PythonCodeClustering.md

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt
import numpy as np
from scipy.spatial.distance import squareform
from scipy.cluster.hierarchy import linkage
from wordcloud import WordCloud
from PIL import Image
import matplotlib.cm as cm
import seaborn as sns
import plotly.express as px
from sklearn.metrics import silhouette_score
from sklearn.metrics import silhouette_samples, silhouette_score
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn import preprocessing
import pylab as pl
from sklearn import decomposition
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as hc
from sklearn.decomposition import PCA
from sklearn.cluster import DBSCAN
from plotnine import *
from scipy.cluster.hierarchy import dendrogram
from sklearn.metrics.pairwise import euclidean_distances
```

Load text data and make a word count dataframe using CountVectorizer

```
DF_dairy = pd.read_csv("pubAg/content_dairy.csv")
DF_corn = pd.read_csv("pubAg/content_corn.csv")
DF_farms = pd.read_csv("pubAg/content_farms.csv")
DF_wheat = pd.read_csv("pubAg/content_wheat.csv")
DF_organic_farms = pd.read_csv("pubAg/content_organic_farms.csv")
DF_dairy.insert(0, "Label", "Dairy")
DF_corn.insert(0, "Label", "Corn")
DF_farms.insert(0, "Label", "Farms")
DF_wheat.insert(0, "Label", "Wheat")
DF_organic_farms.insert(0, "Label", "OrganicFarms")


CSV_DF = pd.concat([DF_dairy, DF_corn, DF_farms, DF_wheat, DF_organic_farms], ignore_index=True)
labels = CSV_DF["Label"]


## Instantiate CV
```

MyCV = CountVectorizer(stop_words='english')

```
## It has to be the code below since it encountered several errors


MyCV_content = MyCV.fit_transform(CSV_DF["abstract"].values.astype('U'))
```

Convert DTM to a DF

```
print(MyCV.vocabulary_)
vocab_dict = MyCV.vocabulary_
dict_key = vocab_dict.keys()
print("The vocab is: ", dict_key, "\n\n")
```

NEXT - Use pandas to create data frames

```
Word_count = pd.DataFrame(MyCV_content.toarray(), columns=dict_key)
```

Add the label back to the text data

```
Word_count.insert(0, "Labels", labels)
Word_count.to_csv('pubAg_5content_word_count.csv', index=False)
```

Text Clustering

WordCloud of generating the whole text data

Checking for NaN values

```
CSV_DF.isna().sum()
```

As shown in the result, no NaN values in the "abstract" column

Creating the text variable

```
content_list = CSV_DF["abstract"].tolist()
text = ','.join(str(v) for v in content_list)
```

Creating word_cloud with text as argument in .generate() method

```
word_cloud = WordCloud(collocations=False, background_color='#fbeed1', width=1000, height=600, max_words=100000).gener
```

Display the generated Word Cloud

```
plt.imshow(word_cloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

WordCloud of generating each label cluster

Dairy content dataset

```
content_list = DF_dairy["abstract"].tolist()
text = ','.join(str(v) for v in content_list)
word_cloud = WordCloud(collocations=False, background_color='#fbeed1', width=1000, height=600, max_words=100000).gener
plt.imshow(word_cloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Create a function to generate pretty wordcloud

```
def generate_pretty_wordcloud(data, title, filename):
    location = 'data cleaning/clustering/' + filename
    mask = np.array(Image.open(location))
    content_list = data["abstract"].tolist()
    text = ','.join(str(v) for v in content_list)
    cloud = WordCloud(scale=3,
                      max_words=100000,
                      mask=mask,
                      background_color='#fbeed1',
                      collocations=True,
                      contour_color='#5d0f24',
                      contour_width=2).generate_from_text(text)
    plt.figure(figsize=(10, 8))
    plt.imshow(cloud)
    plt.axis('off')
    plt.title(title, x=0.3, y=0.9, fontname="ITC Officina Sans", fontweight="bold", color="#186101")
    plt.show()
```

Use the function

