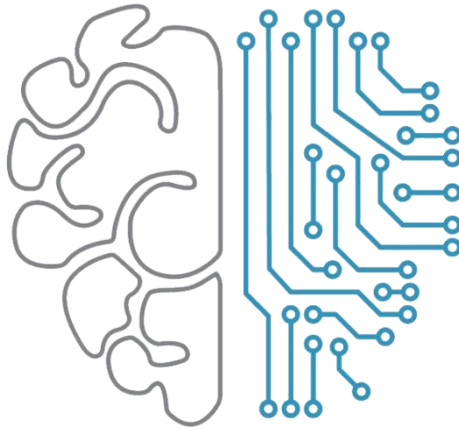


LAPORAN PRAKTIKUM

DASAR PEMROGRAMAN



INTELLIGENT **COMPUTING**
LABORATORY

NAMA : Dino Adianto Silalahi

NIM : 202331193

KELAS : D

DOSEN : DR. DRA. DWINA KUSWARDANI, M.KOM

NO.PC : 28

ASISTEN : 1. VIANA SALSABILA FAIRUZ SYAHLA

2. MUHAMMAD FADEL DESYAPUTRA

3. FAUZAN ARROYAN

4. CLARENCA SWEETDIVA PEREIRA

INSTITUT TEKNOLOGI PLN

TEKNIK INFORMATIKA

2024

Screenshot hasil praktikum K-Means

Jupyter Praktikum K-means Last Checkpoint: 4 days ago

File Edit View Run Kernel Settings Help Trusted

JupyterLab Python 3 (ipykernel)

Import Library

```
[1]: import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns
```

Memuat Dataset

```
[2]: data = pd.read_csv('sales.csv')
print("Dataset Preview:")
print(data.head())
```

Dataset Preview:

	Product_Code	W0	W1	W2	W3	W4	W5	W6	W7	W8	...	Normalized 42 \
0	P1	11	12	10	8	13	12	14	21	6	...	0.06
1	P2	7	6	3	2	7	1	6	3	3	...	0.20
2	P3	7	11	8	9	10	8	7	13	12	...	0.27
3	P4	12	8	13	5	9	6	9	13	13	...	0.41
4	P5	8	5	13	11	6	7	9	14	9	...	0.27

	Normalized 43	Normalized 44	Normalized 45	Normalized 46	Normalized 47 \
0	0.22	0.28	0.39	0.50	0.00
1	0.40	0.50	0.10	0.10	0.40
2	1.00	0.18	0.18	0.36	0.45
3	0.47	0.06	0.12	0.24	0.35
4	0.53	0.27	0.60	0.20	0.20

	Normalized 48	Normalized 49	Normalized 50	Normalized 51
0	0.22	0.17	0.11	0.39
1	0.50	0.10	0.60	0.00
2	1.00	0.45	0.45	0.36
3	0.71	0.35	0.29	0.35
4	0.13	0.53	0.33	0.40

[5 rows x 107 columns]

Informasi Dataset

```
[3]: print("\nInformasi Dataset:")
print(data.info())
```

Jupyter Praktikum K-means Last Checkpoint: 4 days ago

File Edit View Run Kernel Settings Help Trusted

JupyterLab Python 3 (ipykernel)

Informasi Dataset:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 811 entries, 0 to 810
Columns: 107 entries, Product_Code to Normalized 51
dtypes: float64(52), int64(54), object(1)
memory usage: 678.1+ KB
None
```

Memeriksa dataset

```
[4]: print(data.columns)
```

Index(['Product_Code', 'W0', 'W1', 'W2', 'W3', 'W4', 'W5', 'W6', 'W7', 'W8',
...,
'Normalized 42', 'Normalized 43', 'Normalized 44', 'Normalized 45',
'Normalized 46', 'Normalized 47', 'Normalized 48', 'Normalized 49',
'Normalized 50', 'Normalized 51'],
dtype='object', length=107)

