Household	1964-06-19
1756	744-25-5747	F	0	Abigail	Estrada	2	0	3	293 Michael Divide	715 OQT	2470	0	24	East Jill	NewCar	1975-02-17
2346	463-78-3098	F	0	Jessica	Jones	2	1	2	02759 Williams Roads	869 SYK	3745	0	30	Lake Debra	UsedCar	1977-02-16
2996	414-44-6527	M	0	William	Shaffer	0	1	3	19797 Turner Rue	48-A601	3549	0	36	North Judithbury	Vacation	1976-07-27

```
def plot_hist(ax, feature, title):
    ax.hist(feature, bins=20, edgecolor='black', linewidth=1.2);
    ax.set_title(title, fontsize=20);
    ax.tick_params(axis='both', which='major', labelsize=18);
    ax.grid(True);

plt.figure(figsize=(16, 10));
plt.grid(True);
plot_hist(plt.subplot(3, 2, 1), X['Duration'], 'duration')
plot_hist(plt.subplot(3, 2, 2), X['Amount'],'loan amount')

plt.tight_layout();
```



Turning objects into categories for Baseline model

```
X_train['Sex'] = X_train['Sex'].astype('category')
X_train['PreviousDefault'] = X_train['PreviousDefault'].astype('category')
X_train['Married'] = X_train['Married'].astype('category')
X_train['Purpose'] = X_train['Purpose'].astype('category')

y_train = y_train.astype('category')
```

```
y_train.head()
```

```
3897    0
5628    0
1756    0
2346    0
2996    0
Name: BadCredit, dtype: category
Categories (2, int64): [0, 1]
```

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4800 entries, 3897 to 860
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype   
---  ------             --------------  -----   
 0   UserID             4800 non-null   object  
 1   Sex                4800 non-null   category
 2   PreviousDefault    4800 non-null   category
 3   FirstName          4800 non-null   object  
 4   LastName           4800 non-null   object  
 5   NumberPets         4800 non-null   int64   
 6   PreviousAccounts   4800 non-null   int64   
 7   ResidenceDuration  4800 non-null   int64   
 8   Street             4800 non-null   object  
 9   LicensePlate       4800 non-null   object  
 10  Amount             4800 non-null   int64   
 11  Married            4800 non-null   category
 12  Duration           4800 non-null   int64   
 13  City               4800 non-null   object  
 14  Purpose            4800 non-null   category
 15  DateOfBirth        4800 non-null   object  
dtypes: category(4), int64(5), object(7)
memory usage: 506.9+ KB
```

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
```

```
X_train.head()
```

	UserID	Sex	PreviousDefault	FirstName	LastName	NumberPets	PreviousAccounts	ResidenceDuration	Street	LicensePlate	Amount	Married	Duration	City	Purpose	DateOfBirth
3897	236-22-6766	M	0	Jerry	Black	2	0	2	0466 Brown Wall	3-U8282	3329	0	12	New Roberttown	Household	1970-04-22
5628	766-20-5986	F	0	Julia	Jones	0	2	2	6095 Larson Causeway	LWO 912	2996	0	36	Ericmouth	Household	1964-06-19
1756	744-25-5747	F	0	Abigail	Estrada	2	0	3	293 Michael Divide	715 OQT	2470	0	24	East Jill	NewCar	1975-02-17
2346	463-78-3098	F	0	Jessica	Jones	2	1	2	02759 Williams Roads	869 SYK	3745	0	30	Lake Debra	UsedCar	1977-02-16
2996	414-44-6527	M	0	William	Shaffer	0	1	3	19797 Turner Rue	48-A601	3549	0	36	North Judithbury	Vacation	1976-07-27

