Part 1 - EWCS Dataset (Unsupervised PCA and Clustering)

```
import seaborn as sns import
matplotlib.pyplot as plt import pandas
as pd import numpy as np
```

```
df = pd.read_csv('EWCS_2016.csv')
df.head()
```

```
print('Number of rows:', len(df))
```

```
df.info()
```

```
for col in df.columns:
    print(col, df[col].unique())
# there are some error data (-999) in all cols
```

Checking for erroneous data

```
print('Q2b min', df['Q2b'].describe()['min'])
print('Q2b max', df['Q2b'].describe()['max'])
```

```
print("\033[4mNumber of -999 in each col\033[0m")
for col in df.columns:
    print(col, ":", df[df[col]==-999]['Q2a'].count())
# count number of -999 in each col
```

```
num_neg_infinity = df[df.sum(axis=1) < 0]['Q2a'].count()
print('Number of rows with -999 in at least 1 column:', num_neg_infinity)
```

```
print('Percentage of rows with -999:', round(num_neg_infinity*100/len(df),2),"%")
```

```
# drop rows with -999 as percentage of erroneous data is not
high df = df.drop(df[df.sum(axis=1) < 0].index) print('New
number of rows:', len(df))
```

Data Cleaning

```
# for unscaling MinMax of age in clustering
Q2b_min = df['Q2b'].describe().T['min']
Q2b_max = df['Q2b'].describe().T['max']
max_min_diff = Q2b_max - Q2b_min
```

```
print('Q2b min', df['Q2b'].describe()['min'])
print('Q2b max', df['Q2b'].describe()['max'])
```

```
corr = df.corr() mask =
np.zeros_like(corr)
mask[np.triu_indices_from(mask)]= True
f, ax = plt.subplots(figsize=(15, 15))

heatmap = sns.heatmap(corr,
mask = mask,                       square =
True,                       linewidths = .5,
cmap= 'Blues',                       cbar_kws
= {'shrink': .5,
                                'ticks' : [-1, -.6, 0, 0.5,
1]},                      vmin = -1,
vmax = 1,                       annot = True,
                      annot_kws = {'size': 12})

#add the column names as labels
ax.set_yticklabels(corr.columns, rotation = 0)
ax.set_xticklabels(corr.columns)

sns.set_style({'xtick.bottom': True}, {'ytick.left': True})
```

Exploratory Data Analytics

Observation

- All Q87 are moderate correlated to each other - This shows that individual's life fulfilment tends to be holistic and covers a wide spectrum of their life. It is unlikely that an individual will feel fulfilled in only one segment of life.

- Q90a is moderately positiviely correlated to Q90b - signifying that individual's energy level at work is related to how enthusiastic they are about work. Q90c - weak correlation with
-
- Q90f - weak correlation with Q90a

```
df.hist(bins=30, figsize=(15, 10))
```

Observation
Q90a and Q90b

-
- Similar number of male and females

Ages of individuals seem to follow a right skewed normal distribution, peaking at around 40-45

- For Q87a-Q90c, option 2 (most of the time) is the most common option - Individuals are generally satisfied at work and life
- For Q90f, option 1 (Always) is the most common option - Individuals generally feel that they are good at their job

Checking for skewness for numerical data

```
print('Skewness of Q2b:', df['Q2b'].skew())
```

Skewness of Q2b is low, no additional transformation is required

```
from sklearn.preprocessing import
MinMaxScaler mm = MinMaxScaler() for col in
df.columns:
    df[col] = mm.fit_transform(df[[col]])
```

MinMax scale data

```
df.head()
```

Unsupervised PCA and Clustering Models

- Principal Component Analysis
- K-means clustering
- Hierarchical agglomerative clustering (HAC)
- DBScan
- OPTICS

```
from sklearn.decomposition import PCA

pca_list = list()
feature_weight_list = list()

# Fit a range of PCA models
 for n in range(1,
10):

    # Create and fit the model
    PCAmod = PCA(n_components=n)
    PCAmod.fit(df)

    # Store the model and variance
    pca_list.append(pd.Series({'n':n, 'model':PCAmod,
                               'var': PCAmod.explained_variance_ratio_.sum()}))

    # Calculate and store feature importances
    abs_feature_values = np.abs(PCAmod.components_).sum(axis=0)
feature_weight_list.append(pd.DataFrame({'n':n,
                                         'features': df.columns,

'values':abs_feature_values/abs_feature

pca_var_df = pd.concat(pca_list, axis=1).T.set_index('n')
pca_var_df
```

Principal Component Analysis

In [ ]:

In [ ]:

```
plt.plot(range(1,10), pca_var_df['var'])
plt.xlabel('Number of components')
plt.ylabel('cumulative explained variance')
plt.title('Cumulative explained variance to PCs');
```