```
[5]: print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 811 entries, 0 to 810
Columns: 107 entries, Product_Code to Normalized 51
dtypes: float64(52), int64(54), object(1)
memory usage: 678.1+ KB
None
```

Pemilihan Fitur

```
[6]: selected_features = ['W0', 'W1']
data_selected = data[selected_features]
```

Normalisasi data

```
[7]: scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data_selected)
```

Metode Elbow

```
[8]: inertia = []
K = range(1, 11)
```

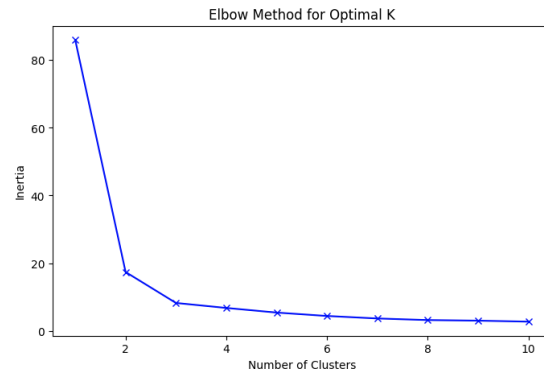
```
[9]: for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_normalized)
```

```
[8]: inertia = []
    K = range(1, 11)

[9]: for k in K:
      kmeans = KMeans(n_clusters=k, random_state=42)
      kmeans.fit(data_normalized)
      inertia.append(kmeans.inertia_)
```

Visualisasi metode Elbow

```
[10]: plt.figure(figsize=(8, 5))
      plt.plot(K, inertia, 'bx-')
      plt.xlabel('Number of Clusters')
      plt.ylabel('Inertia')
      plt.title('Elbow Method for Optimal K')
      plt.show()
```



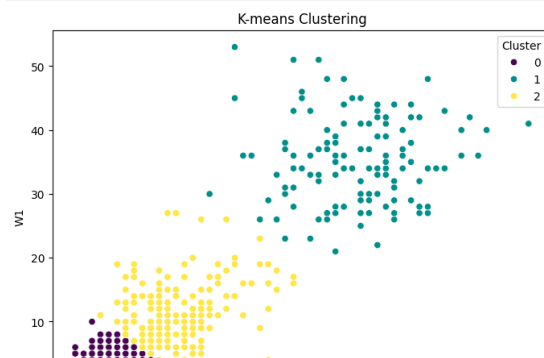
Clustering Dengan K optimal

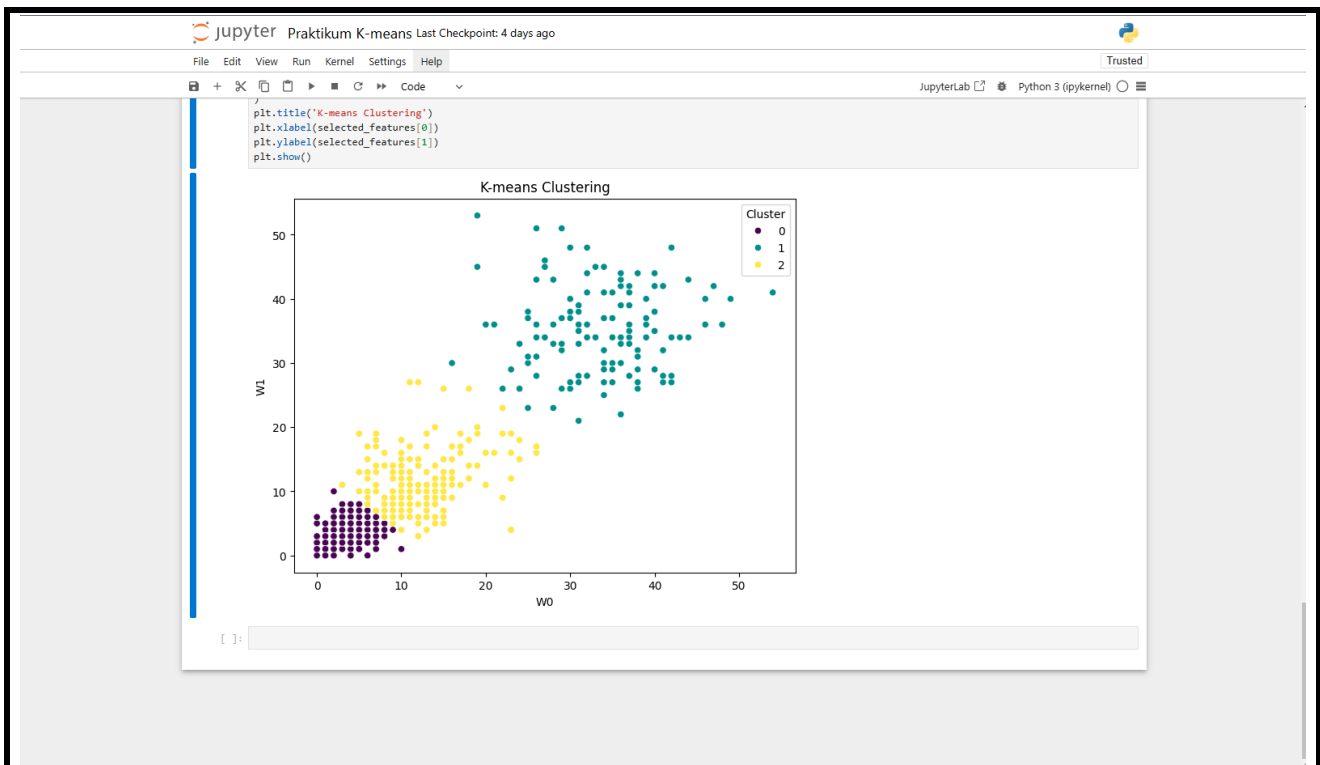
Clustering Dengan K optimal

