BDA Lab

visualisation

```
import plotly.express as px
df = px.data.iris()
fig = px.line(df, y="sepal_width",)
fig.show()

fig = px.line(df, y="sepal_width", line_group='species')
fig.show()

df = px.data.tips()
# Creating the bar chart
fig = px.bar(df, x='day', y="total_bill")
fig.show()


df = px.data.tips()
# plotting the histogram
fig = px.histogram(df, x="total_bill")
fig.show()


df = px.data.tips()
# plotting the histogram
fig = px.histogram(df, x="total_bill", color='sex', nbins=5
0, histnorm='percent',barmode='overlay')
fig.show()

df = px.data.tips()
fig = px.pie(df, values="total_bill", names="day")
fig.show()
```

```
df = px.data.tips()
fig = px.box(df, x="day", y="tip")
fig.show()

df = px.data.tips()
# plotting the violin plot
fig = px.violin(df, x="day", y="tip")
fig.show()



df = px.data.tips()
fig = px.scatter_3d(df, x="total_bill", y="sex", z="tip")
fig.show()
```

Hierchical

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering

customer_data = pd.read_csv('hierarchical-clustering.csv')
#customerid, gender, age, income, spending score

data = customer_data.iloc[:, 3:5].values  #selecting annual income and spending score

plt.figure(figsize=(10, 7))
plt.title("Customer Dendrogram")
shc.dendrogram(shc.linkage(data, method='ward'))
plt.show()



cluster = AgglomerativeClustering(n_clusters=5, metric='euclidean', linkage='ward')
# affinity = 'euclidean'
```

```
labels_ = cluster.fit_predict(data)

plt.figure(figsize=(10, 7))
plt.scatter(data[:, 0], data[:, 1], c=labels_, cmap='rainbo
w')
plt.xlabel("Annual Income")
plt.ylabel("Spending Score")
plt.title("Clusters of Customers")
plt.show()
```

DBSCAN

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN

df = pd.read_csv('hierarchical-clustering.csv')

df = df.dropna(subset=['Annual Income (k$)','Spending Score
(1-100)'])

df=df[['Annual Income (k$)','Spending Score (1-100)']].valu
es

plt.scatter(df[:,0], df[:,1], s=10, c="black")
plt.title("Scatter Plot of Data")
plt.xlabel("Annual Income")
plt.ylabel("Spending Score")
plt.show()



wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters= i,init = 'k-means++', max_i
ter= 300, n_init= 10,random_state=42)
```

```
    kmeans.fit(df)
    wcss.append(kmeans.inertia_)


plt.plot(range(1,11), wcss,marker='o',linestyle='--')
plt.title("The Elbow Method")
plt.xlabel("Number of clusters")
plt.ylabel("WCSS")
plt.grid()
plt.show()


dbscan = DBSCAN(eps=5, min_samples=5)
labels = dbscan.fit_predict(df)
print(np.unique(labels))

unique_labels = np.unique(labels)
colors = ['blue', 'red', 'green', 'brown', 'pink', 'yello
w', 'silver']

# Loop over unique labels and assign colors to clusters
for label in unique_labels:
    # For noise points (-1), color them black
    if label == -1:
        plt.scatter(df[labels == label, 0], df[labels == la
bel, 1], s=10, c='black', label='Noise')
    else:
        # For other clusters, assign colors from the list
        color = colors[label % len(colors)]  # Ensures it w
orks if there are more than 6 clusters
        plt.scatter(df[labels == label, 0], df[labels == la
bel, 1], s=10, c=color, label=f'Cluster {label}')

# Adding labels and legend
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.legend()
plt.show()
```

OPTICS

```
from sklearn.datasets import make_blobs
from sklearn.cluster import OPTICS
import numpy as np
import matplotlib.pyplot as plt



X, y = make_blobs(n_samples=1000, centers=[(3, 3), (7, 7)],
n_features=2, cluster_std=0.5, random_state=42)

# Apply OPTICS clustering
db = OPTICS(max_eps=2.0, min_samples=22, cluster_method='x
i', metric='minkowski').fit(X)
labels = db.labels_
no_clusters = len(np.unique(labels[labels != -1]))  # Exclu
de noise points
no_noise = np.sum(labels == -1)

print(f'Estimated no. of clusters: {no_clusters}')
print(f'Estimated no. of noise points: {no_noise}')

# Plot clustering results
plt.figure(figsize=(8, 6))
colors = ['blue' if label == 0 else 'red' if label == 1 els
e 'gray' for label in labels]
plt.scatter(X[:, 0], X[:, 1], c=colors, marker="o")
plt.title('OPTICS clustering')
plt.xlabel('Axis X[0]')
plt.ylabel('Axis X[1]')
plt.show()

# Generate reachability plot
plt.figure(figsize=(8, 6))
plt.plot(db.reachability_[db.ordering_])
plt.title('Reachability plot')
plt.xlabel('Sample Index')
plt.ylabel('Reachability Distance')
plt.show()
```

printing sales

```
#!/usr/bin/env python
import sys
import csv

# Read input from standard input
reader = csv.reader(sys.stdin, delimiter=',')  # Comma-sepa
rated values

for line in reader:
    if len(line) < 3:
        continue  # Skip malformed lines

    year = line[0]  # First column is the year
    sales = line[-1]  # Last column is sales

    try:
        print(f"{year}\t{int(sales)}")  # Ensure sales is a
n integer
    except ValueError:
        continue  # Skip invalid sales values



-----------------------------------------------------------
-----------------
#!/usr/bin/env python
import sys

current_year = None
total_sales = 0

for line in sys.stdin:
    try:
        year, sales = line.strip().split('\t')
        sales = int(sales)  # Convert sales to an integer
    except ValueError:
        continue  # Skip lines with invalid values

    if current_year is None:
        current_year = year

    if year == current_year:
        total_sales += sales
    else:
        print(f"{current_year}\t{total_sales}")
        current_year = year
        total_sales = sales

# Print the last year’s total
if current_year is not None:
    print(f"{current_year}\t{total_sales}")
```

Inverted

```
import sys

def mapper():
    for line in sys.stdin:
        line = line.strip()
        doc, *words = line.split()
        for word in words:
            print(f"{word}\t{doc}")

if __name__ == "__main__":
    mapper()

----------------------------------------------------------

import sys

```

```
def reducer():
    current_word, doc_list = None, set()

    for line in sys.stdin:
        word, doc = line.strip().split("\t")

        if word == current_word:
            doc_list.add(doc)
        else:
            if current_word:
                print(f"{current_word}: {', '.join(sorted(d
oc_list))}")
            current_word, doc_list = word, {doc}

    if current_word:
        print(f"{current_word}: {', '.join(sorted(doc_lis
t))}")

if __name__ == "__main__":
    reducer()

-----------------------------------------------------------
-
doc1 apple banana
doc2 banana orange
doc3 apple mango
doc4 banana apple

apple: doc1, doc3, doc4
banana: doc1, doc2, doc4
orange: doc2
mango: doc3
```