In [ ]:

```
n_components = 3
pca = PCA(n_components=n_components)
pca_df = pca.fit_transform(df)
```

```
def myplot(score,coeff,labels=None):
    q = score[:,0]     p =
score[:,1]     n = coeff.shape[0]
scalex = 1/(q.max() - q.min())
scaley = 1/(p.max() - p.min())
plt.figure(figsize=(20, 15))
    plt.scatter(q * scalex,p * scaley,s=25, c='lightblue')
for i in range(n):
        plt.arrow(0, 0, coeff[i,0], coeff[i,1],color = 'red',alpha = 0.6)
if labels is None:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color =
'g           else:
            plt.text(coeff[i,0]* 1.05, coeff[i,1]* 1.05, labels[i], color =
'darkgre      plt.xlabel("PC1")     plt.ylabel("PC2")

plt.grid()
```

In [ ]:

```
plot1 = [0,2]
myplot(pca_df[:,plot1],np.transpose(pca.components_[plot1,:]),list(df.columns))
plt.title('PCA biplot', fontsize = 30) plt.show()
```

```
feature_df = (pd.concat(feature_weight_list)
               .pivot(index='n', columns='features', values='values'))

feature_df # how much each feature contribute to each component
```

PCA Feature importance

In [ ]:

In [ ]:

```
ax = feature_df[0:3].plot(kind='bar',
figsize=(12,4)) ax.legend(loc='upper right')
ax.set(xlabel='Number of dimensions',
ylabel='Relative importance',
       title='Feature importance vs Dimensions');
```

Observation

- Qn2a has the greatest feature importance in PC1 - PC3.
- Q2b has relatively low feature importance across PC1 - PC4.
- Q90b and Q90c have high feature importance in PC4 and PC5.
- Q90f has low feature importance across all PCs.

```
# function for jitter scatter plot for clustering visuals
def rand_jitter(array):
    sd = 0.01 * (max(array) - min(array))
```

Model visualizations evaluation

In [ ]:

```
    return array + np.random.randn(len(array)) * sd
 def jitter(x, y, s=20, c='b',
marker='o'):
    from matplotlib.colors import ListedColormap
scatter = plt.scatter
    colours = ListedColormap(['r','b','g','y'])
plt.figure(figsize=(5,2))
    return  scatter(rand_jitter(x),  rand_jitter(y),  s=s,  c=c,  marker=marker,
alpha=0
```

In [ ]:

```
from timeit import default_timer as timer # Calcuate time elapsed in training
 model
```

K Means Clustering

In [ ]:

```
from sklearn.cluster import
 KMeans

inertia = [] # inertia is the
list_num_clusters = list(range(1,11))
for num_clusters in list_num_clusters:
km = KMeans(n_clusters=num_clusters)
km.fit(pca_df)
    inertia.append(km.inertia_)

plt.figure(figsize=(5,2))
plt.plot(list_num_clusters,inertia)
plt.scatter(list_num_clusters,inertia)
plt.title('Elbow method')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia');
```

```
In [ ]:
start = timer()
cluster_num = 4
km = KMeans(n_clusters=cluster_num,random_state=24,n_init=3) # n_init, number of
tim km.fit(pca_df) df['k_means'] = km.predict(pca_df)

end = timer()
k_means_time = end - start print('Time
Elapsed (sec):', k_means_time)
```

```
In [ ]: jitter(x=rand_jitter(df['Q2a']),
         y=rand_jitter(df['Q90a']),c=df['k_means']) plt.xticks([0, 1])
         plt.xlabel('Gender') plt.ylabel('Enthusiasm')
```

```
In [ ]:
In [ ]:
clusters = [0,1,2,3]

number_of_individuals = []
for clus in clusters:
     count = df[df['k_means'] == clus]['Q2a'].count()
number_of_individuals.append(count)

proportion_of_male = []
for clus in clusters:
     proportion = df[df['k_means'] == clus]['Q2a'].mean()
proportion_of_male.append(proportion)

average_age = []
for clus in
clusters:
     avg = df[df['k_means'] == clus]['Q2b'].mean()
avg = (avg*max_min_diff)+Q2b_min
average_age.append(avg)

avg_Q87a = [] for
clus in clusters:
     avg = df[df['k_means'] == clus]['Q87a'].mean()
avg = avg*6
     avg_Q87a.append(avg)

avg_Q90a = [] for
clus in clusters:
     avg = df[df['k_means'] ==
clus]['Q90a'].mean()      avg = avg*5
avg_Q90a.append(avg)
```

```
pd.DataFrame({'Number of individuals': number_of_individuals, 'Proportion of
males':
```