```
generate_pretty_wordcloud(DF_dairy, "Word Cloud 'DAIRY' content", 'cow.jpg')
generate_pretty_wordcloud(DF_corn, "Word Cloud 'CORN' content", 'corn.png')
generate_pretty_wordcloud(DF_wheat, "Word Cloud 'WHEAT' content", 'Bwheat.png')
generate_pretty_wordcloud(DF_organic_farms, "Word Cloud 'ORGANIC FARMS' content", 'organic.jpg')
generate_pretty_wordcloud(DF_farms, "Word Cloud 'FARMS' content", 'farm.png')
```

Clustering starts here.

First to use Silhouette and Elbow method to find the optimal clusters

Take off the label from Word_count text data

```
labels = Word_count["Labels"]
Word_count.pop("Labels")
```

1.Elbow method

```
SS_dist = []
values_for_k = range(2, 12)
# print(values_for_k)
for k_val in values_for_k:
    print(k_val)
    k_means = KMeans(n_clusters=k_val)
    model = k_means.fit(Word_count)
    SS_dist.append(k_means.inertia_)

print(SS_dist)
print(values_for_k)
plt.figure( facecolor='#fbeed1')
plt.plot(values_for_k, SS_dist)
plt.xlabel('value', fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
plt.ylabel('Sum of squared distances', fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
plt.title('Elbow method for optimal k Choice', fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
plt.style.use('seaborn-darkgrid')
plt.show()
```

2.Silhouette method

```
Sih = []
Cal = []
k_range = range(2, 6)

for k in k_range:
    k_means_n = KMeans(n_clusters=k)
    model = k_means_n.fit(Word_count)
    Pred = k_means_n.predict(Word_count)
    labels_n = k_means_n.labels_
    R1 = metrics.silhouette_score(Word_count, labels_n, metric='euclidean')
    R2 = metrics.calinski_harabasz_score(Word_count, labels_n)
    Sih.append(R1)
    Cal.append(R2)

print(Sih)  ## higher is better
print(Cal)  ## higher is better

fig1, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, facecolor='#fbeed1')
ax1.plot(k_range, Sih)
ax1.set_title("Silhouette", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
ax1.set_xlabel("")
ax2.plot(k_range, Cal)
ax2.set_title("Calinski Harabasz Score", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
ax2.set_xlabel("k values", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
```

Second, use PCA to reduce the dimension of the data

Normalization

Normalize the data before applying the fit method

The method of normalization here is to use the z score

```
Word_count_normalized = (Word_count - Word_count.mean()) / Word_count.std()
print(Word_count_normalized)
print(Word_count_normalized.shape[0])  ## num rows
print(Word_count_normalized.shape[1])  ## num cols
```

```
NumCols = Word_count_normalized.shape[1]
```

Reduce dimensionality

Instantiated my own copy of PCA

```
My_pca = PCA(n_components=2)  ## the two prin columns
```

Transpose it

```
Word_count_normalized = np.transpose(Word_count_normalized)
My_pca.fit(Word_count_normalized)
print(My_pca.components_.T)
# Reformat and view results
Comps = pd.DataFrame(My_pca.components_.T,
                     columns=['PC%s' % _ for _ in range(2)],
                     index=Word_count_normalized.columns)
print(Comps)
print(Comps.iloc[:, 0])
```

2D PCA clusters

```
plt.figure(figsize=(10, 10), facecolor='#fbeed1')
plt.scatter(Comps.iloc[:, 0], Comps.iloc[:, 1], s=20)
plt.xlabel("PC 1", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
plt.ylabel("PC 2", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
plt.title("Scatter Plot Clusters PC 1 and 2", fontsize=15, fontname="ITC Officina Sans", fontweight="bold", color="#72
plt.xlim(-0.03, 0.05)
plt.ylim(-0.04, 0.05)

plt.figure(figsize=(10, 10), facecolor='#fbeed1')
for i, label in enumerate(labels):
    plt.annotate(label, (Comps.iloc[i, 0], Comps.iloc[i, 1]), fontsize=6, fontname="ITC Officina Sans", color="#726abb
    plt.xlim(-0.01, 0.01) #((-0.02, 0.025))
    plt.ylim(-0.01, 0.01) #((-0.02, 0.025))
    plt.xlabel("PC 1", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
    plt.ylabel("PC 2", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
    plt.title("Scatter Plot Clusters PC 1 and 2", fontsize=15, fontname="ITC Officina Sans", fontweight="bold",
              color="#726abb")
plt.show()
```

WARD method