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4800 entries, 3897 to 860
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype   
---  ------             --------------  -----   
 0   UserID             4800 non-null   object  
 1   Sex                4800 non-null   category
 2   PreviousDefault    4800 non-null   category
 3   FirstName          4800 non-null   object  
 4   LastName           4800 non-null   object  
 5   NumberPets         4800 non-null   int64   
 6   PreviousAccounts   4800 non-null   int64   
 7   ResidenceDuration  4800 non-null   int64   
 8   Street             4800 non-null   object  
 9   LicensePlate       4800 non-null   object  
 10  Amount             4800 non-null   int64   
 11  Married            4800 non-null   category
 12  Duration           4800 non-null   int64   
 13  City               4800 non-null   object  
 14  Purpose            4800 non-null   category
 15  DateOfBirth        4800 non-null   object  
dtypes: category(4), int64(5), object(7)
memory usage: 506.9+ KB
```

```
#Pipeline 1

drop_features = ['Sex','Purpose','UserID','FirstName','LastName','Street','LicensePlate','City','DateOfBirth']

clf = RandomForestClassifier(random_state=42)


preprocessor1 = Pipeline(steps=[
     ('ct', ColumnTransformer(
       transformers=[
           ('drop', 'drop', drop_features)],
            remainder = 'passthrough',
           sparse_threshold=0)),
    ])


pipe1 = Pipeline(steps=[("preprocessor", preprocessor1), ("clf", clf)])

pipe1 = pipe1.fit(X_train, y_train)

scores1 = cross_val_score(pipe1, X_train, y_train,
                         scoring='f1_micro', cv=10, n_jobs=-1, error_score='raise',verbose=2)
print(scores1)
print(np.mean(scores1))
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[0.81 0.81 0.81 0.82 0.82 0.82 0.82 0.84 0.81 0.83]
0.8183333333333334
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    5.0s finished
```

F1 score from baseline model: 0.818

```
print(pipe1.named_steps['clf'].feature_importances_)
```

```
[0.01 0.05 0.08 0.11 0.65 0.02 0.09]
```

3.2: Feature engineering

```
df3 = pd.read_csv("https://drive.google.com/uc?export=download&id=1wOhyCnvGeY4jplxI8lZ-bbYN3zLtickf")
df3.info()

from sklearn.model_selection import train_test_split

X = df3.drop('BadCredit', axis=1) #.select_dtypes(['number'])
y = df3['BadCredit']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   UserID             6000 non-null   object
 1   Sex                6000 non-null   object
 2   PreviousDefault    6000 non-null   int64
 3   FirstName          6000 non-null   object
 4   LastName           6000 non-null   object
 5   NumberPets         6000 non-null   int64
 6   PreviousAccounts   6000 non-null   int64
 7   ResidenceDuration  6000 non-null   int64
 8   Street             6000 non-null   object
 9   LicensePlate       6000 non-null   object
 10  BadCredit          6000 non-null   int64
 11  Amount             6000 non-null   int64
 12  Married            6000 non-null   int64
 13  Duration           6000 non-null   int64
 14  City               6000 non-null   object
 15  Purpose            6000 non-null   object
 16  DateOfBirth        6000 non-null   object
dtypes: int64(8), object(9)
memory usage: 797.0+ KB
```

```
X_train.head()
```

	UserID	Sex	PreviousDefault	FirstName	LastName	NumberPets	PreviousAccounts	ResidenceDuration	Street	LicensePlate	Amount	Married	Duration	City	Purpose	DateOfBirth
3897	236-22-6766	M	0	Jerry	Black	2	0	2	0466 Brown Wall	3-U8282	3329	0	12	New Roberttown	Household	1970-04-22
5628	766-20-5986	F	0	Julia	Jones	0	2	2	6095 Larson Causeway	LWO 912	2996	0	36	Ericmouth	Household	1964-06-19
1756	744-25-5747	F	0	Abigail	Estrada	2	0	3	293 Michael Divide	715 OQT	2470	0	24	East Jill	NewCar	1975-02-17
2346	463-78-3098	F	0	Jessica	Jones	2	1	2	02759 Williams Roads	869 SYK	3745	0	30	Lake Debra	UsedCar	1977-02-16
2996	414-44-6527	M	0	William	Shaffer	0	1	3	19797 Turner Rue	48-A601	3549	0	36	North Judithbury	Vacation	1976-07-27