```
[11]: optimal_clusters = 3
      kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
      data['Cluster'] = kmeans.fit_predict(data_normalized)
```

Visualisasi Hasil Clustering

```
[12]: plt.figure(figsize=(8, 6))
      sns.scatterplot(
          x=data_selected.iloc[:, 0],
          y=data_selected.iloc[:, 1],
          hue=data['Cluster'],
          palette='viridis'
      )
      plt.title('K-means Clustering')
      plt.xlabel(selected_features[0])
      plt.ylabel(selected_features[1])
      plt.show()
```





Penjelasan

1. Import Library

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt

import seaborn as sns

- Mengimpor library yang diperlukan:
 - pandas untuk pengolahan data.
 - numpy untuk operasi numerik.
 - KMeans dari sklearn untuk algoritma clustering.
 - MinMaxScaler untuk normalisasi data.
 - matplotlib dan seaborn untuk visualisasi.

2. Memuat Dataset

```
data = pd.read_csv('sales.csv')
```

```
print("Dataset Preview:")
```

```
print(data.head())
```

- Membaca file dataset sales.csv ke dalam dataframe data.
- Menampilkan beberapa baris pertama dataset dengan data.head().

3. Informasi Dataset

```
print("\nInformasi Dataset:")
```

```
print(data.info())
```

- Menampilkan informasi tentang dataset, seperti tipe data setiap kolom dan jumlah nilai yang tidak kosong (non-null).

4. Memeriksa Kolom

```
print(data.columns)
```

- Menampilkan daftar nama kolom di dataset.

5. Pemilihan Fitur

```
selected_features = ['W0', 'W1']
```

```
data_selected = data[selected_features]
```

- Memilih dua kolom W0 dan W1 sebagai fitur yang akan digunakan dalam algoritma clustering.

6. Normalisasi Data

```
scaler = MinMaxScaler()
```

```
data_normalized = scaler.fit_transform(data_selected)
```

- Menggunakan MinMaxScaler untuk menormalisasi data sehingga nilainya berada dalam rentang 0 hingga 1. Hal ini penting untuk memastikan setiap fitur memiliki kontribusi yang seimbang dalam algoritma K-means.

7. Metode Elbow