```
EDist = euclidean_distances(Word_count)
print(EDist)
```

DBSCAN

```
yDBSCAN = DBSCAN(eps=6, min_samples=2)
MyDBSCAN.fit_predict(Word_count)
print(MyDBSCAN.labels_)
```

kmeans

k = 4

```
kmeans_object = sklearn.cluster.KMeans(n_clusters=4)
print(kmeans_object)
MyMatrix = Word_count.values
kmeans_object.fit(MyMatrix)
```

Get cluster assignment labels

```
label_k4 = kmeans_object.labels_
print("k-means with k = 4\n", label_k4)
# Format results as a DataFrame
df_k4 = pd.DataFrame([labels, label_k4]).T
print("k means RESULTS\n", df_k4)


print(Euc_dist)
X = Word_count
# sns.set()  #back to defaults
sns.set(font_scale=3)
Z = linkage(squareform(np.around(euclidean_distances(X), 7)))

fig4 = plt.figure(figsize=(5, 5))
ax4 = fig4.add_subplot(111)
dendrogram(Z, ax=ax4)
ax4.tick_params(axis='x', which='major', labelsize=15)
ax4.tick_params(axis='y', which='major', labelsize=15)
# ax5 = fig4.add_subplot(212)
fig4.savefig('exampleSave.png')
```

k = 7

```
kmeans_object1 = sklearn.cluster.KMeans(n_clusters=7)
# print(kmeans_object)

kmeans_object1.fit(MyMatrix)
# Get cluster assignment labels

labels1 = kmeans_object1.labels_
print("K means with k = 2\n", labels1)
# Format results as a DataFrame

Myresults1 = pd.DataFrame([labels, labels1]).T
print("k means RESULTS\n", Myresults1)
```

k = 9

```
kmeans_object2 = sklearn.cluster.KMeans(n_clusters=7)
# print(kmeans_object)

kmeans_object2.fit(MyMatrix)
# Get cluster assignment labels

labels_k9 = kmeans_object1.labels_
print("K means with k = 2\n", labels_k9)
# Format results as a DataFrame
```

```
df_k9 = pd.DataFrame([labels, labels_k9]).T
print("k means RESULTS\n", df_k9)
```

Visualization

PCA

```
My_pca = PCA(n_components=2)  ## the two prin columns
components = My_pca.fit_transform(Word_count)
fig = px.scatter(components, x=0, y=1, color=labels)
#fig.title('PCA Component Clustering', fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
fig.update_layout(
    title='PCA Component Clustering',
    title_font_family="ITC Officina Sans",
    paper_bgcolor='#fbeed1',
    xaxis_title="",
    yaxis_title="",
    font=dict(
        family="ITC Officina Sans",
        color="#726abb", size=20))
fig.for_each_trace(lambda t: t.update(name = '<b>' + t.name +'</b>'))
fig.show()
```

Hierarchical

```
MyHC = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward')
FIT = MyHC.fit(Word_count)
HC_labels = MyHC.labels_
print(HC_labels)

plt.figure(figsize=(10, 10), facecolor='#fbeed1')
plt.title('Hierarchical Clustering', fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
plt.xlabel("Each article abstract content", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
plt.ylabel("", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
plt.style.use('seaborn-darkgrid')
#plt.xaxis.set_visible(False)
dendro = hc.dendrogram(hc.linkage(Word_count, method='ward'))
ax = plt.gca()
plt.grid(True)
ax.axes.xaxis.set_ticks([])
ax.axes.yaxis.set_ticks([])
plt.show()
```