```
#Pipeline 2

numeric_features = ['Amount', 'Duration', 'NumberPets','Married','ResidenceDuration']

categorical_features = ['City', 'Sex','Purpose'] #Adding a category to purpose

drop_features = ['UserID','FirstName','LastName','Street','LicensePlate']

#DateofBirth Feature Engineering
def get_age_years(feature):
  res = np.array([])
  for instance in feature:
    age = 2021 - int(instance[0:4])
    res = np.append(res, age)
  return res.reshape(-1, 1)


clf = RandomForestClassifier(random_state=42)

numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ])


categorical_transformer = Pipeline(steps=[
      ('encoder', OrdinalEncoder())])

preprocessor2 = Pipeline(steps=[
      ('ct', ColumnTransformer(
         transformers=[
      ('num', numeric_transformer, numeric_features),
      ('cat', categorical_transformer, categorical_features),
      ('age', FunctionTransformer(get_age_years, validate=False), 'DateOfBirth'),
      ('drop', 'drop', drop_features)],
           remainder = 'passthrough',
          sparse_threshold=0)),
    ])


pipe2 = Pipeline(steps=[("preprocessor", preprocessor2), ("clf", clf)])

pipe2 = pipe2.fit(X_train, y_train)

scores2 = cross_val_score(pipe2, X_train, y_train,
                        scoring='f1_micro', cv=10, n_jobs=-1, error_score='raise',verbose=2)
print(scores2)
print(np.mean(scores2))
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[0.89 0.86 0.87 0.86 0.88 0.9  0.89 0.88 0.85 0.89]
0.878125
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    4.5s finished
```

Feature Engineering: (F1 Score improved to 0.878125)

1. Standardized Scaling on numeric features
2. Ordinal Encoder on categorical features which were 'City', 'Sex'&'Purpose'
3. Taking out Age of Users from Date of Birth Feature..

3.3: Feature selection

```
from sklearn.feature_selection import SelectKBest
```

```
df3 = pd.read_csv("https://drive.google.com/uc?export=download&id=1wOhyCnvGeY4jplxI8lZ-bbYN3zLtickf")
df3.info()

from sklearn.model_selection import train_test_split

X = df3.drop('BadCredit', axis=1) #.select_dtypes(['number'])
y = df3['BadCredit']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   UserID             6000 non-null   object
 1   Sex                6000 non-null   object
 2   PreviousDefault    6000 non-null   int64
 3   FirstName          6000 non-null   object
 4   LastName           6000 non-null   object
 5   NumberPets         6000 non-null   int64
 6   PreviousAccounts   6000 non-null   int64
```

```
 7   ResidenceDuration  6000 non-null   int64
 8   Street             6000 non-null   object
 9   LicensePlate       6000 non-null   object
 10  BadCredit          6000 non-null   int64
 11  Amount             6000 non-null   int64
 12  Married            6000 non-null   int64
 13  Duration           6000 non-null   int64
 14  City               6000 non-null   object
 15  Purpose            6000 non-null   object
 16  DateOfBirth        6000 non-null   object
dtypes: int64(8), object(9)
memory usage: 797.0+ KB
```

```
#Pipeline 3

numeric_features = ['Amount', 'Duration', 'NumberPets','Married','ResidenceDuration']

categorical_features = ['City', 'Sex','Purpose'] #Adding a category to purpose

drop_features = ['UserID','FirstName','LastName','Street','LicensePlate']

