

Nama : Az – Zahra Chikal E.

NIM : 1103213039

Kelas : TK-45-05

TUGAS WEEK 10 MLP REGRESSION

Link Google Colab (Apabila yang di zip tidak dapat dibuka):

https://colab.research.google.com/drive/10sDpbGwZo45_24RGP3aZr06ekno1Qrc6?usp=sharing

```
# Import Library
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, accuracy_score
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
✓ 0.0s Python

# Load the dataset
df = pd.read_csv("bike_sharing_hour.csv")
df.head(5)
✓ 0.0s Python


| instant | dteday | season     | yr | mnth | hr | holiday | weekday | workingday | weathersit | temp | atemp | hum    | windspeed | casual | registered | cnt |    |
|---------|--------|------------|----|------|----|---------|---------|------------|------------|------|-------|--------|-----------|--------|------------|-----|----|
| 0       | 1      | 2011-01-01 | 1  | 0    | 1  | 0       | 0       | 6          | 0          | 1    | 0.24  | 0.2879 | 0.81      | 0.0    | 3          | 13  | 16 |
| 1       | 2      | 2011-01-01 | 1  | 0    | 1  | 1       | 0       | 6          | 0          | 1    | 0.22  | 0.2727 | 0.80      | 0.0    | 8          | 32  | 40 |
| 2       | 3      | 2011-01-01 | 1  | 0    | 1  | 2       | 0       | 6          | 0          | 1    | 0.22  | 0.2727 | 0.80      | 0.0    | 5          | 27  | 32 |
| 3       | 4      | 2011-01-01 | 1  | 0    | 1  | 3       | 0       | 6          | 0          | 1    | 0.24  | 0.2879 | 0.75      | 0.0    | 3          | 10  | 13 |
| 4       | 5      | 2011-01-01 | 1  | 0    | 1  | 4       | 0       | 6          | 0          | 1    | 0.24  | 0.2879 | 0.75      | 0.0    | 0          | 1   | 1  |


✓ 0.0s Python

df.info()
✓ 0.0s Python

df.isnull().sum()
✓ 0.0s Python
```

Output:

instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt	
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   instant     17379 non-null   int64  
 1   dteday      17379 non-null   object 
 2   season      17379 non-null   int64  
 3   yr          17379 non-null   int64  
 4   mnth        17379 non-null   int64  
 5   hr          17379 non-null   int64  
 6   holiday     17379 non-null   int64  
 7   weekday     17379 non-null   int64  
 8   workingday  17379 non-null   int64  
 9   weathersit  17379 non-null   int64  
 10  temp         17379 non-null   float64 
 11  atemp        17379 non-null   float64 
 12  hum          17379 non-null   float64 
 13  windspeed    17379 non-null   float64 
 14  casual       17379 non-null   int64  
 15  registered   17379 non-null   int64  
 16  cnt          17379 non-null   int64  
dtypes: float64(4), int64(12), object(1)
memory usage: 2.3+ MB
instant          0
dteday           0
season           0
yr               0
mnth             0
hr               0
holiday          0
weekday          0
workingday       0
weathersit       0
temp              0
atemp             0
hum               0
windspeed         0
casual            0
registered        0
cnt               0
dtype: int64

```

Analisis:

Kode di atas digunakan untuk memuat library untuk analisis data, pembelajaran mesin, dan visualisasi, seperti pandas, PyTorch, dan seaborn. Dataset bike_sharing_hour.csv diimpor menggunakan pd.read_csv(), dan metode seperti .head(), .info(), dan .isnull().sum() digunakan untuk mengeksplorasi data, termasuk menampilkan 5 baris pertama, informasi kolom, tipe data, dan jumlah nilai kosong.

```

# Statistik Deskriptif
df.describe()
✓ 0.0s
Python

df = df.drop_duplicates()
df.info()
✓ 0.0s
Python

```

Output:

	instant	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
count	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000
mean	8690.0000	2.501640	0.502561	6.537775	11.546752	0.028770	3.003683	0.682721	1.425283	0.496987	0.475775	0.627229	0.190098	35.676218	153.766869	189.463088
std	5017.0295	1.106918	0.500088	3.438776	6.914405	0.167165	2.05771	0.465431	0.89357	0.192556	0.17850	0.192930	0.122340	49.305030	151.357286	181.387599
min	1.0000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.020000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	4345.5000	2.000000	0.000000	4.000000	6.000000	0.000000	1.000000	0.000000	1.000000	0.340000	0.333300	0.480000	0.104500	4.000000	34.000000	40.000000
50%	8690.0000	3.000000	1.000000	7.000000	12.000000	0.000000	3.000000	1.000000	0.500000	0.484800	0.630000	0.194000	17.000000	115.000000	142.000000	
75%	13034.5000	3.000000	1.000000	10.000000	18.000000	0.000000	5.000000	1.000000	2.000000	0.660000	0.621200	0.780000	0.253700	48.000000	220.000000	281.000000
max	17379.0000	4.000000	1.000000	12.000000	23.000000	1.000000	6.000000	1.000000	4.000000	1.000000	1.000000	0.850700	367.000000	886.000000	977.000000	

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 17 columns):
 #   Column   Non-Null Count Dtype  
--- 
 0   instant    17379 non-null   int64  
 1   dteday     17379 non-null   object  
 2   season     17379 non-null   int64  
 3   yr          17379 non-null   int64  
 4   mnth        17379 non-null   int64  
 5   hr          17379 non-null   int64  
 6   holiday    17379 non-null   int64  
 7   weekday    17379 non-null   int64  
 8   workingday 17379 non-null   int64  
 9   weathersit 17379 non-null   int64  
 10  temp        17379 non-null   float64 
 11  atemp       17379 non-null   float64 
 12  hum          17379 non-null   float64 
 13  windspeed   17379 non-null   float64 
 14  casual      17379 non-null   int64  
 15  registered  17379 non-null   int64  
 16  cnt          17379 non-null   int64  
dtypes: float64(4), int64(12), object(1)
memory usage: 2.3+ MB

```

Analisis:

Kode tersebut menghapus duplikasi data dalam dataset menggunakan `df.drop_duplicates()` dan mengecek kembali struktur data dengan `df.info()`, yang memberikan informasi seperti jumlah kolom, tipe data, dan jumlah entri non-kosong. Selanjutnya, `df.describe()` digunakan untuk menghitung statistik deskriptif seperti rata-rata, standar deviasi, nilai minimum, dan maksimum untuk kolom numerik, yang membantu memahami distribusi data.

```

# Convert date column to datetime
df['dteday'] = pd.to_datetime(df['dteday'])

# Add new feature: hour of the day
if 'hr' in df.columns:
    df['hour'] = df['hr']

# Distribusi data untuk setiap kolom numerik
numerical_cols = ['temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt']
for col in numerical_cols:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[col], kde=True, bins=30, color='blue')
    plt.title(f"Distribusi {col}")
    plt.show()

```

Python

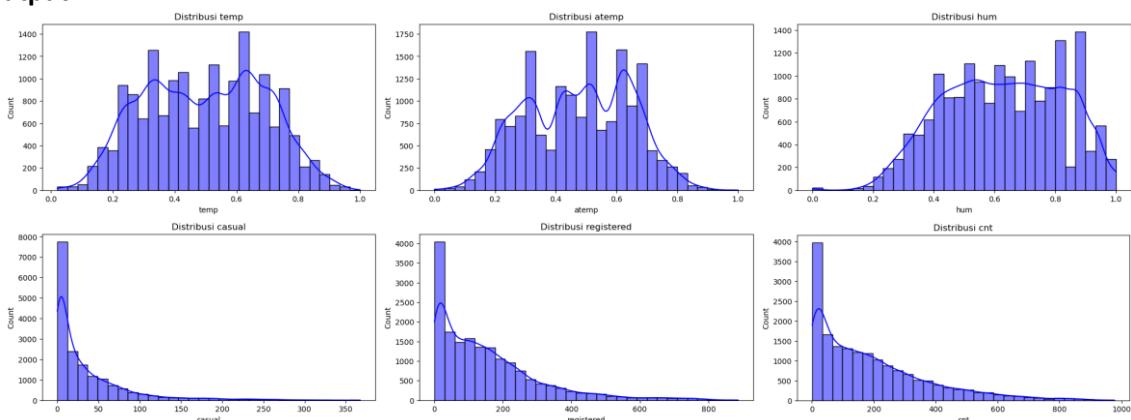
```

# Distribusi data untuk setiap kolom numerik
numerical_cols = ['temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt']
for col in numerical_cols:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[col], kde=True, bins=30, color='blue')
    plt.title(f"Distribusi {col}")
    plt.show()

```

Python

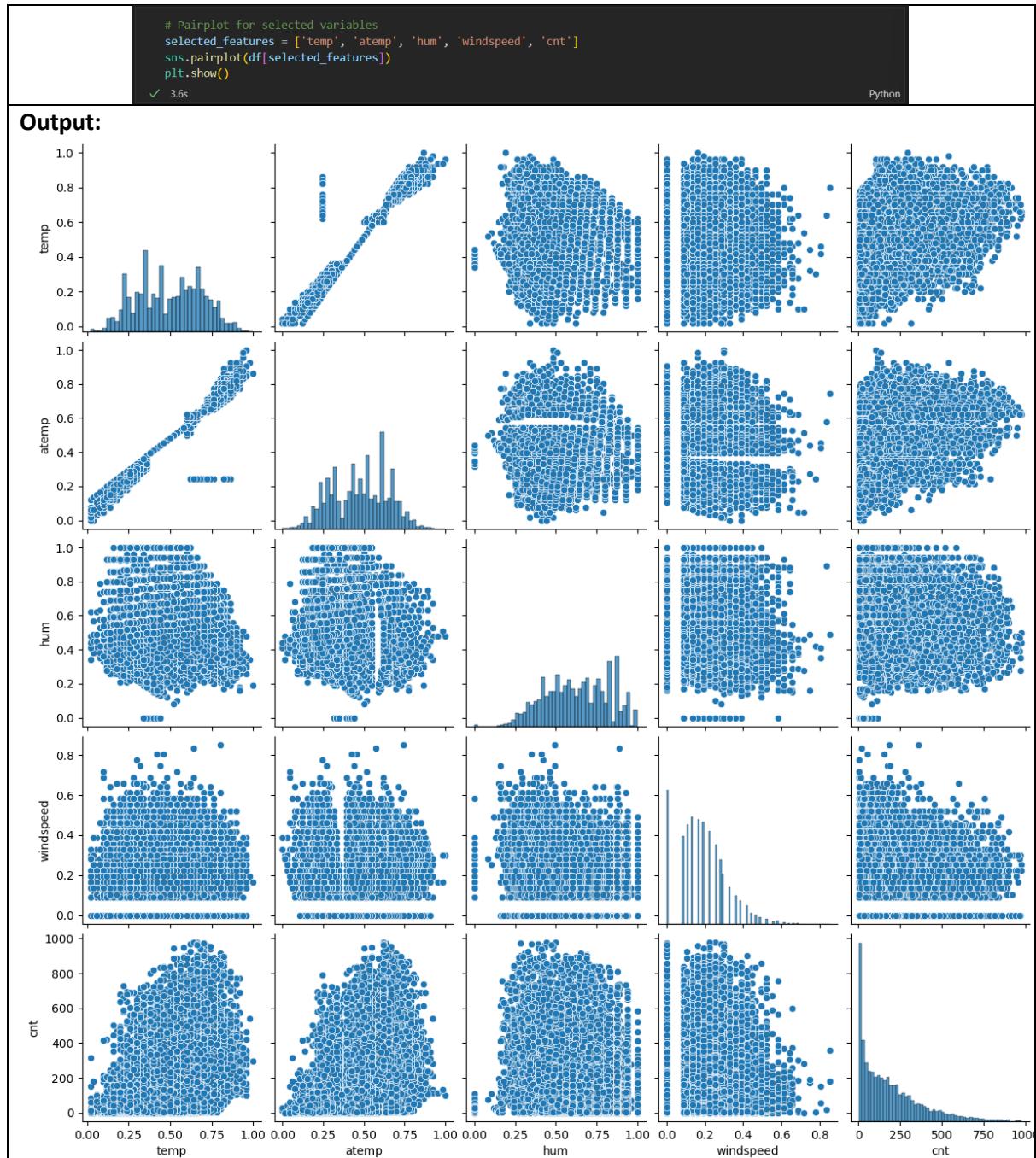
Output:



Analisis:

Kode di atas mengonversi kolom `dteday` menjadi tipe `datetime` menggunakan `pd.to_datetime()` untuk memastikan format tanggal sesuai. Kemudian, jika kolom `hr` ada dalam dataset, kolom baru

hour ditambahkan untuk merepresentasikan jam dalam sehari. Selanjutnya, untuk memahami distribusi data kolom numerik seperti temp, atemp, hum, dan lainnya, dilakukan visualisasi menggunakan histogram dengan kernel density estimation (KDE) menggunakan sns.histplot(). Plot ini memberikan gambaran tentang distribusi data, pola, dan variasi setiap kolom.



Analisis:

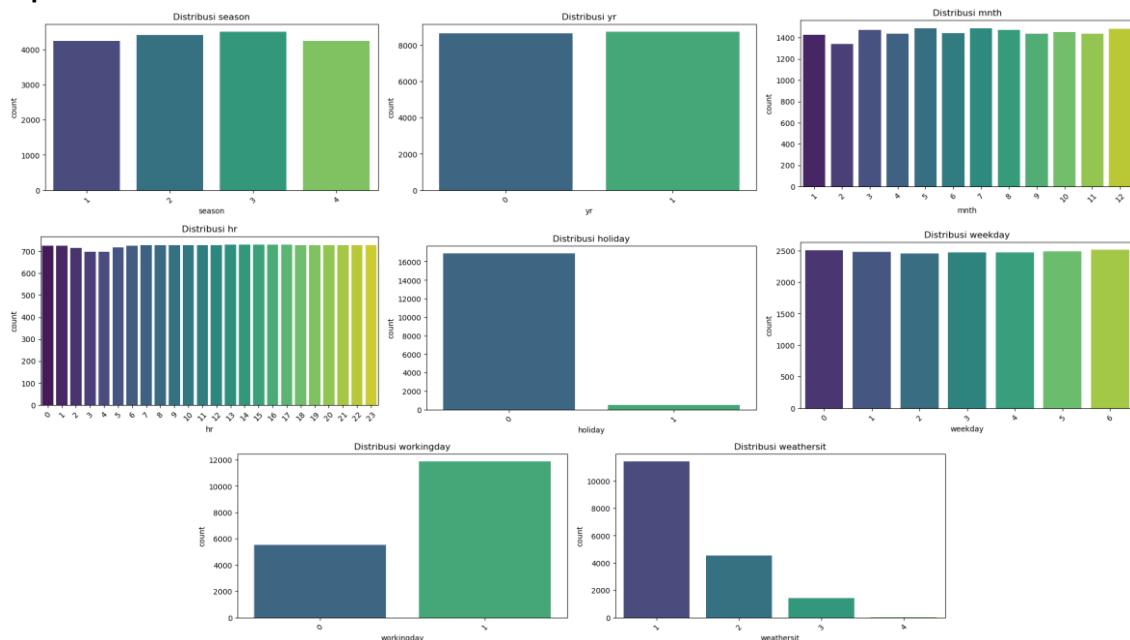
Kode tersebut menggunakan seaborn's pairplot untuk memvisualisasikan hubungan antar variabel dalam dataset, khususnya pada kolom terpilih: temp, atemp, hum, windspeed, dan cnt. Plot ini menghasilkan pasangan grafik scatterplot untuk setiap kombinasi variabel, serta distribusi univariat di sepanjang diagonal. Tujuan visualisasi ini adalah untuk memahami korelasi, pola, atau tren antar variabel, yang berguna untuk eksplorasi data lebih lanjut dan analisis hubungan.

```
# Frekuensi untuk kolom kategoris
categorical_cols = ['season', 'yr', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', 'weathersit']
for col in categorical_cols:
    plt.figure(figsize=(8, 4))
    sns.countplot(data=df, x=col, palette='viridis')
    plt.title(f"Distribusi {col}")
    plt.xticks(rotation=45)
    plt.show()
```

✓ 0.9s

Python

Output:



Analisis:

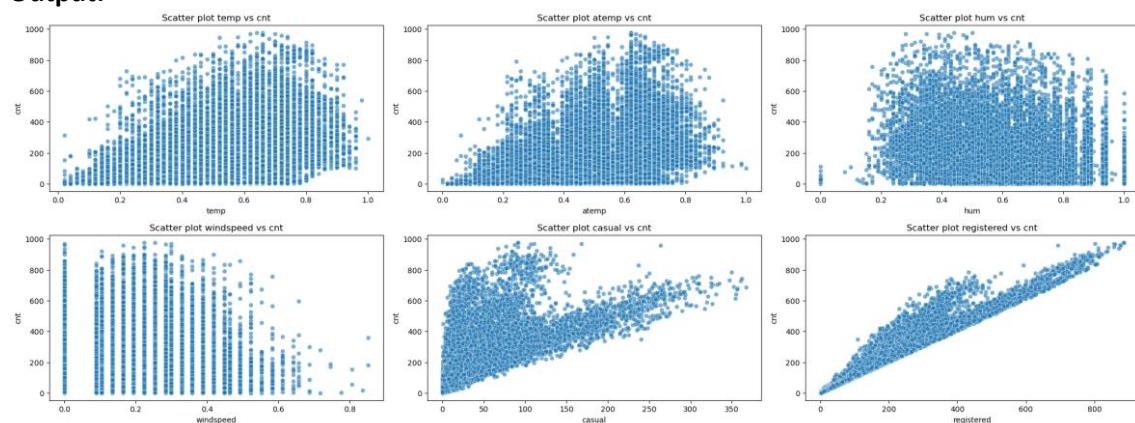
Kode tersebut membuat visualisasi frekuensi untuk kolom-kolom kategoris seperti season, yr, mnth, hr, dan lainnya menggunakan sns.countplot(). Setiap grafik menunjukkan jumlah observasi untuk setiap kategori dalam kolom tertentu. Palet warna viridis digunakan untuk memberikan tampilan visual yang menarik. Teks pada sumbu X diputar 45 derajat dengan plt.xticks(rotation=45) agar lebih mudah dibaca. Visualisasi ini membantu memahami distribusi data dalam kolom kategoris dan mengidentifikasi pola atau ketidakseimbangan kategori.

```
# Hubungan cnt dengan fitur lain
for col in numerical_cols[:-1]: # Exclude 'cnt'
    plt.figure(figsize=(8, 4))
    sns.scatterplot(data=df, x=col, y='cnt', alpha=0.6)
    plt.title(f"scatter plot {col} vs cnt")
    plt.show()
```

✓ 0.8s

Python

Output:

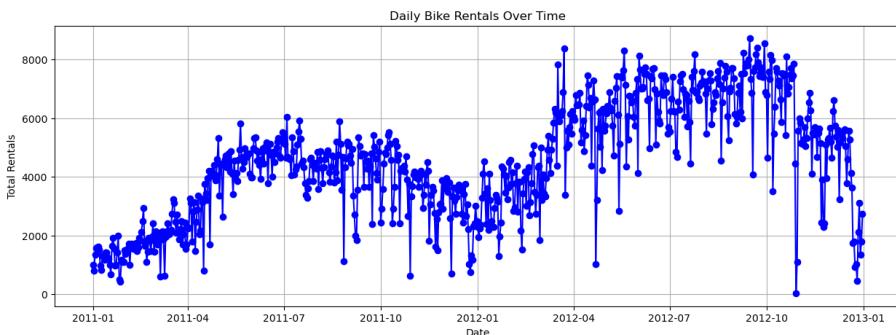


Analisis:

Kode ini memvisualisasikan hubungan antara jumlah total sepeda yang digunakan ('cnt') dengan fitur numerik lainnya, seperti 'temp', 'atemp', 'hum', dan 'windspeed', menggunakan *scatter plot* ('sns.scatterplot()'). Plot ini menunjukkan bagaimana variabel independen memengaruhi 'cnt'. Parameter `alpha=0.6` digunakan untuk membuat titik transparan, sehingga pola menjadi lebih jelas meskipun ada tumpang tindih data. Tujuannya adalah untuk mengidentifikasi korelasi atau tren antara 'cnt' dan variabel numerik lainnya.

```
# Time-series analysis: daily counts
daily_data = df.groupby('dteday').agg({'cnt': 'sum'}).reset_index()
daily_data['dteday'] = pd.to_datetime(daily_data['dteday'])
plt.figure(figsize=(15, 5))
plt.plot(daily_data['dteday'], daily_data['cnt'], marker='o', linestyle='-', color='blue')
plt.title("Daily Bike Rentals Over Time")
plt.xlabel("Date")
plt.ylabel("Total Rentals")
plt.grid()
plt.show()
```