```
inertia = []
```

```
K = range(1, 11)
```

```
for k in K:
```

```
    kmeans = KMeans(n_clusters=k, random_state=42)
```

```
    kmeans.fit(data_normalized)
```

```
    inertia.append(kmeans.inertia_)
```

- Menggunakan metode Elbow untuk menentukan jumlah cluster yang optimal.
- inertia mencatat total jarak kuadrat antara setiap titik data dan pusat cluster-nya untuk setiap nilai k.

8. Visualisasi Metode Elbow

```
plt.figure(figsize=(8, 5))
```

```
plt.plot(K, inertia, 'bx-')
```

```
plt.xlabel('Number of Clusters')
```

```
plt.ylabel('Inertia')
```

```
plt.title('Elbow Method for Optimal K')
```

```
plt.show()
```

- Membuat grafik metode Elbow untuk mengidentifikasi jumlah cluster yang optimal berdasarkan perubahan inertia.

9. Clustering dengan K Optimal

```
optimal_clusters = 3
```

```
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
```

```
data['Cluster'] = kmeans.fit_predict(data_normalized)
```

- Menentukan jumlah cluster optimal (dalam contoh, $k=3$).
- Melakukan clustering menggunakan K-means dan menambahkan hasil cluster ke dataframe data.

10. Visualisasi Hasil Clustering

```
plt.figure(figsize=(8, 6))

sns.scatterplot(

    x=data_selected.iloc[:, 0],

    y=data_selected.iloc[:, 1],

    hue=data['Cluster'],

    palette='viridis'

)

plt.title('K-means Clustering')

plt.xlabel(selected_features[0])

plt.ylabel(selected_features[1])