#DateofBirth Feature Engineering
def get_age_years(feature):
  res = np.array([])
  for instance in feature:
    age = 2021 - int(instance[0:4])
    res = np.append(res, age)
  return res.reshape(-1, 1)


clf = RandomForestClassifier(random_state=42)

numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ])


categorical_transformer = Pipeline(steps=[
      ('encoder', OrdinalEncoder())])

preprocessor3 = Pipeline(steps=[
      ('ct', ColumnTransformer(
          transformers=[
      ('num', numeric_transformer, numeric_features),
      ('cat', categorical_transformer, categorical_features),
      ('age', FunctionTransformer(get_age_years, validate=False), 'DateOfBirth'),
      ('drop', 'drop', drop_features)],
             remainder = 'passthrough',
           sparse_threshold=0)),
      ('feature_selector', SelectKBest(k=10))
    ])


pipe3 = Pipeline(steps=[("preprocessor", preprocessor3), ("clf", clf)])

pipe3 = pipe3.fit(X_train, y_train)

scores3 = cross_val_score(pipe3, X_train, y_train,
                         scoring='f1_micro', cv=10, n_jobs=-1, error_score='raise',verbose=2)
print(scores3)
print(np.mean(scores3))
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[0.89 0.87 0.87 0.87 0.87 0.9  0.89 0.89 0.86 0.9 ]
0.8804166666666667
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    4.4s finished
```

Feature Selection of using the 10 features with best K score did not improve F-1 score to 0.8804

3.4: Hyperparameter tuning

```
df3 = pd.read_csv("https://drive.google.com/uc?export=download&id=1wOhyCnvGeY4jplxI8lZ-bbYN3zLtickf")
df3.info()

from sklearn.model_selection import train_test_split

X = df3.drop('BadCredit', axis=1) #.select_dtypes(['number'])
y = df3['BadCredit']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   UserID             6000 non-null   object
 1   Sex                6000 non-null   object
 2   PreviousDefault    6000 non-null   int64
 3   FirstName          6000 non-null   object
 4   LastName           6000 non-null   object
 5   NumberPets         6000 non-null   int64
 6   PreviousAccounts   6000 non-null   int64
 7   ResidenceDuration  6000 non-null   int64
 8   Street             6000 non-null   object
 9   LicensePlate       6000 non-null   object
 10  BadCredit          6000 non-null   int64
 11  Amount             6000 non-null   int64
 12  Married            6000 non-null   int64
 13  Duration           6000 non-null   int64
 14  City               6000 non-null   object
 15  Purpose            6000 non-null   object
 16  DateOfBirth        6000 non-null   object
dtypes: int64(8), object(9)
memory usage: 797.0+ KB
```

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform
```

```
#Pipeline 4

numeric_features = ['Amount', 'Duration', 'NumberPets','Married','ResidenceDuration']

categorical_features = ['City', 'Sex','Purpose'] #Adding a category to purpose

drop_features = ['UserID','FirstName','LastName','Street','LicensePlate']

#DateofBirth Feature Engineering
def get_age_years(feature):
  res = np.array([])
  for instance in feature:
    age = 2021 - int(instance[0:4])
    res = np.append(res, age)
  return res.reshape(-1, 1)


clf = RandomForestClassifier(random_state=42)

numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ])


categorical_transformer = Pipeline(steps=[
      ('encoder', OrdinalEncoder())])

preprocessor4 = Pipeline(steps=[
      ('ct', ColumnTransformer(
          transformers=[
      ('num', numeric_transformer, numeric_features),
      ('cat', categorical_transformer, categorical_features),
      ('age', FunctionTransformer(get_age_years, validate=False), 'DateOfBirth'),
      ('drop', 'drop', drop_features)],
             remainder = 'passthrough',
           sparse_threshold=0)),
           ('feature_selector', SelectKBest(k=10))
    ])


pipe4 = Pipeline(steps=[("preprocessor", preprocessor4), ("clf", clf)])

params = {
    'preprocessor__ct__num__scaler__with_mean': [True, False],
    'preprocessor__ct__num__scaler__with_std': [True, False],
    "clf__criterion": ["gini", "entropy"],
    'clf__class_weight':[None, 'balanced'],
    'clf__max_depth': [None, 3, 10],
    "clf__max_features": uniform(0.0, 1.0),
    }

pipe4 = RandomizedSearchCV(pipe4, params, n_jobs=-1 , scoring='f1_micro', cv=10, verbose=1, return_train_score=True)
```

```
pipe4 = pipe4.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   22.3s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   48.9s finished
```

```
def cv_results_to_df(cv_results):
    results = pd.DataFrame(list(cv_results['params']))
    #results['mean_fit_time'] = cv_results['mean_fit_time']
    #results['mean_score_time'] = cv_results['mean_score_time']
    #results['mean_train_score'] = cv_results['mean_train_score']
    #results['std_train_score'] = cv_results['std_train_score']
    results['mean_val_score'] = cv_results['mean_test_score']
    #results['std_val_score'] = cv_results['std_test_score']
    results['rank_val_score'] = cv_results['rank_test_score']

    results = results.sort_values(['mean_val_score'], ascending=False)
    return results
```