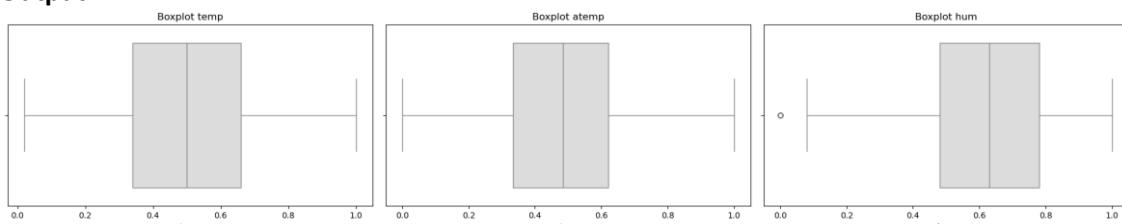
Python

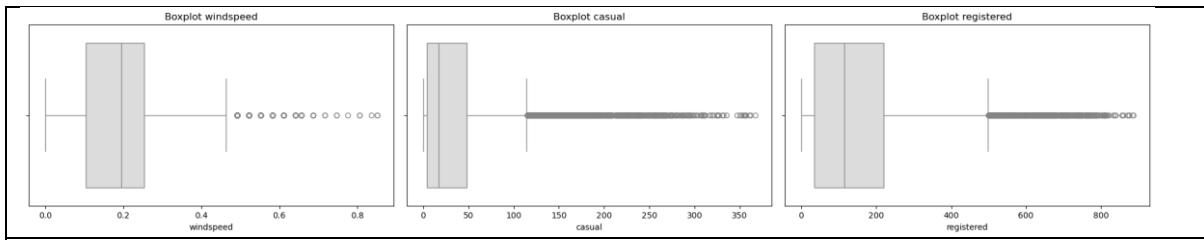
Output:**Analisis:**

Kode ini melakukan analisis deret waktu (time-series analysis) untuk jumlah penyewaan sepeda harian ('cnt'). Data dikelompokkan berdasarkan tanggal ('dteday') menggunakan `groupby()` dan dihitung total penyewaan harian dengan fungsi agregasi `sum()`. Tanggal dikonversi ke format datetime agar sesuai untuk analisis deret waktu. Grafik garis dibuat menggunakan `plt.plot()`, dengan sumbu X menampilkan tanggal dan sumbu Y menampilkan total penyewaan harian. Titik-titik pada garis ditandai dengan `marker='o'` untuk meningkatkan visibilitas data. Grafik ini membantu mengidentifikasi pola tren, musiman, atau anomali dalam penggunaan sepeda harian.

```
# Boxplot untuk mendekati outlier
for col in numerical_cols:
    plt.figure(figsize=(8, 4))
    sns.boxplot(data=df, x=col, palette='coolwarm')
    plt.title(f"Boxplot {col}")
    plt.show()
```

Python

Output:



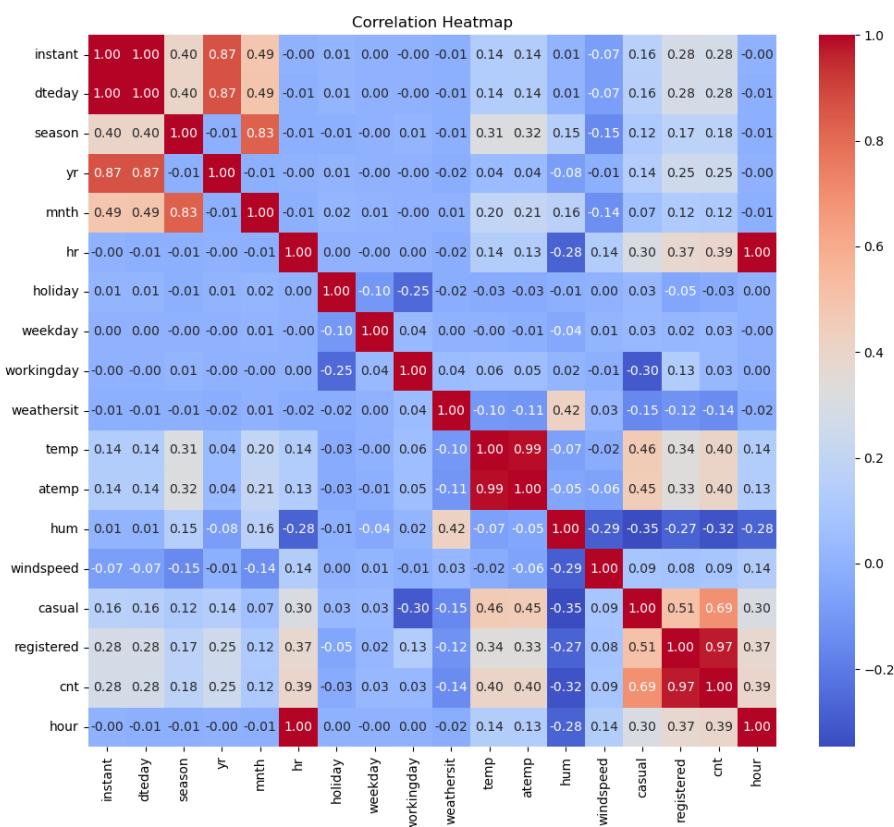
Analisis:

Kode ini menghasilkan boxplot untuk setiap kolom numerik dalam dataset menggunakan `sns.boxplot()`. Boxplot digunakan untuk mendeteksi outlier atau penciran dalam data. Boxplot menunjukkan distribusi data melalui kuartil, dengan garis tengah (median), kotak yang mewakili rentang interkuartil, dan garis keluar (whiskers) yang menunjukkan rentang data yang dianggap normal. Titik-titik di luar whiskers dianggap sebagai outlier. Setiap boxplot membantu untuk memvisualisasikan sebaran data dan mengidentifikasi apakah ada nilai yang tidak biasa dalam setiap fitur numerik.

```
# Correlation heatmap
correlation_matrix = df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
plt.title("Correlation Heatmap")
plt.show()
```

Python
0.4s

Output:



Analisis:

Kode ini menghasilkan heatmap korelasi antara kolom numerik dalam dataset menggunakan `sns.heatmap()`. Matriks korelasi dihitung dengan `df.corr()`, yang menghasilkan nilai korelasi antara setiap pasangan fitur numerik. Heatmap ini membantu untuk memahami hubungan antara fitur-fitur dalam dataset dan mengidentifikasi fitur yang saling berkorelasi.

```
# Correlation analysis
print("\nCorrelation Matrix:\n", correlation_matrix)
✓ 0.0s
```

Python

Output:

```
Correlation Matrix:
instant      dteday    season      yr      mnth      hr  \
instant    1.000000  0.999995  0.404046  0.866014  0.489164 -0.004775
dteday     0.999995  1.000000  0.404452  0.865648  0.489808 -0.006161
season     0.404046  0.404452  1.000000 -0.010742  0.830386 -0.006117
yr        0.866014  0.865648 -0.010742  1.000000 -0.010473 -0.003867
mnth      0.489164  0.489808  0.830386 -0.010473  1.000000 -0.005772
hr        -0.004775 -0.006161 -0.006117 -0.003867 -0.005772  1.000000
holiday    0.014723  0.014737 -0.009585  0.006692  0.018430  0.000479
weekday    0.001357  0.001372 -0.002335 -0.004485  0.010400 -0.003498
workingday -0.003416 -0.003366  0.013743 -0.002196 -0.003477  0.002285
weathersit -0.014198 -0.014063 -0.014524 -0.019157  0.005400 -0.020203
temp       0.136178  0.136074  0.312025  0.040913  0.201691  0.137603
atemp      0.137615  0.137543  0.319380  0.039222  0.208096  0.133750
hum        0.009577  0.010196  0.150625 -0.083546  0.164411 -0.276498
windspeed   -0.074505 -0.074645 -0.149773 -0.008740 -0.135386  0.137252
casual      0.158295  0.157821  0.120206  0.142779  0.068457  0.301202
registered  0.282046  0.281450  0.174226  0.253684  0.122273  0.374141
cnt         0.278379  0.277753  0.178056  0.250495  0.120638  0.394071
hour        -0.004775 -0.006161 -0.006117 -0.003867 -0.005772  1.000000

          holiday  weekday  workingday  weathersit  temp  atemp  \
instant  0.014723  0.001357 -0.003416 -0.014198  0.136178  0.137615
dteday   0.014737  0.001372 -0.003366 -0.014063  0.136074  0.137543
...
casual   -0.347028  0.090287  1.000000  0.506618  0.694564  0.301202
registered -0.273933  0.082321  0.506618  1.000000  0.972151  0.374141
cnt      -0.322911  0.093234  0.694564  0.972151  1.000000  0.394071
hour     -0.276498  0.137252  0.301202  0.374141  0.394071  1.000000

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Analisis:

Kode ini menampilkan matriks korelasi dalam bentuk teks menggunakan `print()` untuk menunjukkan nilai korelasi antara setiap pasangan kolom numerik dalam dataset. Matriks korelasi memberikan informasi tentang sejauh mana dua variabel saling berhubungan, dengan nilai berkisar antara -1 hingga 1. Nilai yang mendekati 1 menunjukkan korelasi positif yang kuat, sementara nilai mendekati -1 menunjukkan korelasi negatif yang kuat. Nilai mendekati 0 menandakan tidak ada korelasi linier yang signifikan antara variabel-variabel tersebut.

```
# Drop unnecessary columns and define features/target
x = df.drop(columns=['instant', 'dteday', 'casual', 'registered', 'cnt']).values
y = df['cnt'].values

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Confirm shapes of the processed data
print(f"Training data shape: {X_train_scaled.shape}, Test data shape: {X_test_scaled.shape}")

✓ 0.0s
```

Python

```
# Define the MLP Regression model
class MLPRegression(nn.Module):
    def __init__(self, input_size, hidden_layers, neurons, activation):
        super(MLPRegression, self).__init__()
        layers = []
        for _ in range(hidden_layers):
            layers.append(nn.Linear(input_size, neurons))
            if activation == 'relu':
                layers.append(nn.ReLU())
            elif activation == 'sigmoid':
                layers.append(nn.Sigmoid())
            elif activation == 'tanh':
                layers.append(nn.Tanh())
            input_size = neurons
        layers.append(nn.Linear(input_size, 1)) # Output layer
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)
```

Python

Analisis:

Pada code diatas pada kolom instant,dteday,casual,registered,cnt dihapus dari fitur X. Sedangkan

kolom cnt digunakan untuk target y yang merupakan variabel yg ingin di prediksi. Kemudian melakukan pembagian data dengan rasio 80:20. lalu standardscaler untuk memastikan bahwa fitur memiliki skala yang sama. Setelah menentukan target, membuat kelas model mlp regression menggunakan pytorch.