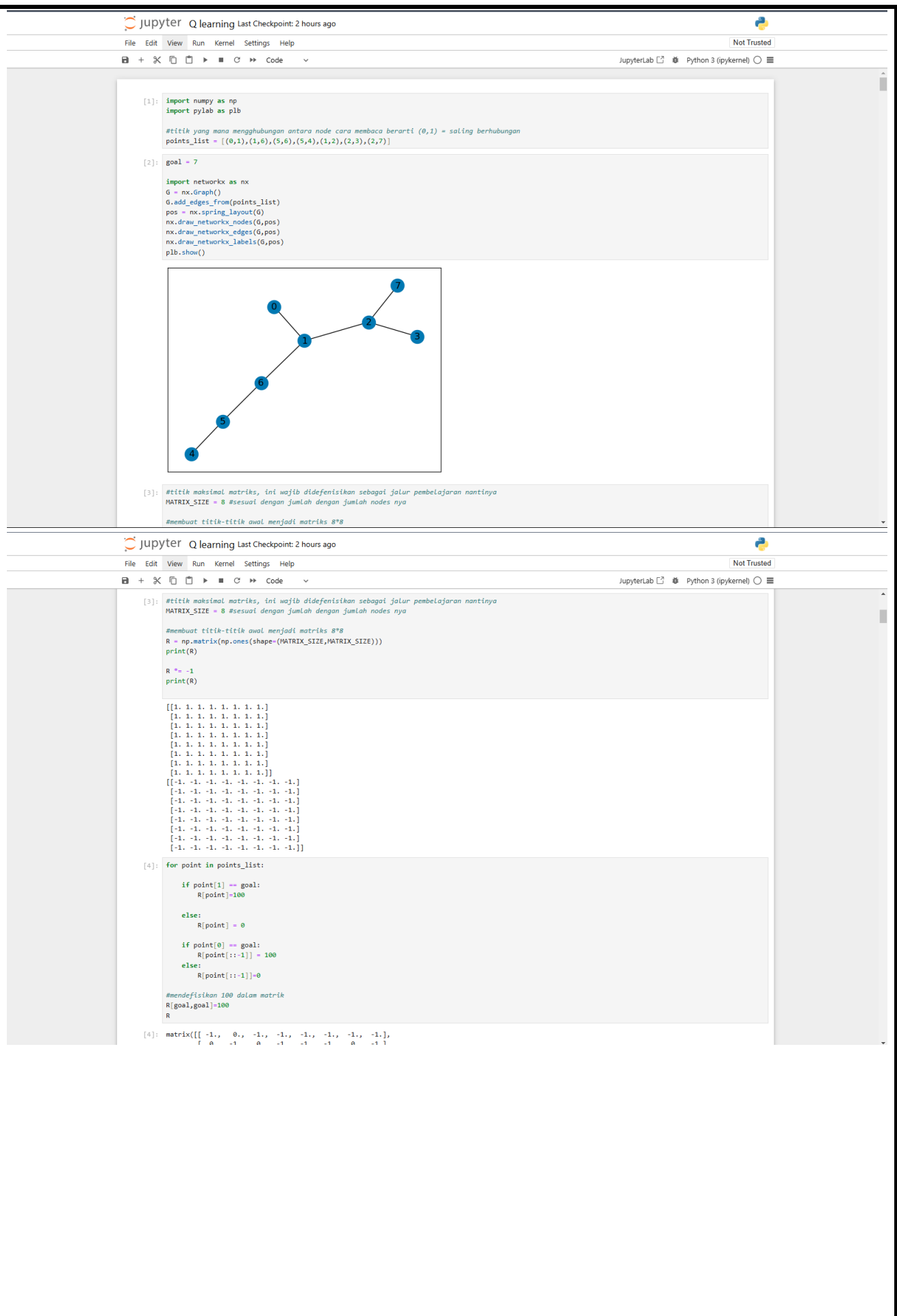
plt.show()
```

- Membuat visualisasi scatter plot untuk menunjukkan hasil clustering. Data diwarnai berdasarkan cluster.

Penjelasan mengenai dataset sales.csv

Dataset **sales.csv** terdiri dari 811 baris dan 107 kolom. Berikut adalah ringkasan strukturnya:

- **Kolom-kolom:**
 - Ada satu kolom dengan tipe data objek bernama **Product_Code**, yang kemungkinan merupakan pengenal unik untuk produk.
 - Sisanya adalah kolom numerik (52 tipe float64 dan 54 tipe int64), yang kemungkinan berisi data kuantitatif terkait dengan karakteristik atau performa produk.
- **Contoh Data:** Berikut adalah beberapa kolom dari dataset:
 - **W0 – W51:** Kolom-kolom ini tampaknya merepresentasikan angka atau metrik tertentu untuk setiap produk.
 - **Normalized 0 - Normalized 51:** Kolom ini mungkin hasil normalisasi dari data mentah.



Jupyter

Q learning Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help

Not Trusted

JupyterLab Python 3 (ipykernel)

```
[4]: matrix([[ -1.,  0., -1., -1., -1., -1., -1., -1.],
[ 0., -1.,  0., -1., -1., -1.,  0., -1.],
[ -1.,  0., -1.,  0., -1., -1., -1., 100.],
[ -1., -1.,  0., -1., -1., -1., -1., -1.],
[ -1., -1., -1., -1., -1.,  0., -1., -1.],
[ -1., -1., -1., -1.,  0., -1.,  0., -1.],
[ -1.,  0., -1., -1., -1.,  0., -1., -1.],
[ -1., -1.,  0., -1., -1., -1., -1., 100.]])