Hyper-parameter tuning results

```
cv_results_to_df(pipe4.cv_results_)
print(pipe4.best_params_)
```

	clf__class_weight	clf__criterion	clf__max_depth	clf__max_features	preprocessor__ct__num__scaler__with_mean	preprocessor__ct__num__scaler__with_std	mean_val_score	rank_val_score
4	None	entropy	10.0	0.262813	True	True	0.885625	1
3	None	gini	10.0	0.842388	True	False	0.879167	2
7	None	entropy	NaN	0.237827	True	True	0.877708	3
9	None	gini	NaN	0.408538	True	False	0.876250	4
5	None	entropy	NaN	0.578849	True	False	0.874375	5
2	balanced	gini	10.0	0.710767	True	True	0.872917	6
6	balanced	entropy	10.0	0.082059	True	False	0.864167	7
0	balanced	gini	3.0	0.225737	False	False	0.835000	8
1	balanced	entropy	3.0	0.249810	False	False	0.835000	8
8	None	entropy	3.0	0.203523	False	False	0.828958	10

```
{'clf__class_weight': None, 'clf__criterion': 'entropy', 'clf__max_depth': 10, 'clf__max_features': 0.26281264394106196, 'preprocessor__ct__num__scaler__with_mean': True, 'preprocessor__ct__num__scaler__with_std': True}
```

The F1 Score increased to 0.8856 when performing Hyperparameter Tuning, with sticking to the same Feature Engineering & Feature Selection

3.5: Performance estimation

```
test_rf = pipe4.predict(X_test)
test_prob_rf=pipe4.predict_proba(X_test)
```

The metrics of the model in production are shown below. The final Accuracy & F1- Score is 0.89, which is in a good range.

```
from sklearn.metrics import accuracy_score, cohen_kappa_score, f1_score, log_loss

print("Accuracy = {:.2f}".format(accuracy_score(y_test, test_rf)))
print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, test_rf)))
print("F1 Score = {:.2f}".format(f1_score(y_test, test_rf,average='micro')))
print("Log Loss = {:.2f}".format(log_loss(y_test, test_prob_rf)))
```

```
Accuracy = 0.89
Kappa = 0.56
F1 Score = 0.89
Log Loss = 0.26
```

```
# Confusion Matrix returns in the format: cm[0,0], cm[0,1], cm[1,0], cm[1,1]: tn, fp, fn, tp

# Sensitivity
def custom_sensitivity_score(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    tn, fp, fn, tp = cm[0][0], cm[0][1], cm[1][0], cm[1][1]
    return (tp/(tp+fn))

# Specificity
def custom_specificity_score(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    tn, fp, fn, tp = cm[0][0], cm[0][1], cm[1][0], cm[1][1]
    return (tn/(tn+fp))

# Positive Predictive Value
def custom_ppv_score(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    tn, fp, fn, tp = cm[0][0], cm[0][1], cm[1][0], cm[1][1]
    return (tp/(tp+fp))

# Negative Predictive Value
def custom_npv_score(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    tn, fp, fn, tp = cm[0][0], cm[0][1], cm[1][0], cm[1][1]
    return (tn/(tn+fn))

# Accuracy
def custom_accuracy_score(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    tn, fp, fn, tp = cm[0][0], cm[0][1], cm[1][0], cm[1][1]
    return ((tn+tp)/(tn+tp+fn+fp))
```

```
print('Metrics of the Random Forest model: \n')

cm = np.transpose(confusion_matrix(y_test, test_rf))
print("Confusion matrix: \n" + str(cm))

print("                                   Accuracy: " + str(custom_accuracy_score(y_test, test_rf)))
print("                   SENSITIVITY (aka RECALL): " + str(custom_sensitivity_score(y_test, test_rf)))
print("                 SPECIFICITY (aka FALL-OUT): " + str(custom_specificity_score(y_test, test_rf)))
print(" POSITIVE PREDICTIVE VALUE, (aka PRECISION): " + str(custom_ppv_score(y_test, test_rf)))
print("                 NEGATIVE PREDICTIVE VALUE): " + str(custom_npv_score(y_test, test_rf)))
```

```
Metrics of the Random Forest model:

Confusion matrix:
[[963 103]
 [ 28 106]]
                                   Accuracy: 0.8908333333333334
                   SENSITIVITY (aka RECALL): 0.5071770334928323
                 SPECIFICITY (aka FALL-OUT): 0.9717457114026236
 POSITIVE PREDICTIVE VALUE, (aka PRECISION): 0.7910447761194029
                 NEGATIVE PREDICTIVE VALUE): 0.9033771106941839
```