```

# hidden layer
hidden_layers = [1]
neurons = [4, 8, 16, 32, 64]
activations = ['linear', 'tanh', 'relu']
losses = ['MSELoss', 'MAELoss', 'BCELoss']
epochs_list = [1, 10, 25, 50, 100, 250]
learning_rates = [0.1, 0.01, 0.001, 0.0001]
batch_sizes = [16, 32, 64, 128, 256, 512]

# results
results_MLP = []

# Device setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Loop through hyperparameter combinations for hidden layer = 1
for layers in hidden_layers:
    for neurons in neurons:
        for activation in activations:
            for epochs in epochs_list:
                for lr in learning_rates:
                    for batch_size in batch_sizes:
                        # Initialize model
                        model = MLPRegression(input_sizeX_train_scaled.shape[1],
                                              hidden_layers=layers,
                                              neurons=neurons,
                                              activation=activation).to(device)

                        # Define loss function and optimizer
                        criterion = nn.MSELoss()
                        optimizer = optim.Adam(model.parameters(), lr=lr)

                        # Convert data to tensors
                        X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32).to(device)
                        Y_train_tensor = torch.tensor(y_train, dtype=torch.float32).to(device)
                        X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32).to(device)
                        y_test_tensor = torch.tensor(y_test, dtype=torch.float32).to(device)

                        # Training loop
                        for epoch in range(epochs):
                            model.train()
                            optimizer.zero_grad()
                            outputs = model(X_train_tensor).squeeze()
                            loss = criterion(outputs, y_train_tensor)
                            loss.backward()
                            optimizer.step()

                            # Evaluate validation set
                            model.eval()
                            with torch.no_grad():
                                y_pred = model(X_test_tensor).squeeze().cpu().numpy()
                                rmse = mean_absolute_error(y_test, y_pred)
                                r2 = r2_score(y_test, y_pred)
                                accuracy = (1 - rmse / max(y_test)) # estimated accuracy

                        # Final evaluation after all epochs
                        model.eval()
                        with torch.no_grad():
                            y_pred = model(X_test_tensor).squeeze().cpu().numpy()
                            rmse = mean_absolute_error(y_test, y_pred)
                            r2 = r2_score(y_test, y_pred)
                            accuracy = (1 - rmse / max(y_test))

                        # Store results
                        results_MLP.append({
                            'layers': layers,
                            'neurons': neurons,
                            'activation': activation,
                            'epoch': epoch,
                            'lr': lr,
                            'batch_size': batch_size,
                            'rmse': rmse,
                            'r2': r2,
                            'accuracy': accuracy
                        })

# Convert results to DataFrame
results_MLP = pd.DataFrame(results_MLP)
results_MLP.to_csv("MLP_Regression_Hidden_Layer_1.csv", index=False)
print("CSV file saved")

```

Output:

```

Hidden layer: 1, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 16
MSE: 65921.8463, MAE: 185.3165, R^2: -1.0818, Accuracy: 0.8103
=====
Hidden layer: 1, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 32
MSE: 65628.5247, MAE: 184.5419, R^2: -1.0726, Accuracy: 0.8111
=====
Hidden layer: 1, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 64
MSE: 65587.2898, MAE: 184.3143, R^2: -1.0713, Accuracy: 0.8113
=====
Hidden layer: 1, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 128
MSE: 65576.0398, MAE: 184.7854, R^2: -1.0709, Accuracy: 0.8109
=====
Hidden layer: 1, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 256
MSE: 65869.1981, MAE: 185.1583, R^2: -1.0802, Accuracy: 0.8105
=====
Hidden layer: 1, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 512
MSE: 65805.1963, MAE: 184.8303, R^2: -1.0781, Accuracy: 0.8108
=====
Hidden layer: 1, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.01, Batch Size: 16
MSE: 65784.3716, MAE: 184.5566, R^2: -1.0775, Accuracy: 0.8111
=====
Hidden layer: 1, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.01, Batch Size: 32
MSE: 66070.5240, MAE: 185.4334, R^2: -1.0865, Accuracy: 0.8102
=====
Hidden layer: 1, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.01, Batch Size: 64
...
=====
Hidden layer: 1, Neurons: 64, Activation: tanh, Epochs: 250, LR: 0.0001, Batch Size: 512
MSE: 65639.1129, MAE: 184.6152, R^2: -1.0729, Accuracy: 0.8110
=====

Output was truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```

Analisis:

Kode ini berfungsi untuk mencari kombinasi pengaturan terbaik untuk model MLP Regression dengan satu lapisan tersembunyi. Beberapa hal yang diuji adalah jumlah neuron di lapisan tersembunyi, fungsi aktivasi (seperti ReLU atau Sigmoid), jumlah epoch (berapa lama model dilatih), laju pembelajaran (berapa cepat model belajar), dan ukuran batch (berapa banyak data yang diproses sekaligus). Setiap kombinasi pengaturan diuji satu per satu dengan melatih model, lalu mengukur seberapa baik model tersebut dengan metrik seperti MSE (kesalahan kuadrat rata-rata), MAE (kesalahan rata-rata mutlak), R² (koefisien determinasi), dan akurasi. Setelah model selesai dilatih, hasil-hasil evaluasi disimpan dalam sebuah tabel, yang kemudian disimpan dalam

bentuk file CSV. Tujuan dari proses ini adalah untuk menemukan pengaturan terbaik bagi model dan memahami seberapa baik model bekerja dengan berbagai kombinasi parameter.

```

import pandas as pd
import matplotlib.pyplot as plt
import os

# Load the results dataset
results_dfl_3 = pd.read_csv("celp_regression_hidden_layer_3.csv") # Sesuaikan nama file jika berbeda

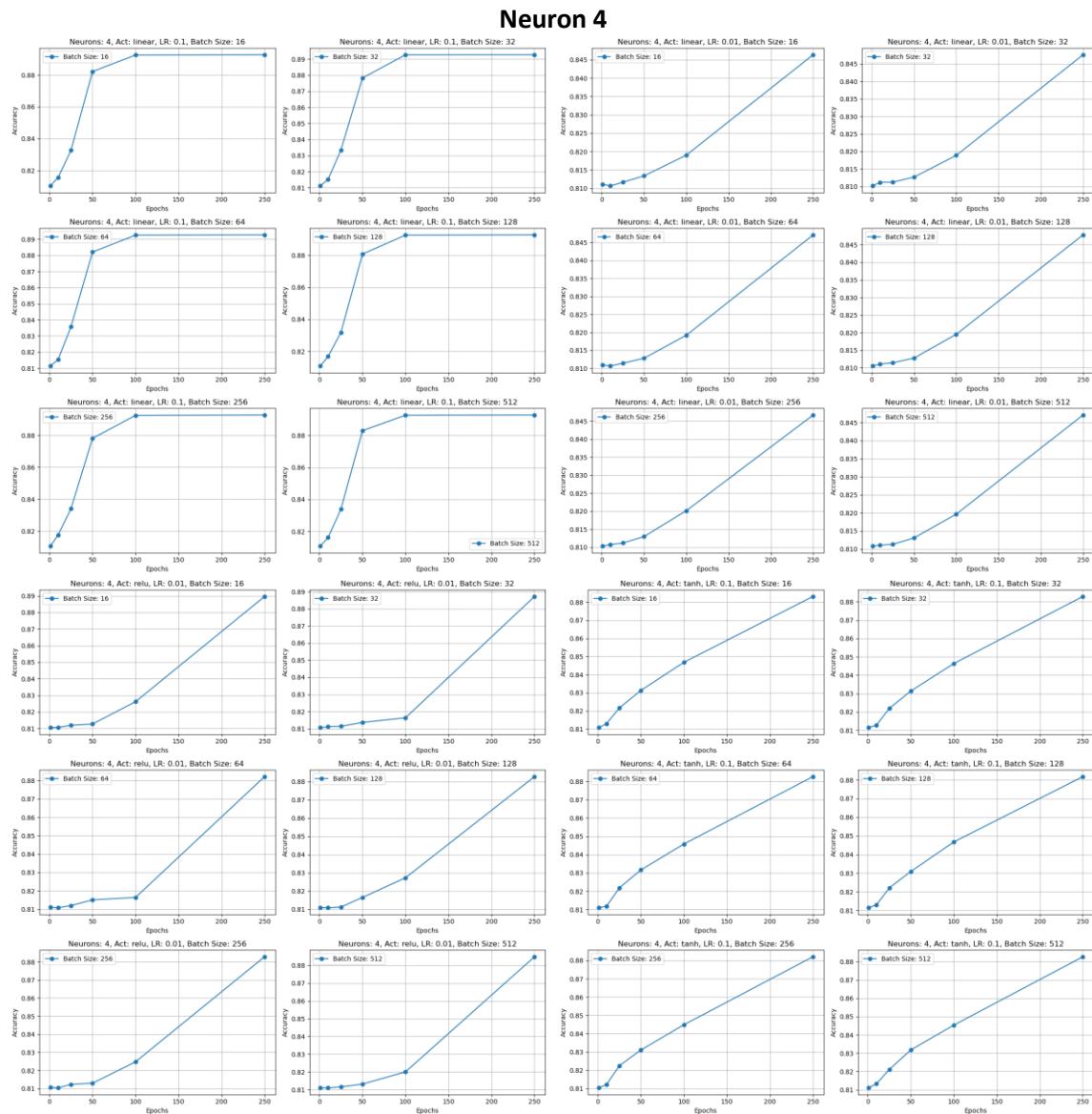
# Load the column names, activation, and learning rate
neuron_list = results_dfl_3['neuron'].unique()
activation_list = results_dfl_3['activation'].unique()
learning_rate = results_dfl_3['lr'].unique()

# Get unique activation rates
activation_rates = results_dfl_3['lr'].unique()

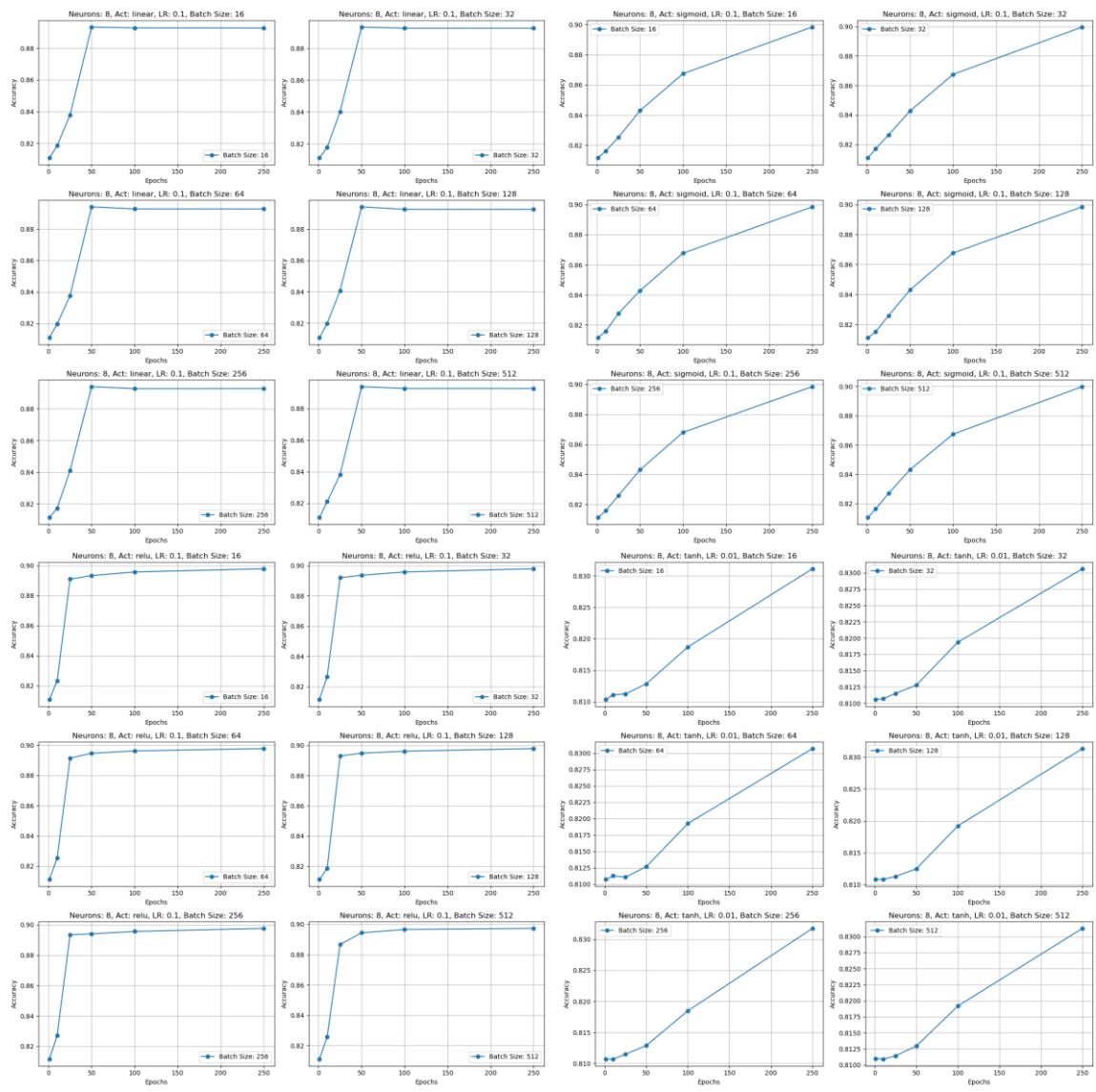
for neuron in neuron_list:
    for activation in activation_list:
        for lr in learning_rate:
            if activation == "tanh" or lr == "0.01":
                continue
            else:
                filtered_data = results_dfl_3[(results_dfl_3['neuron'] == neuron) & (results_dfl_3['activation'] == activation) & (results_dfl_3['lr'] == lr)]
                if not filtered_data.empty:
                    # Plot accuracy per epoch for each batch size
                    filtered_data['batch_size'] = filtered_data['batch_size'].unique()
                    plt.figure(figsize=(12, 12))
                    for b_size in filtered_data['batch_size']:
                        subplot(2, 2, b_size)
                        data = filtered_data[filtered_data['batch_size'] == b_size]
                        plot(data['epoch'], data['accuracy'], 'o-', label=f'Batch Size: {b_size}')
                    plt.title(f'Neurons: {neuron}, Act: {activation}, LR: {lr}, Batch sizes: {batch_size}')
                    plt.xlabel('Epochs')
                    plt.ylabel('Accuracy')
                    plt.legend()
                    plt.tight_layout()
                    plt.show()