[5]: Q = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))

# Learning parameter
gamma = 0.8

initial_state = 1

#Aksi yang akan diambil dari vector
def available_actions(state):
    current_state_row = R[state,]
    av_act = np.where(current_state_row >= 0)[1]
    return av_act

available_act = available_actions(initial_state)

#Mesin memutuskan untuk mesin selanjutnya
def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_act,1))
    return next_action

action = sample_next_action(available_act)

#menyimpan hasil dari langkah yang diambil
def update(current_state, action, gamma):

    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]

    Q[current_state, action] = R[current_state, action] + gamma * max_value
    print('max_value', R[current_state, action] + gamma * max_value)

    if (np.max(Q) > 0):

Q = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))

# Learning parameter
gamma = 0.8

initial_state = 1

#Aksi yang akan diambil dari vector
def available_actions(state):
    current_state_row = R[state,]
    av_act = np.where(current_state_row >= 0)[1]
    return av_act

available_act = available_actions(initial_state)

#Mesin memutuskan untuk mesin selanjutnya
def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_act,1))
    return next_action

action = sample_next_action(available_act)

#menyimpan hasil dari langkah yang diambil
def update(current_state, action, gamma):

    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]

    Q[current_state, action] = R[current_state, action] + gamma * max_value
    print('max_value', R[current_state, action] + gamma * max_value)

    if (np.max(Q) > 0):
        return(np.sum(Q/np.max(Q)*100))
    else:
        return (0)

update(initial_state, action, gamma)

max_value 0.0

[5]: 0
```

Jupyter

Q learning Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help

Not Trusted

JupyterLab Python 3 (ipykernel)

```
[5]: Q = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))

# Learning parameter
gamma = 0.8

initial_state = 1

#Aksi yang akan diambil dari vector
def available_actions(state):
    current_state_row = R[state,]
    av_act = np.where(current_state_row >= 0)[1]
    return av_act

available_act = available_actions(initial_state)

#Mesin memutuskan untuk mesin selanjutnya
def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_act,1))
    return next_action

action = sample_next_action(available_act)

#menyimpan hasil dari langkah yang diambil
def update(current_state, action, gamma):

    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]

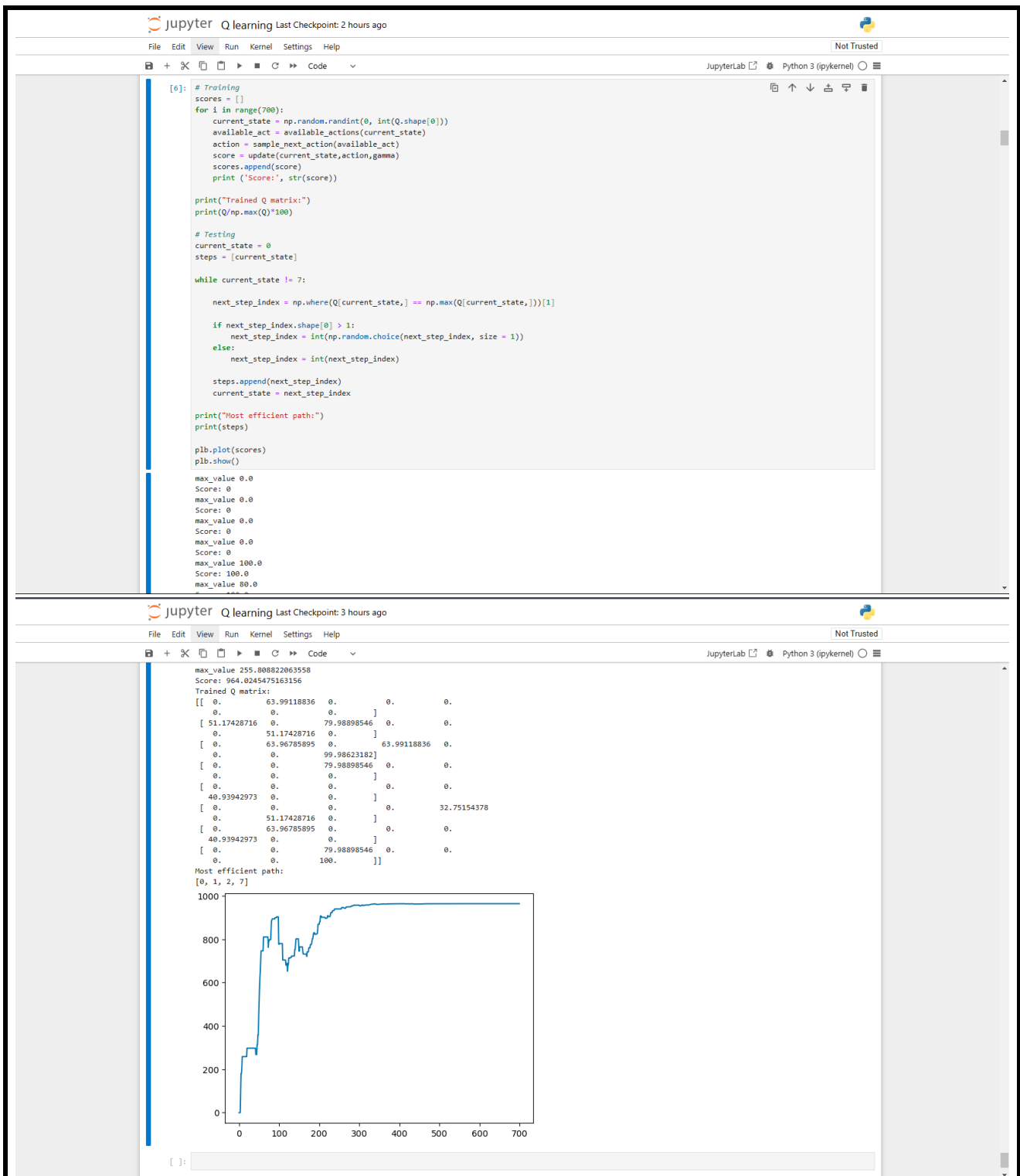
    Q[current_state, action] = R[current_state, action] + gamma * max_value
    print('max_value', R[current_state, action] + gamma * max_value)

