BDC Final Preperation

pd.read\_csv('xyz.csv') to read csv

pd.read\_excel('xyz.xlsx') to read excel file (xlsx)

df.shape to get tuple like (rows no, columns no)

df.head(5) to get top 5 rows

df.tail(5) to get last 5 rows

df['Column\_name'].count() or df.shape[0] to get total number of that rows

df.isnull().sum() # Get the number of missing values for each column

df['column\_name'].value\_counts() list out unique value in column and it’s count

num\_duplicates = df.duplicated().sum() to get total number of duplicates

df.drop\_duplicates(inplace=True) to drop duplicates

df.dropna(inplace=True) to drop missing value

df.rename(columns={'old\_column\_name': 'new\_column\_name', 'old\_column\_name2': 'new\_column\_name2'}, inplace=True) to rename columns

df.dtypes to see the types of attributes in df

Just examples to create new column with arithmetic operation:

order['Total Sale'] = order['Quantity'] \* order['Price']

emp\_attendance['total'] = emp\_attendance[['Day1', 'Day2', 'Day3', 'Day4', 'Day5', 'Day6', 'Day7', 'Day8', 'Day9', 'Day10', 'Day11', 'Day12', 'Day13', 'Day14']].sum(axis=1)

to get index of max value row

index = df['numerical\_column'].idxmax()

to get vale of column\_2 at index given

df.loc[index, 'column\_2']

to df by multiple column values

df.sort\_values(by=['numerical column 1', 'nc2'], ascending=[False,False])

to derive three top(max) values

top\_three\_values = df['Column1'].nlargest(3)

example for with groupby operation:

product\_max\_sale\_3 = order.groupby('Product Name')['Total Sale'].sum().nlargest(3)

plt.barh(product\_max\_sale\_3.index, product\_max\_sale\_3.values).index to get index values, .values to get numerical values

In pandas, the .unstack() method is used in group operations to pivot a level of a MultiIndex (hierarchical index) from the innermost level to become the new columns in a DataFrame. # Usefull in indexing task to convert inner level index as a new column

df.groupby('Column\_name')['numerical\_column'].mean() to get list of unique values of column and average of numerical column for that unique values

new\_df = df[df['column\_name'] == 'match\_value'] to derive records only that matches condition

max\_value\_record = df[df[numerical\_Column] == df[' numerical\_Column '].max()] to get maximum value containing row

single\_value = max\_value\_record['column\_name'].values[0] to get value of desired column of derived record

float\_variable:.2f to get float variable value until 2 units after point

import matplotlib.pyplot as plt for graph purpose

plt.plot(x, y, label='label'): Create a line plot.

plt.scatter(x, y, label='label'): Create a scatter plot.

plt.bar(x, height, width=0.8, label='label'): Create a bar plot.

plt.hist(data, bins, label='label'): Create a histogram.

plt.pie(sizes, labels=labels, autopct='%1.1f%%'): Create a pie chart.

plt.barh(x,y)

plt.xticks(rotation='vertical') to show xlabels vertically

plt.xlabel('xlabel') and plt.ylabel('ylabel'): Set labels for x and y axes.

plt.title('title'): Set the title of the plot.

plt.legend(): Display the legend.

To display subplots example:

plt.figure(figsize=(12, 5)) # Adjust the width and height as needed

plt.subplot(1,2,1)

plt.bar(gender\_wages.columns,gender\_wages.values[0],color='red')

plt.title("Mean Wage for Female")

plt.ylabel('Mean Wage (RM)')

plt.xlabel('Year')

plt.subplot(1,2,2)

plt.bar(gender\_wages.columns,gender\_wages.values[1],color='blue')

plt.title("Mean Wage for male")

plt.xlabel('Year')

plt.tight\_layout()

plt.show()

for re operations:

import re

pattern = r’\bxyz\b’ to find xyz in text

replaced\_text = re.sub(pattern, 'abc', text) to replace xyz with abc in text

pattern = r'\[\d+\]' to find pattern like [5] (citation)

text.replace(pattern,'',inplace=True, regex=True) to replace that patter with nothing ‘ ’

NUmpy :

import numpy as np

To convert raw file into numpyarrat:

file\_path = "Attendance\_RAW.txt"

attendance\_data = np.genfromtxt(file\_path, delimiter=',',dtype=str)

To reshape the structure of numpy array:

rows = 15

column = 14

na = na.reshape(rows, column)

to get list of sum of all columns in na(Numpy array):

na = na.astype(int)

list\_of\_sum\_all\_column = np.sum(na, axis=0)

list\_of\_sum\_all\_column = list\_of\_sum\_all\_column.tolist()

to create narray:

my\_array = np.array([[33, 22, 11], [55, 66, 44], [99, 98, 77]])

to sort array:

sorted\_array = np.sort(my\_array)

sum of all elements:

total\_sum = np.sum(sorted\_array)

sum of each column and sum of each row:

column\_sums = np.sum(sorted\_array, axis=0)

row\_sums = np.sum(sorted\_array, axis=1)

for Mean:

total\_mean = np.mean(sorted\_array)

column\_means = np.mean(sorted\_array, axis=0)

row\_means = np.mean(sorted\_array, axis=1)

cumulative sum of each column and cumulative product of each row:

colcumsum = np.cumsum(sorted\_array, axis=0)

row\_cumprod = np.cumprod(sorted\_array, axis=1)

to convert nparray object to df:

df = pd.DataFrame(nparray, columns=[list of columns in np object])

example:

pd.DataFrame(attendance\_data, columns=['Day1', 'Day2', 'Day3', 'Day4', 'Day5', 'Day6', 'Day7', 'Day8', 'Day9', 'Day10', 'Day11', 'Day12', 'Day13', 'Day14'])

To fill Missing values of column with it’s mean:

mean\_of\_c1 = df['c1'].mean()

df['c1'].fillna(mean\_of\_c1, inplace=True)

or example if you want to fill with any value ex ‘1’ just place 1 in param:

df['c1'].fillna(1, inplace=True)

To fill Missing values of column with forward fill and backward fill method:

df['c1'].fillna(method='ffill', inplace=True)

df['c1'].fillna(method='bfill', inplace=True)

to drop rows that contains less then specified threshold for values should be in a row:

df.dropna(thresh=4 <- you can put any number here that you atleast want, inplace=True)

Example use of pd.cut:

bins\_age = [20, 30, 40, 50, 60, 70, 80, 90]

age\_labels = ['20-29', '30-39', '40-49', '50-59', '60-69', '70-79', '80-89']

diabetes\_df['Age\_Category'] = pd.cut(diabetes\_df['Age'], bins=bins\_age, labels=age\_labels, right=False)

To merge dfs:

Df = pd.merge(df1,df2, on=[name of same columns])

Ex: merged\_df = pd.merge(births\_df, deaths\_df, on=['Year', 'State', 'Sex'])

to set index of df:

df.set\_index(['Year', 'State'])

ex to use .agg() method with group by:

grouped\_data = merged\_df.groupby('State').agg({

'Number of Live births': 'sum',

'Number of death': 'sum',

}).reset\_index()

grouped\_data = grouped\_data.set\_index(['State'])

get and set value from df in loop using .at():

value = df.at[row\_label, column\_label] and df.at[row\_label, column\_label] = value

example to get the list of index of missing values of column:

missing\_City = tallest[tallest['City'].isnull()].index.tolist()

For clustering task just write step of importing kmeans:

from sklearn.cluster import KMeans

.iterrows() method use for df example:

for index, row in df.iterrows():

print(f"Index: {index}, Name: {row['Name']}, Age: {row['Age']}, City: {row['City']}")