Question 4: Uncle Steve's Wind Farm

Instructions

Uncle Steve has invested in wind. He's built a BIG wind farm with a total of 700 turbines. He's been running the farm for a couple of years now and things are going well. He sells the power generated by the farm to the Kingston government and makes a tidy profit. And, of course, he has been gathering data about the turbines' operations.

One area of concern, however, is the cost of maintenece. While the turbines are fairly robust, it seems like one breaks/fails every couple of days. When a turbine fails, it usually costs around $20,000 to repair it. Yikes!

Currently, Uncle Steve is not doing any preventative maintenance. He just waits until a turbine fails, and then he fixes it. But Uncle Steve has recently learned that if he services a turbine *before* it fails, it will only cost around $2,000.

Obviously, there is a potential to save a lot of money here. But first, Uncle Steve would need to figure out *which* turbines are about to fail. Uncle Steve being Uncle Steve, he wants to use ML to build a predictive maintenance model. The model will alert Uncle Steve to potential turbine failures before they happen, giving Uncle Steve a chance to perform an inspection on the turbine and then fix the turbine before it fails. Uncle Steve plans to run the model every morning. For all the turbines that the model predicts will fail, Uncle Steve will order an inspection (which cost a flat $500, no matter if the turbine was in good health or not; the $500 would not be part of the $2,000 service cost). For the rest of the turbines, Uncle Steve will do nothing.

Uncle Steve has used the last few year's worth of operation data to build and assess a model to predict which turbines will fail on any given day. (The data includes useful features like sensor readings, power output, weather, and many more, but those are not important for now.) In fact, he didn't stop there: he built and assessed two models. One model uses using deep learning (in this case, RNNs), and the other uses random forests.

He's tuned the bejeebers out of each model and is comfortable that he has found the best-performing version of each. Both models seem really good: both have accuracy scores > 99%. The RNN has better recall, but Uncle Steve is convinced that the random forest model will be better for him since it has better precision. Just to be sure, he has hired you to double check his calculations.

Your task

Which model will save Uncle Steve more money? Justify.

In addition to the details above, here is the assessment of each model:

- Confusion matrix for the random forest:

	Predicted Fail	Predicted No Fail
Actual Fail	201	55
Actual No Fail	50	255195

- Confusion matrix for the RNN:

	Predicted Fail	Predicted No Fail
Actual Fail	226	30
Actual No Fail	1200	254045

Marking

- *Quality*. Response is well-justified and convincing.
- *Style*. Response uses proper grammar, spelling, and punctuation. Response is clear and professional. Response is complete, but not overly-verbose. Response follows length guidelines.