    if (np.max(Q) > 0):
        return(np.sum(Q/np.max(Q)*100))
    else:
        return (0)

update(initial_state, action, gamma)

max_value 0.0

[5]: 0
```



Penjelasan

1. Import Library dan Definisi Titik

import numpy as np

import pylab as plt

```
# Titik-titik yang saling terhubung (0,1) berarti node 0 terhubung ke node 1
```

```
points_list = [(0,1),(1,6),(5,6),(5,4),(1,2),(2,3),(2,7)]
```

- Mengimpor numpy untuk operasi numerik dan pylab untuk visualisasi.
- Mendefinisikan daftar hubungan antar node dalam bentuk pasangan koordinat (graph).

2. Visualisasi Graph

```
goal = 7
```

```
import networkx as nx
```

```
G = nx.Graph()
```

```
G.add_edges_from(points_list)
```

```
pos = nx.spring_layout(G)
```

```
nx.draw_networkx_nodes(G, pos)
```

```
nx.draw_networkx_edges(G, pos)
```

```
nx.draw_networkx_labels(G, pos)
```

```
plt.show()
```

- Menggunakan networkx untuk membangun dan memvisualisasikan graph.
- goal = 7 adalah node tujuan dalam graph.
- Visualisasi graph dilakukan dengan posisi node dihitung menggunakan spring_layout.

3. Matriks Reward Awal

```
MATRIX_SIZE = 8 # Jumlah node
```

```
R = np.matrix(np.ones(shape=(MATRIX_SIZE, MATRIX_SIZE))) * -1
```

```
print(R)
```

- Membuat matriks reward berukuran 8x8 (sesuai jumlah node), dengan nilai awal -1 untuk semua elemen.

4. Mengatur Nilai Reward

```
for point in points_list:
```

```
if point[1] == goal:
```

```
    R[point] = 100
```

```
else:
```

```
    R[point] = 0
```

```
if point[0] == goal:
```

```
    R[point[:-1]] = 100
```

```
else:
```

```
    R[point[:-1]] = 0
```

```
R[goal, goal] = 100
```

```
R
```

- Mengatur nilai reward:
 - Jika suatu node terhubung langsung ke node tujuan (goal), maka reward-nya adalah 100.
 - Hubungan lain diatur menjadi 0.
 - Node tujuan memiliki reward 100 untuk dirinya sendiri.

5. Inisialisasi Matriks Q dan Parameter