```

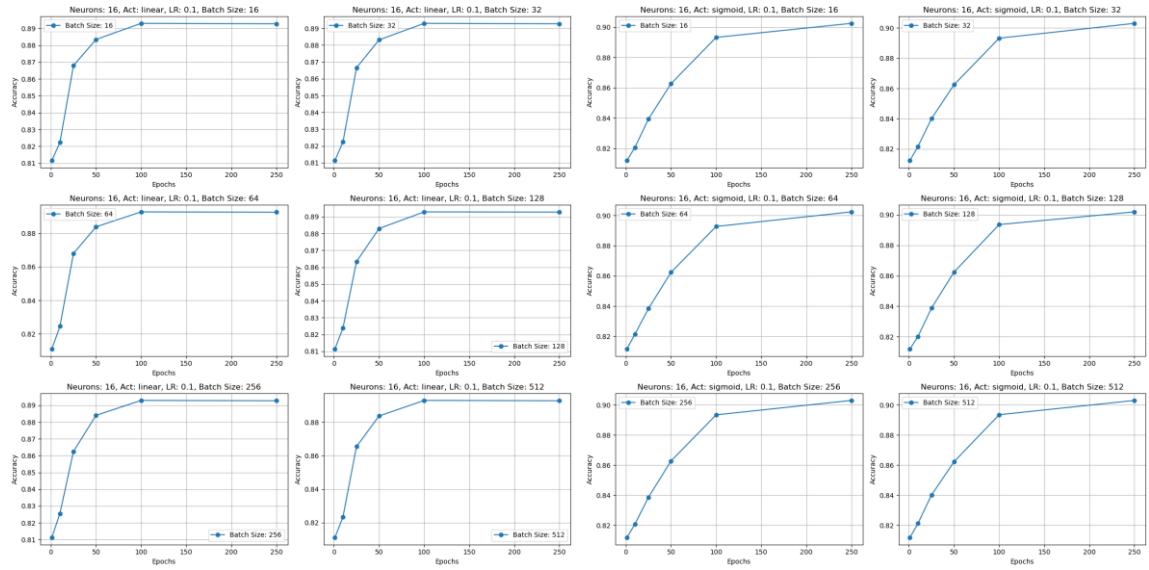
Output:

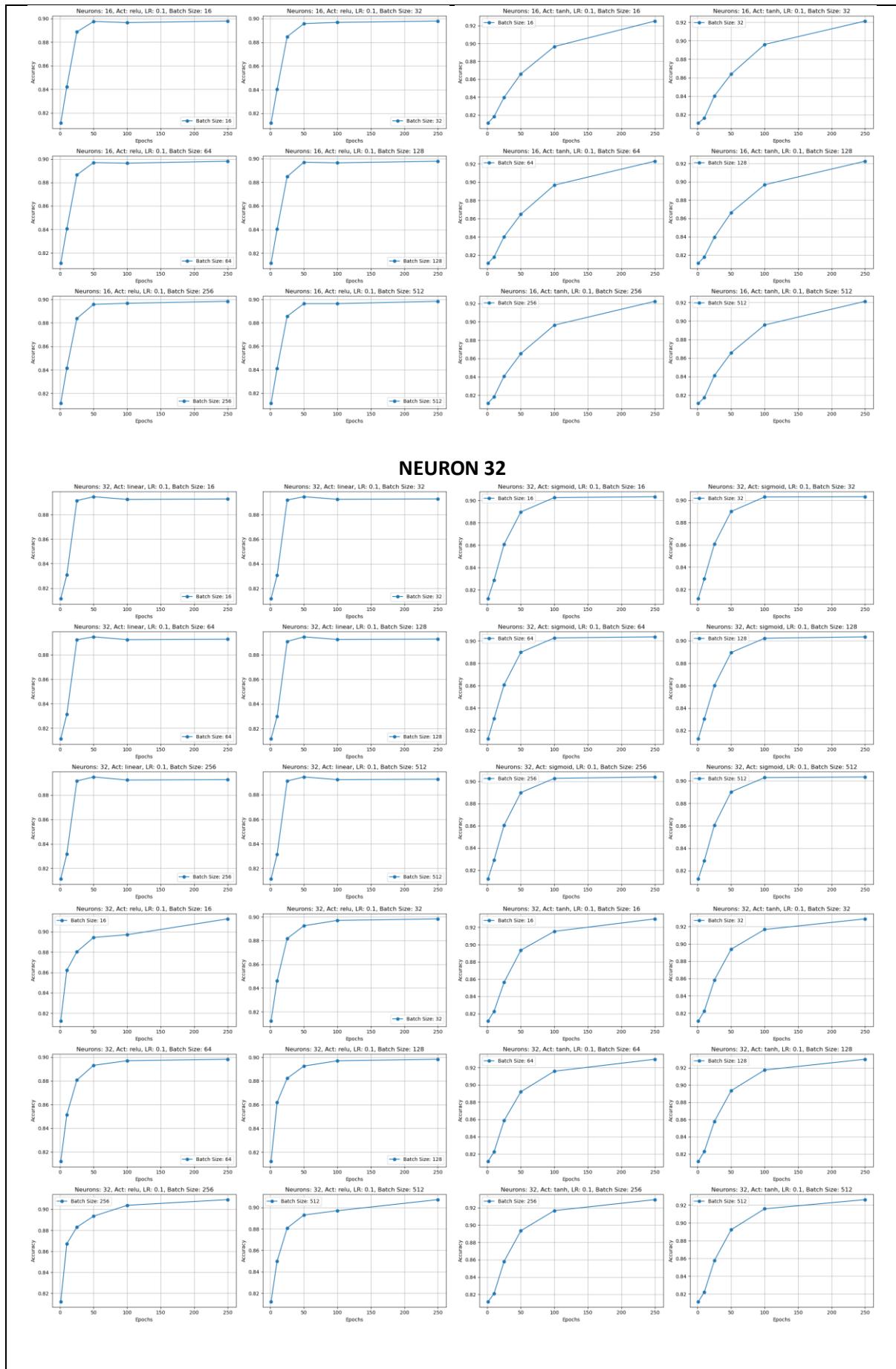


NEURON 8

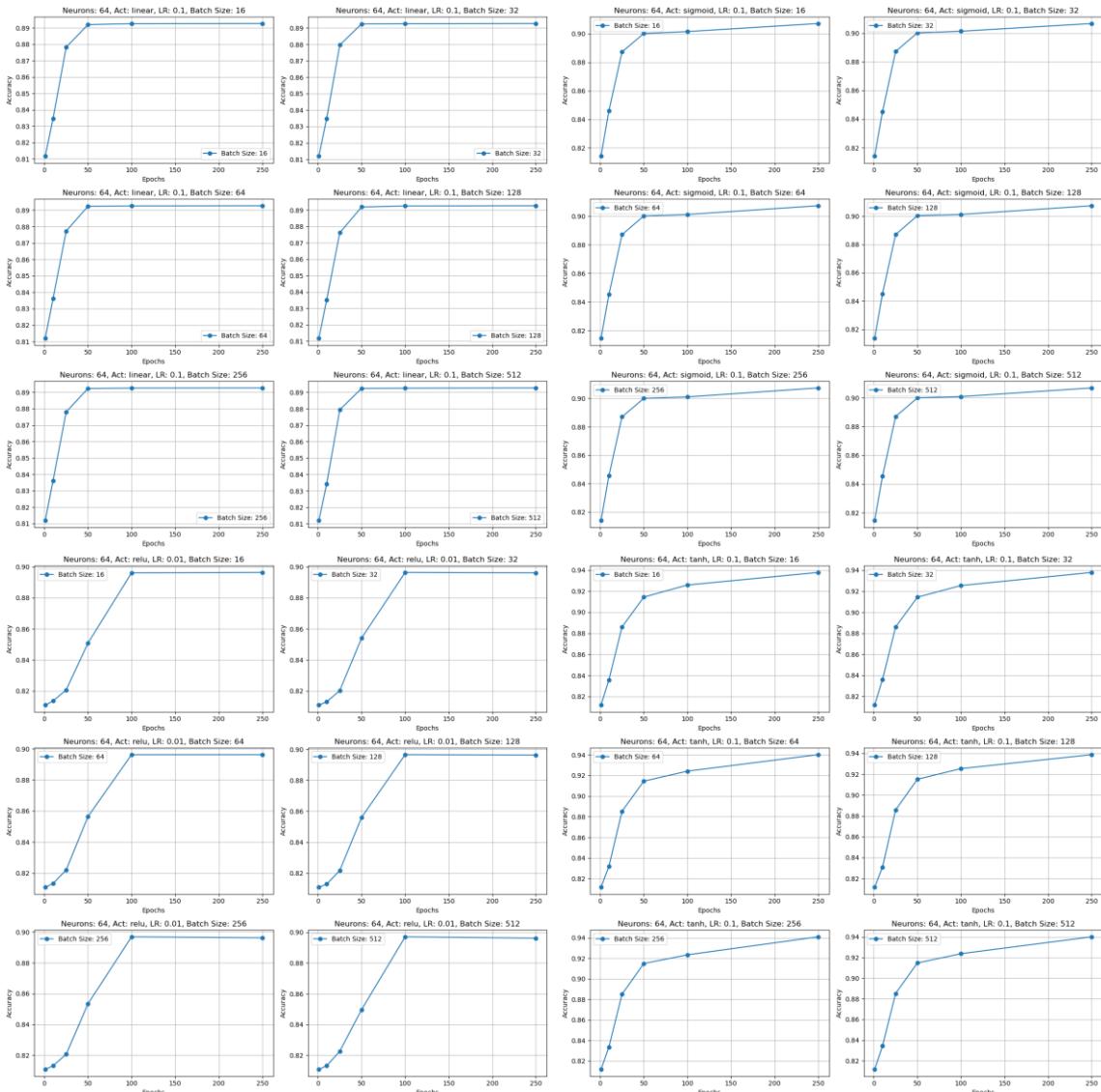


NEURON 16





NEURON 64



Analisis:

beberapa grafik akurasi terlihat sama meskipun ada perubahan pada kombinasi parameter model dapat terjadi karena beberapa alasan. Pertama, perubahan ukuran batch mungkin tidak terlalu mempengaruhi akurasi jika model sudah cukup stabil dalam proses pembelajaran atau dataset tidak memiliki kompleksitas yang tinggi. Kedua, jika model terlalu sederhan (misalnya dengan jumlah neuron yang terbatas atau lapisan tersembunyi yang sedikit) maka model tersebut tidak memiliki kapasitas yang cukup untuk menangkap pola-pola kompleks dalam data. Dalam kasus ini, meskipun ada variasi pada parameter seperti ukuran batch atau fungsi aktivasi, model tetap tidak dapat meningkatkan akurasinya secara signifikan. Ketiga yaitu pemilihan learning rate yang kurang optimal dalam data juga bisa menyebabkan performa model tetap stagnan meskipun ada perubahan parameter.

```

# hidden layer = 1
hidden_layers = [1]
neurons = [4, 8, 16, 32, 64]
activations = ['linear', 'sigmoid', 'tanh', 'tanh']
epochs_list = [1, 10, 25, 50, 100, 200]
learning_rates = [0.1, 0.01, 0.001, 0.0001]
batch_sizes = [16, 32, 64, 128, 256, 512]

# store results
results_MLP = []

# loop through device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# loop through hyperparameter combinations for hidden layer = 2
for layers in hidden_layers:
    for neuron in neurons:
        for activation in activations:
            for epochs in epochs_list:
                for lr in learning_rates:
                    for batch_size in batch_sizes:
                        # initialize model
                        model = MLPregression([input_layer_train, train_scaled.shape[1]], hidden_layers=2, neurons=neuron, activation=activation).to(device)

                        # define loss function and optimizer
                        criterion = nn.MSELoss()
                        optimizer = optim.Adam(model.parameters(), lr=lr)

                        # Convert data to tensors
                        X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float).to(device)
                        Y_train_tensor = torch.tensor(Y_train_scaled, dtype=torch.float).to(device)
                        X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float).to(device)
                        Y_test_tensor = torch.tensor(Y_test, dtype=torch.float).to(device)

                        # Training loop
                        for epoch in range(epochs):
                            model.train()
                            optimizer.zero_grad()
                            output = model(X_train_tensor).squeeze()
                            loss = criterion(output, Y_train_tensor)
                            loss.backward()
                            optimizer.step()

                            # Evaluate validation set
                            with torch.no_grad():
                                y_pred = model(X_test_tensor).squeeze().cpu().numpy()
                                mse = np.mean((y_test - y_pred) ** 2)
                                mae = mean_absolute_error(y_test, y_pred)
                                rmse = np.sqrt(mse)
                                accuracy = 1 - rmse / np.std(y_test) # estimate akurasi

                        model.eval()
                        with torch.no_grad():
                            y_pred = model(X_test_tensor).squeeze().cpu().numpy()
                            mse = mean_absolute_error(y_test, y_pred)
                            rmse = np.sqrt(mse)
                            accuracy = 1 - rmse / np.std(y_test)

                        results_MLP.append({
                            'layers': layers,
                            'neuron': neuron,
                            'activation': activation,
                            'epochs': epochs,
                            'lr': lr,
                            'batch_size': batch_size,
                            'mse': mse,
                            'mae': mae,
                            'rmse': rmse,
                            'accuracy': accuracy
                        })

print("Hidden layer: [Layers], Neurons: [neuron], Activation: [activation], Epochs: [epoch], lr: [lr], batch_size: [batch_size]")
print("MSE: [mse], MAE: [mae], RMSE: [rmse], Accuracy: [accuracy]")

# convert results to pandas
results_df_2 = pd.DataFrame(results_MLP)
results_df_2.to_csv("mlp_regression_hidden_layer_2.csv", index=False)

```

Output:

```

Hidden layer: 2, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 16
MSE: 65811.2245, MAE: 184.8452, R^2: -1.0783, Accuracy: 0.8108
=====
Hidden layer: 2, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 32
MSE: 65862.5427, MAE: 185.1453, R^2: -1.0800, Accuracy: 0.8105
=====
Hidden layer: 2, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 64
MSE: 65656.5960, MAE: 184.6566, R^2: -1.0734, Accuracy: 0.8110
=====
Hidden layer: 2, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 128
MSE: 65595.8237, MAE: 184.2617, R^2: -1.0715, Accuracy: 0.8114
=====
Hidden layer: 2, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 256
MSE: 65737.0733, MAE: 184.7894, R^2: -1.0760, Accuracy: 0.8109
=====
Hidden layer: 2, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 512
MSE: 65865.7794, MAE: 184.9218, R^2: -1.0801, Accuracy: 0.8107
=====
Hidden layer: 2, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.01, Batch Size: 16
MSE: 66085.5189, MAE: 185.5687, R^2: -1.0870, Accuracy: 0.8101
=====
Hidden layer: 2, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.01, Batch Size: 32
MSE: 65868.2264, MAE: 184.9213, R^2: -1.0801, Accuracy: 0.8107
=====
Hidden layer: 2, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.01, Batch Size: 64
...
=====
Hidden layer: 2, Neurons: 64, Activation: tanh, Epochs: 250, LR: 0.0001, Batch Size: 512
MSE: 65200.7388, MAE: 183.9476, R^2: -1.0591, Accuracy: 0.8117
=====

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```

Analisis:

Kode ini berfungsi untuk mencari kombinasi pengaturan terbaik untuk model MLP Regression dengan satu lapisan tersembunyi. Beberapa hal yang diuji adalah jumlah neuron di lapisan tersembunyi, fungsi aktivasi (seperti ReLU atau Sigmoid), jumlah epoch (berapa lama model dilatih), laju pembelajaran (berapa cepat model belajar), dan ukuran batch (berapa banyak data yang diproses sekaligus). Setiap kombinasi pengaturan diuji satu per satu dengan melatih model, lalu mengukur seberapa baik model tersebut dengan metrik seperti MSE (kesalahan kuadrat rata-rata), MAE (kesalahan rata-rata mutlak), R² (koefisien determinasi), dan akurasi. Setelah model selesai dilatih, hasil-hasil evaluasi disimpan dalam sebuah tabel, yang kemudian disimpan dalam bentuk file CSV. Tujuan dari proses ini adalah untuk menemukan pengaturan terbaik bagi model dan memahami seberapa baik model bekerja dengan berbagai kombinasi parameter.

```

import pandas as pd
import matplotlib.pyplot as plt
import os

# Load the results dataset
results_df_2 = pd.read_csv("mlp_regression_hidden_layer_2.csv") # Sesuaikan nama file jika berbeda

# Loop melalui semua kombinasi Neurons, Activation, dan Learning Rate
neurons_list = results_df_2['neurons'].unique()
activations_list = results_df_2['activation'].unique()
learning_rates = results_df_2['lr'].unique()

for neuron in neurons_list:
    for activation in activations_list:
        for lr in learning_rates:
            # Filter data untuk kombinasi parameter ini
            filtered_data = results_df_2[
                (results_df_2['neurons'] == neuron) &
                (results_df_2['activation'] == activation) &
                (results_df_2['lr'] == lr)
            ]

            if not filtered_data.empty:
                # Plot akurasi terhadap epochs untuk setiap batch size
                batch_sizes = filtered_data['batch_size'].unique()

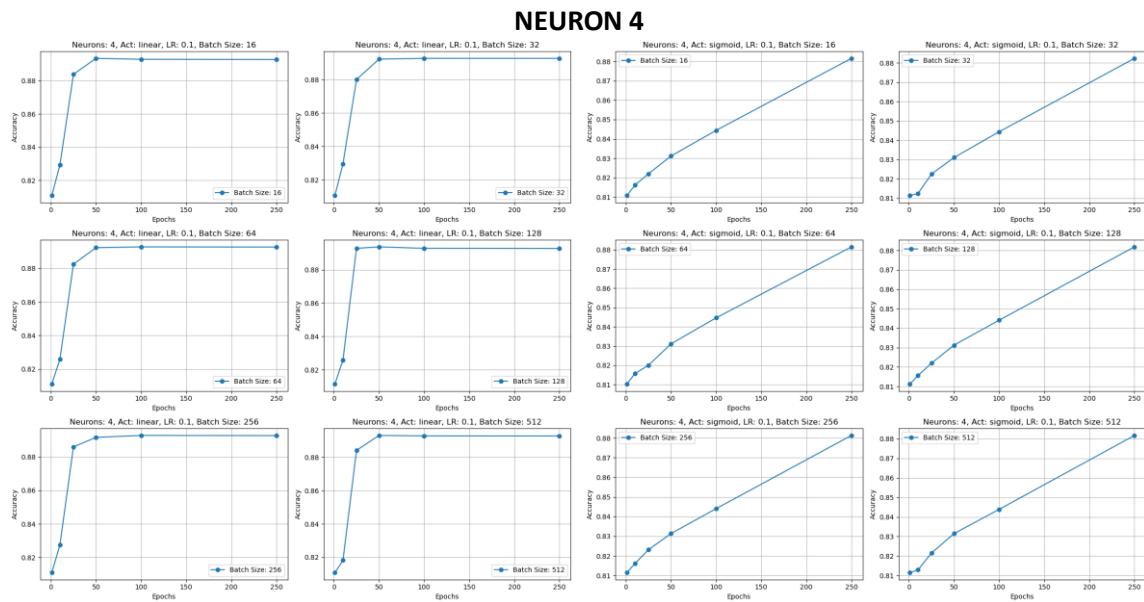
                plt.figure(figsize=(12, 12))
                for i, batch_size in enumerate(batch_sizes, 1):
                    plt.subplot(3, 2, i) # Buat grid 3x2
                    data = filtered_data[filtered_data['batch_size'] == batch_size]
                    plt.plot(data['epochs'], data['accuracy'], marker='o', label=f'Batch Size: {batch_size}')
                    plt.title(f'Neurons: {neuron}, Act: {activation}, LR: {lr}, Batch Size: {batch_size}')
                    plt.xlabel('Epochs')
                    plt.ylabel('Accuracy')
                    plt.legend()
                    plt.grid()