Hierarchical Agglomerative Clustering Model

In [ ]:

```
from sklearn.cluster import AgglomerativeClustering

start = timer()
num_clusters = 4
ag = AgglomerativeClustering(n_clusters=num_clusters, linkage='ward',
compute_full_t ag = ag.fit(pca_df) df['agglom'] = ag.fit_predict(pca_df)

end = timer()
HAC_time = end - start
print('Time Elapsed (sec):', HAC_time)
```

In [ ]:

```
from scipy.cluster import hierarchy

Z = hierarchy.linkage(ag.children_, method='ward')
fig, ax = plt.subplots(figsize=(10,5))

den = hierarchy.dendrogram(Z, orientation='top',
                           p=4, truncate_mode='level', # p=30,
truncate_mode='lastp'                show_leaf_counts=True, ax=ax)
```

In [ ]:

```
jitter(x=rand_jitter(df['Q2a']),
  y=rand_jitter(df['Q90a']),c=df['agglom']) plt.xlabel('Gender')
  plt.ylabel('Q87b') plt.title('HAC clustering')
```

In [ ]:

```
clusters = [0,1,2,3]

number_of_individuals = []
for clus in clusters:
     count = df[df['agglom'] ==
clus]['Q2a'].count()
number_of_individuals.append(count)
proportion_of_male = []
for clus in clusters:
     proportion = df[df['agglom'] == clus]['Q2a'].mean()
proportion_of_male.append(proportion)

average_age = []
for clus in
clusters:
     avg = df[df['agglom'] == clus]['Q2b'].mean()
avg = (avg*max_min_diff)+Q2b_min
average_age.append(avg)

avg_Q87a = [] for
clus in clusters:
     avg = df[df['agglom'] == clus]['Q87a'].mean()
avg = avg*6
     avg_Q87a.append(avg)

avg_Q90a = [] for
clus in clusters:
```

```
pd.DataFrame({'Number of individuals': number_of_individuals, 'Proportion of
males':
```

```
      avg = df[df['agglom'] ==
 clus]['Q90a'].mean()      avg = avg*5
 avg_Q90a.append(avg)
```

In [ ]:

In [ ]:

```
# Comparing with KMeans results:
(df[['agglom','k_means']]
 .groupby(['agglom','k_means'])
 .size()
 .to_frame()
 .rename(columns={0:'count'}))
```

```
from sklearn.neighbors import NearestNeighbors
neighbor = NearestNeighbors(n_neighbors=2)
numbers = neighbor.fit(pca_df)
distances, indices = numbers.kneighbors(pca_df)

distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.ylabel('Eps')
plt.xlabel('Distance')
plt.plot(distances)
```

DBScan Clustering

In [ ]:

In [ ]:

In [ ]:

```
from sklearn.cluster import DBSCAN
from collections import Counter

num_clusters = []
min_samples = [] for
min_sam in range(0,75):
    min_samples.append(min_sam)
    dbscan = DBSCAN(eps=0.2, min_samples=min_sam, metric='euclidean')
num_clus = len(Counter(dbscan.fit_predict(pca_df)))
num_clusters.append(num_clus)

min_sam_clust = pd.DataFrame({'Min Samples':min_samples, 'Number of
clusters':num_cl
plt.plot(min_sam_clust['Min Samples'], min_sam_clust['Number of
clusters']) plt.xlabel('Min_samples') plt.ylabel('Number of clusters')
```

In [ ]:

```
dbscan = DBSCAN(eps=0.05, min_samples=20, metric='euclidean')
df['DBScan_unoptimized'] = dbscan.fit_predict(pca_df)
print('Number of clusters:',
df['DBScan_unoptimized'].nunique())
```

```
In [ ]: start = timer()
dbs = DBSCAN(eps=0.5, min_samples=20,
metric='euclidean') df['DBScan_optimized'] =
dbs.fit_predict(pca_df)
print('Number of clusters:',
df['DBScan_optimized'].nunique())

end = timer()
DBScan_time = end - start print('Time Elapsed (sec):',
DBScan_time)
```

```
In [ ]:
fig, axs = plt.subplots(2, figsize=(20,10))
fig.suptitle('Vertically stacked subplots')
axs[0].scatter(x=pca_df[:,0], y=pca_df[:,2],
c=df['DBScan_unoptimized']) axs[0].set_title('DBScan - Eps = 0.05')
axs[1].scatter(x=pca_df[:,0], y=pca_df[:,2], c=df['DBScan_optimized'])
axs[1].set_title('DBScan - Eps = 0.2')
```

```
In [ ]: clusters = [0,1,-1]

number_of_individuals = []
for clus in clusters:
     count = df[df['DBScan_optimized'] == clus]['Q2a'].count()
number_of_individuals.append(count)

proportion_of_male = []
for clus in clusters:
     proportion = df[df['DBScan_optimized'] == clus]['Q2a'].mean()
proportion_of_male.append(proportion)

average_age = [] for
clus in clusters:
     avg = df[df['DBScan_optimized'] == clus]['Q2b'].mean()
avg = (avg*max_min_diff)+Q2b_min
average_age.append(avg)

avg_Q87a = [] for
clus in clusters:
     avg = df[df['DBScan_optimized'] == clus]['Q87a'].mean()
avg = avg*6
     avg_Q87a.append(avg)

avg_Q90a = [] for
clus in clusters:
     avg = df[df['DBScan_optimized'] ==
clus]['Q90a'].mean()      avg = avg*5
avg_Q90a.append(avg)
```

```
In [ ]: pd.DataFrame({'Number of individuals': number_of_individuals, 'Proportion of
          males':
```

```
In [ ]:

jitter(x=rand_jitter(df['Q2a']),
y=rand_jitter(df['Q90f']),c=df['DBScan_optimized']) plt.xlabel('Gender')
plt.ylabel('Q87b') plt.title('DBScan clustering')
```

```
from sklearn.cluster import OPTICS, cluster_optics_dbscan

start = timer() opt = OPTICS(min_samples=10, xi=.02,
min_cluster_size=0.1) df['optics'] =
opt.fit_predict(pca_df)

end = timer()
OPTICS_time = end - start
print('Time Elapsed (sec):', OPTICS_time)
print('Numbe of clusters:',df['optics'].nunique())
```