```
Q = np.matrix(np.zeros([MATRIX_SIZE, MATRIX_SIZE]))
```

```
# Parameter pembelajaran
```

```
gamma = 0.8
```

```
initial_state = 1
```

- Membuat matriks Q berukuran sama dengan matriks R, diinisialisasi dengan nilai 0.
- gamma adalah faktor diskon, memengaruhi seberapa jauh reward masa depan dipertimbangkan.
- initial_state adalah node awal.

6. Fungsi untuk Menentukan Aksi

```
def available_actions(state):
```

```
    current_state_row = R[state, ]
```

```
    av_act = np.where(current_state_row >= 0)[1]
```

```
    return av_act
```

```
def sample_next_action(available_actions_range):
```

```
    next_action = int(np.random.choice(available_actions_range, 1))
```

```
    return next_action
```

- available_actions: Menentukan aksi yang tersedia dari suatu state berdasarkan matriks R.
- sample_next_action: Memilih aksi berikutnya secara acak dari daftar aksi yang tersedia.

7. Fungsi Update Q-matrix

```
def update(current_state, action, gamma):
```

```
    max_index = np.where(Q[action, ] == np.max(Q[action, ]))[1]
```

```
    if max_index.shape[0] > 1:
```

```
        max_index = int(np.random.choice(max_index, size=1))
```

```
    else:
```

```
        max_index = int(max_index)
```

```
    max_value = Q[action, max_index]
```

```
    Q[current_state, action] = R[current_state, action] + gamma * max_value
```

```
    if np.max(Q) > 0:
```

```
        return np.sum(Q / np.max(Q) * 100)
```

```
    else:
```

```
        return 0
```

- Fungsi ini memperbarui nilai matriks Q berdasarkan aturan Q-learning:
$$Q[s,a]=R[s,a]+\gamma \cdot \max_{a'}(Q[s',a'])$$
$$Q[s, a] = R[s, a] + \gamma \cdot \max(Q[s', a'])$$
- Menggunakan nilai maksimal dari Q untuk state berikutnya sebagai bagian dari perhitungan.

8. Training Model

```
scores = []
```

```
for i in range(700):
```

```
    current_state = np.random.randint(0, Q.shape[0])
```

```
    available_act = available_actions(current_state)
```

```
    action = sample_next_action(available_act)
```

```
    score = update(current_state, action, gamma)
```

```
    scores.append(score)
```

```
print("Trained Q matrix:")
```

```
print(Q / np.max(Q) * 100)
```

- Model dilatih selama 700 iterasi.
- Di setiap iterasi, dipilih state dan aksi secara acak, lalu matriks Q diperbarui.
- Hasil matriks Q dilatih diubah ke skala persentase.

9. Testing dan Jalur Efisien

```
current_state = 0
```

```
steps = [current_state]
```

```
while current_state != 7:
```

```
    next_step_index = np.where(Q[current_state, ] == np.max(Q[current_state, ]))[1]
```

```
    if next_step_index.shape[0] > 1:
```

```
        next_step_index = int(np.random.choice(next_step_index, size=1))
```

```
    else:
```

```
        next_step_index = int(next_step_index)
```

```
    steps.append(next_step_index)
```

```
    current_state = next_step_index
```

```
print("Most efficient path:")
```

```
print(steps)
```

- Menentukan jalur paling efisien dari node awal ke node tujuan (7) berdasarkan matriks Q yang telah dilatih.

10. Visualisasi Skor

```
plb.plot(scores)
```

```
plb.show()
```

- Membuat grafik perubahan skor selama proses pelatihan untuk menunjukkan peningkatan kinerja.

Screenshot pengisian Form