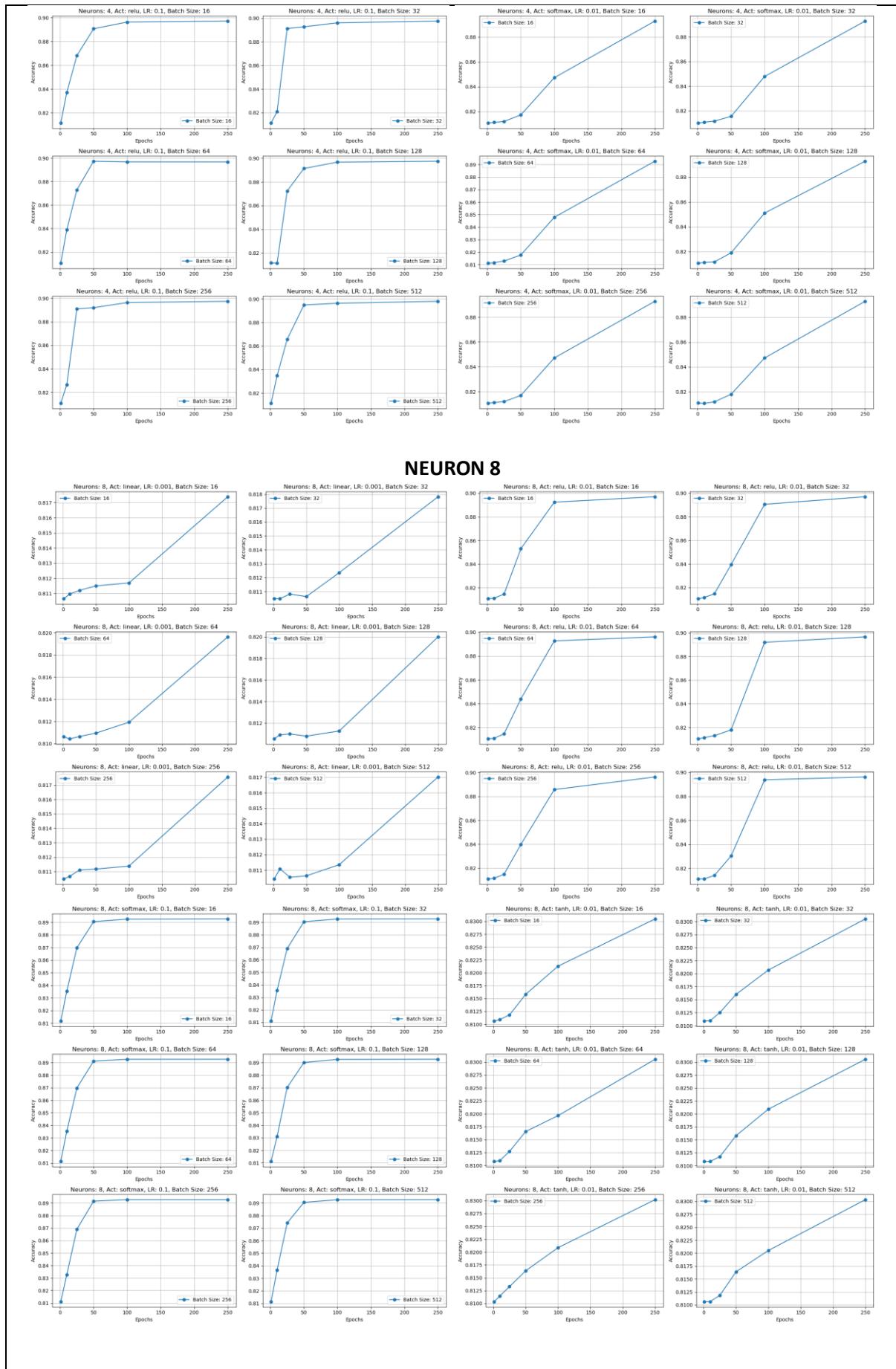
                plt.tight_layout()
                plt.show()

```

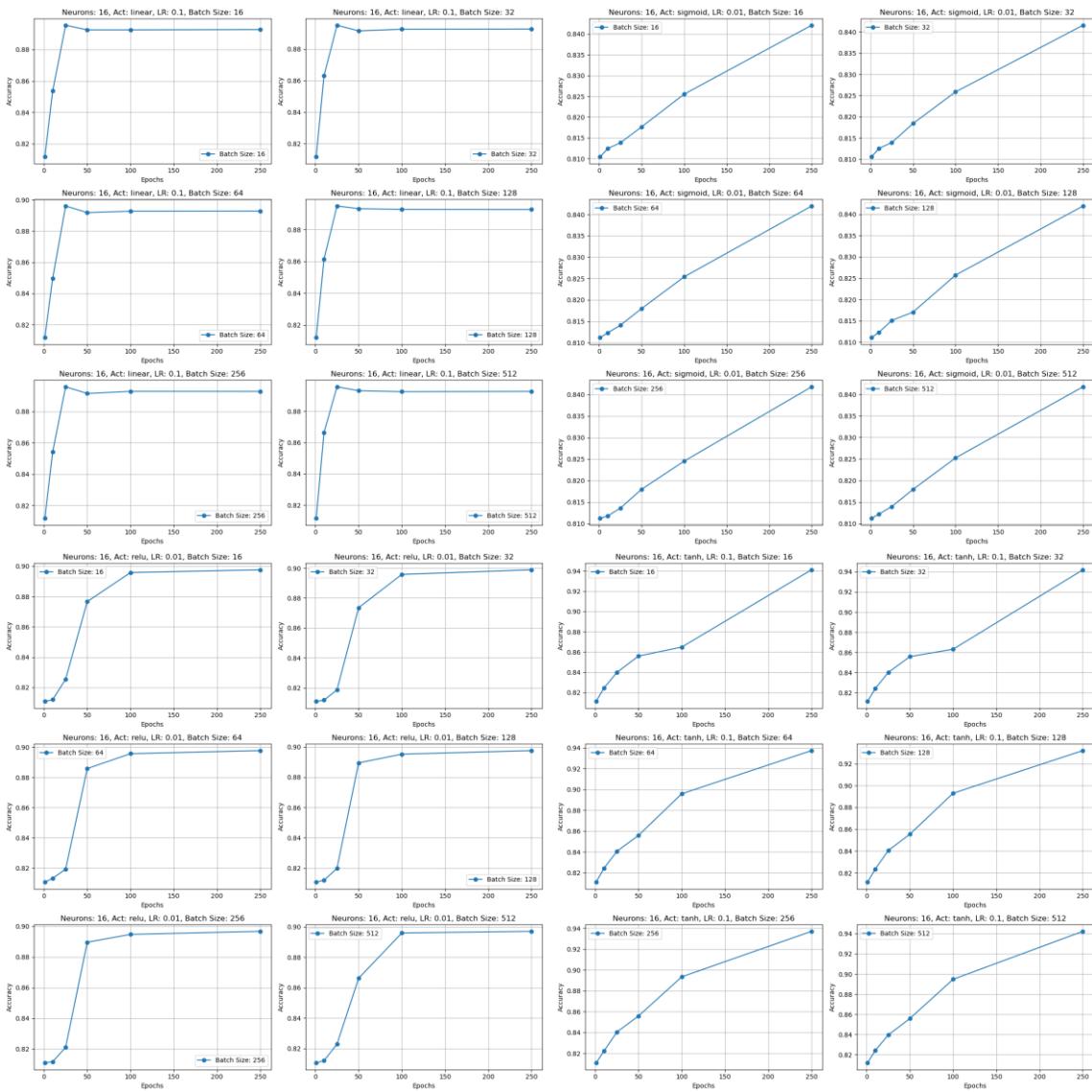
✓ 1m 8.8s Python

Output:

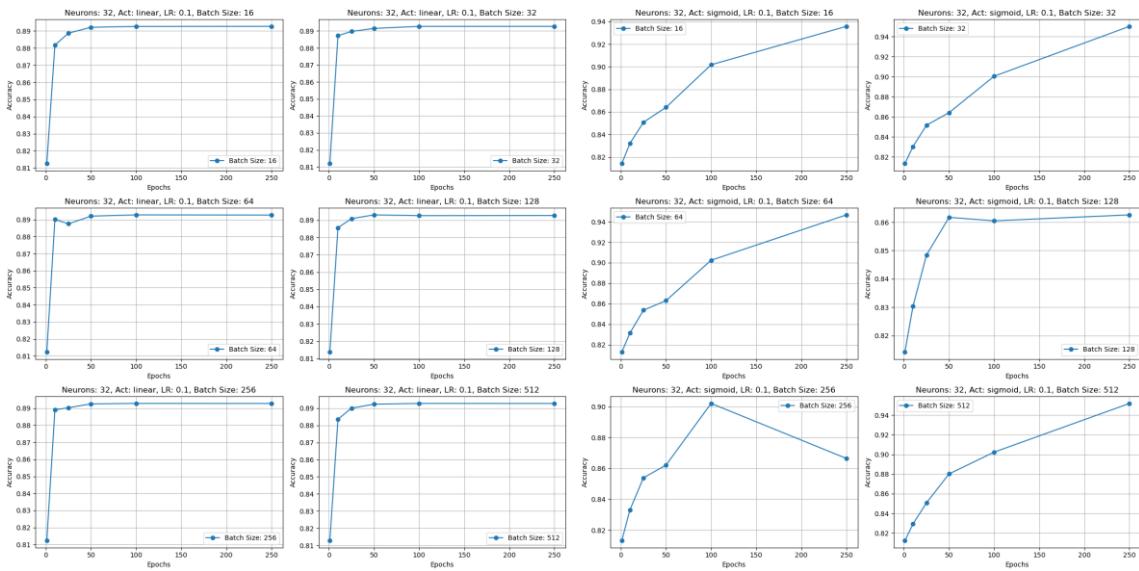


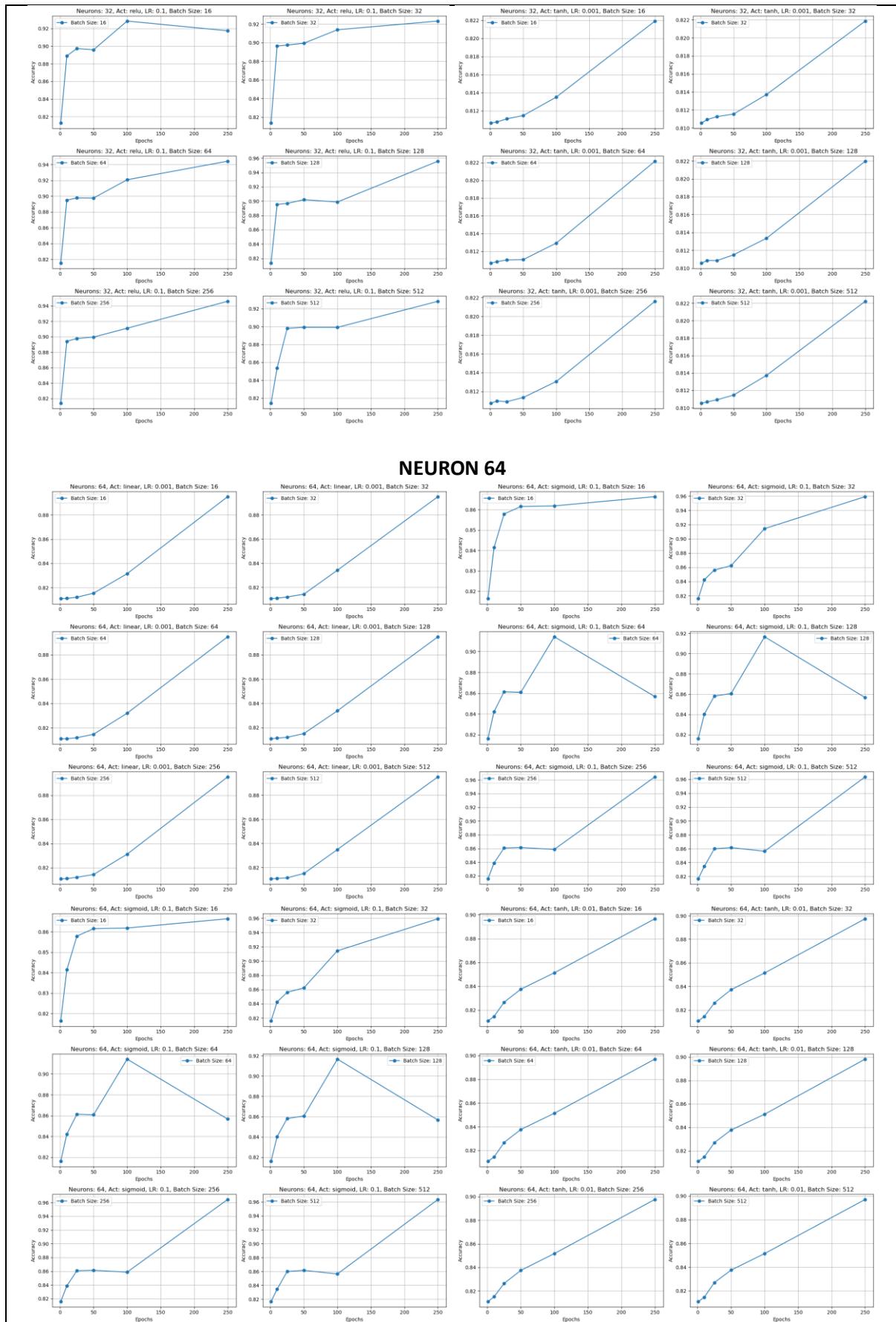


NEURON 16



NEURON 32





Analisis:

Beberapa grafik akurasi terlihat sama meskipun ada perubahan pada kombinasi parameter model

dapat terjadi karena beberapa alasan. Pertama, perubahan ukuran batch mungkin tidak terlalu mempengaruhi akurasi jika model sudah cukup stabil dalam proses pembelajaran atau dataset tidak memiliki kompleksitas yang tinggi. Kedua, jika model terlalu sederhan (misalnya dengan jumlah neuron yang terbatas atau lapisan tersembunyi yang sedikit) maka model tersebut tidak memiliki kapasitas yang cukup untuk menangkap pola-pola kompleks dalam data. Dalam kasus ini, meskipun ada variasi pada parameter seperti ukuran batch atau fungsi aktivasi, model tetap tidak dapat meningkatkan akurasinya secara signifikan. Ketiga yaitu pemilihan learning rate yang kurang optimal dalam data juga bisa menyebabkan performa model tetap stagnan meskipun ada perubahan parameter.