OPTICS Clustering

In [ ]:

```
In [ ]: plt.figure(figsize=(20,10)) plt.scatter(x=pca_df[:,0],
        y=pca_df[:,2], c=df['optics'])
```

In [ ]:
```
clusters = [0,1,-1]

number_of_individuals = []
for clus in clusters:
     count = df[df['optics'] == clus]['Q2a'].count()
number_of_individuals.append(count)

proportion_of_male = []
for clus in clusters:
     proportion = df[df['optics'] == clus]['Q2a'].mean()
proportion_of_male.append(proportion)

average_age = [] for
clus in clusters:
     avg = df[df['optics'] == clus]['Q2b'].mean()
avg = (avg*max_min_diff)+Q2b_min
average_age.append(avg)

avg_Q87a = [] for
clus in clusters:
     avg = df[df['optics'] ==
clus]['Q87a'].mean()      avg = avg*6
avg_Q87a.append(avg)

avg_Q90a = [] for
clus in clusters:
     avg = df[df['optics'] ==
clus]['Q90a'].mean()      avg = avg*5
avg_Q90a.append(avg)
```

In [ ]:
```
pd.DataFrame({'Number of individuals': number_of_individuals, 'Proportion of
males':
```

In [ ]:
```
jitter(x=rand_jitter(df['Q2a']), y=rand_jitter(df['Q87b']),c=df['optics'])
plt.xlabel('Gender')
```

```
time_elapsed = [k_means_time, HAC_time, DBScan_time, OPTICS_time]

labels = ['k means', 'HAC', 'DBScan', 'OPTICS']

eval_df = pd.DataFrame({'Time Elapsed':time_elapsed},index=labels)
eval_df
```

Elapsed time of all models

In [ ]:

In [ ]:

```
plt.ylabel('Q87b')
plt.title('OPTICS clustering')
```

In [ ]:

```
fig, axs = plt.subplots(4,2,figsize=(12,12))
fig.suptitle('Vertically stacked subplots')
axs[0,0].scatter(x=pca_df[:,0], y=pca_df[:,1], c=df['k_means'])
axs[0,0].set_title('K Means - PC1 and PC2')
axs[0,1].scatter(x=pca_df[:,0], y=pca_df[:,2], c=df['k_means'])
axs[0,1].set_title('K Means - PC1 and PC3')
axs[1,0].scatter(x=pca_df[:,0], y=pca_df[:,1], c=df['agglom'])
axs[1,0].set_title('HAC - PC1 and PC2')
axs[1,1].scatter(x=pca_df[:,0], y=pca_df[:,2], c=df['agglom'])
axs[1,1].set_title('HAC - PC1 and PC3')
axs[2,0].scatter(x=pca_df[:,0], y=pca_df[:,1], c=df['DBScan_optimized'])
axs[2,0].set_title('DBScan - PC1 and PC2')
axs[2,1].scatter(x=pca_df[:,0], y=pca_df[:,2], c=df['DBScan_optimized'])
axs[2,1].set_title('DBScan - PC1 and PC3')
axs[3,0].scatter(x=pca_df[:,0], y=pca_df[:,1], c=df['optics'])
axs[3,0].set_title('OPTICS - PC1 and PC2')
axs[3,1].scatter(x=pca_df[:,0], y=pca_df[:,2], c=df['optics'])
axs[3,1].set_title('OPTICS - PC1 and PC3')
```