Tips

- Figure out how much Uncle Steve is currently (i.e., without any predictive maintinance models) paying in maintenance costs.
- Use the information provided above to create a cost matrix.
- Use the cost matrix and the confusion matrices to determine the costs of each model.
- The cost of an inspection is the same, no matter if the turbine is in good condition or is about to fail.
- If the inspection determines that a turbine is about to fail, then it will be fixed right then and there for the additional fee.
- For simplicity, assume the inspections are perfect: i.e., that inspecting a turbine will definitely catch any problems that might exist, and won't accidentally flag an otherwise-healthy turbine.

Answers for Qs 4

Without any predictive maintenance model Uncle Steve is currently losing approximately $ 3.6 Million in repairing the turbines when they break down every couple of days later

- Cost Matrix

	Predicted Fail	Predicted No Fail
Actual Fail	2500	20000
Actual No Fail	500	0

- Total Cost for the random forest: $ 1,627,500

	Predicted Fail	Predicted No Fail
Actual Fail	502,500	1,100,000
Actual No Fail	25,000	0

- Total Cost for the RNN Model: $ 1,765,000

	Predicted Fail	Predicted No Fail
Actual Fail	565,000	600,000
Actual No Fail	600,000	0

Manual Calculations using the Confusion Matrix:

	RF	RNN
Accuracy	0.9996	0.9952
Error	0.0004	0.0048
Sensitivity	0.7852	0.8828
Specificity	0.9998	0.9953
Precision	0.8007	0.1585
F1	0.7928	0.2687

- **Conclusion**:

As seen from the cost calculations above, the Random Forest is giving us a lower cost i.e. 1,627,500 compared to 1,765,000 from the RNN. Hence, we are saving 137,500 from using the predictions on the random forest model.

If we look at the metrics of both models above, RF has a higher F1 score and has far better precision in identifying the true turbines that are going to fail, hence it is saving more money and is less cost-effective.

```
# RF Accuracy

TP= 201
TN= 255195
FP=50
FN=55
print((TP + TN) / (TP + TN + FP + FN))
```

```
0.9995890427043338
```

```
# RF Sensitivity
sensitivity = TP / (FN + TP)

print(sensitivity)
print(1-sensitivity)
```

```
0.78515625
0.21484375
```

```
classification_error = (FP + FN) / (TP + TN + FP + FN)

print(classification_error)
```

```
0.0004109572956661618
```

```
# Specifity RF
specificity = TN / (TN + FP)

print(specificity)
```

```
0.9998041097768811
```

```
#RF precision
precision = TP / (TP + FP)

print(precision)
print(1-precision) #false discovery rate
```

```
0.8007968127490039
0.19920318725099606
```

```
#RF F1
F1 = (2*precision*sensitivity)/(precision+sensitivity)

print(F1)
```

```
0.7928994082840237
```

```
#RNN Accuracy

TP= 226
TN= 254045
FP=1200
FN=30
print((TP + TN) / (TP + TN + FP + FN))
```

```
0.9951859288221964
```

```
# RNN Erro
classification_error = (FP + FN) / float(TP + TN + FP + FN)

print(classification_error)
```

```
0.004814071177803609
```

```
# sensitivity
sensitivity = TP / float(FN + TP)
```

```
print(sensitivity)
print(1-sensitivity) #false negative rate
```

```
0.8828125
0.1171875
```

```
# Specificity
specificity = TN / (TN + FP)

print(specificity)
```

```
0.9952986346451449
```

```
# precision RNN
precision = TP / float(TP + FP)

print(precision)
print(1-precision)
```

```
0.1584852734922861
0.8415147265077139
```

```
#F1 RNN
F1 = (2*precision*sensitivity)/(precision+sensitivity)
print(F1)
```

```
0.26872770511296074
```

✓ 0s completed at 19:58