```

# hidden layers = 3
hidden_layers = [1]
neurons = [4, 32, 64]
activations = ['linear', 'sigmoid', 'relu', 'softmax', 'tanh']
epochs_list = [1, 10, 25, 50, 100, 250]
learning_rate_list = [0.0001, 0.001, 0.01]
batch_sizes = [16, 32, 64, 128, 256, 512]

# store results
results_h3 = []

# Device setup
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Loop through hyperparameter combinations for hidden layer = 3
for layers in hidden_layers:
    for neurons in neurons:
        for activation in activations:
            for epochs in epochs_list:
                for lr in learning_rate_list:
                    for batch_size in batch_sizes:
                        print(f"\nHidden layer: {layers}, Neurons: {neurons}, Activation: {activation}, Epochs: {epochs}, LR: {lr}, Batch Size: {batch_size}")

                        # Create model
                        model = nn.Linear(1, neurons)
                        model = nn.Sequential(nn.Linear(1, neurons), nn.ReLU(), nn.Linear(neurons, 1))

                        # Define loss function and optimizer
                        criterion = nn.MSELoss()
                        optimizer = optim.Adam(model.parameters(), lr=lr)

                        # Prepare data in tensors
                        X_train_tensor = torch.tensor(x_train_scaled, dtype=torch.float32).to(device)
                        y_train_tensor = torch.tensor(y_train, dtype=torch.float32).to(device)
                        X_text_tensor = torch.tensor(x_text_scaled, dtype=torch.float32).to(device)
                        y_text_tensor = torch.tensor(y_text, dtype=torch.float32).to(device)
                        X_val_tensor = torch.tensor(x_val_scaled, dtype=torch.float32).to(device)
                        y_val_tensor = torch.tensor(y_val, dtype=torch.float32).to(device)

                        # Training loop
                        for epoch in range(epochs):
                            model.train()
                            model.zero_grad()
                            outputs = model(X_train_tensor).squeeze()
                            loss = criterion(outputs, y_train_tensor)
                            loss.backward()
                            optimizer.step()

                            # Evaluate on validation set
                            model.eval()
                            with torch.no_grad():
                                y_pred = model(X_val_tensor).squeeze().cpu().numpy()
                                rmse = mean_absolute_error(y_val, y_pred)
                                rse = mean_squared_error(y_val, y_pred)
                                r2 = r2_score(y_val, y_pred)
                                accuracy = 1 - rmse / max(y_val)

                        # Final evaluation after all epochs
                        model.eval()
                        with torch.no_grad():
                            y_pred = model(X_text_tensor).squeeze().cpu().numpy()
                            rmse = mean_absolute_error(y_text, y_pred)
                            rse = mean_squared_error(y_text, y_pred)
                            r2 = r2_score(y_text, y_pred)
                            accuracy = 1 - rmse / max(y_text)

                        # Store results
                        results_h3.append([
                            layers,
                            neurons,
                            activation,
                            epochs,
                            lr,
                            batch_size,
                            rmse,
                            rse,
                            r2,
                            accuracy
                        ])

                        print(f"Hidden layer: {layers}, Neurons: {neurons}, Activation: {activation}, Epochs: {epochs}, LR: {lr}, Batch Size: {batch_size}")
                        print(f"RMSE: {rmse}, MAE: {mae}, R^2: {r2}, Accuracy: {accuracy}\n")
print("Done!")

```

Output:

```

Hidden layer: 3, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 16
MSE: 65497.9510, MAE: 184.1430, R^2: -1.0684, Accuracy: 0.8115
=====
Hidden layer: 3, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 32
MSE: 65505.1865, MAE: 184.0739, R^2: -1.0687, Accuracy: 0.8116
=====
Hidden layer: 3, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 64
MSE: 65910.9464, MAE: 185.0587, R^2: -1.0815, Accuracy: 0.8106
=====
Hidden layer: 3, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 128
MSE: 65593.3002, MAE: 184.0714, R^2: -1.0714, Accuracy: 0.8115
=====
Hidden layer: 3, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 256
MSE: 65679.1938, MAE: 184.5318, R^2: -1.0742, Accuracy: 0.8111
=====
Hidden layer: 3, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.1, Batch Size: 512
MSE: 65920.3518, MAE: 185.2019, R^2: -1.0818, Accuracy: 0.8104
=====
Hidden layer: 3, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.01, Batch Size: 16
MSE: 65984.5526, MAE: 185.3555, R^2: -1.0838, Accuracy: 0.8103
=====
Hidden layer: 3, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.01, Batch Size: 32
MSE: 65709.2517, MAE: 184.5386, R^2: -1.0751, Accuracy: 0.8111
=====
Hidden layer: 3, Neurons: 4, Activation: linear, Epochs: 1, LR: 0.01, Batch Size: 64
...
=====
Hidden layer: 3, Neurons: 64, Activation: tanh, Epochs: 250, LR: 0.0001, Batch Size: 512
MSE: 64330.6672, MAE: 181.8854, R^2: -1.0316, Accuracy: 0.8138
=====

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```

Analisis:

Kode ini berfungsi untuk mencari kombinasi pengaturan terbaik untuk model MLP Regression dengan satu lapisan tersembunyi. Beberapa hal yang diuji adalah jumlah neuron di lapisan tersembunyi, fungsi aktivasi (seperti ReLU atau Sigmoid), jumlah epoch (berapa lama model dilatih), laju pembelajaran (berapa cepat model belajar), dan ukuran batch (berapa banyak data yang diproses sekaligus). Setiap kombinasi pengaturan diuji satu per satu dengan melatih model, lalu mengukur seberapa baik model tersebut dengan metrik seperti MSE (kesalahan kuadrat rata-rata), MAE (kesalahan rata-rata mutlak), R² (koefisien determinasi), dan akurasi. Setelah model selesai dilatih, hasil-hasil evaluasi disimpan dalam sebuah tabel, yang kemudian disimpan dalam bentuk file CSV. Tujuan dari proses ini adalah untuk menemukan pengaturan terbaik bagi model dan memahami seberapa baik model bekerja dengan berbagai kombinasi parameter.

```
import pandas as pd
import matplotlib.pyplot as plt
import os

# Load the results dataset
results_df_3 = pd.read_csv("mlp_regression_hidden_layer_3.csv") # Sesuaikan nama file jika berbeda

# Loop melalui semua kombinasi Neurons, Activation, dan Learning Rate
neurons_list = results_df_3['neurons'].unique()
activations_list = results_df_3['activation'].unique()
learning_rates = results_df_3['lr'].unique()

for neuron in neurons_list:
    for activation in activations_list:
        for lr in learning_rates:
            # Filter data untuk kombinasi parameter ini
            filtered_data = results_df_3[
                (results_df_3['neurons'] == neuron) &
                (results_df_3['activation'] == activation) &
                (results_df_3['lr'] == lr)
            ]

            if not filtered_data.empty:
                # Plot akurasi terhadap epochs untuk setiap batch size
                batch_sizes = filtered_data['batch_size'].unique()

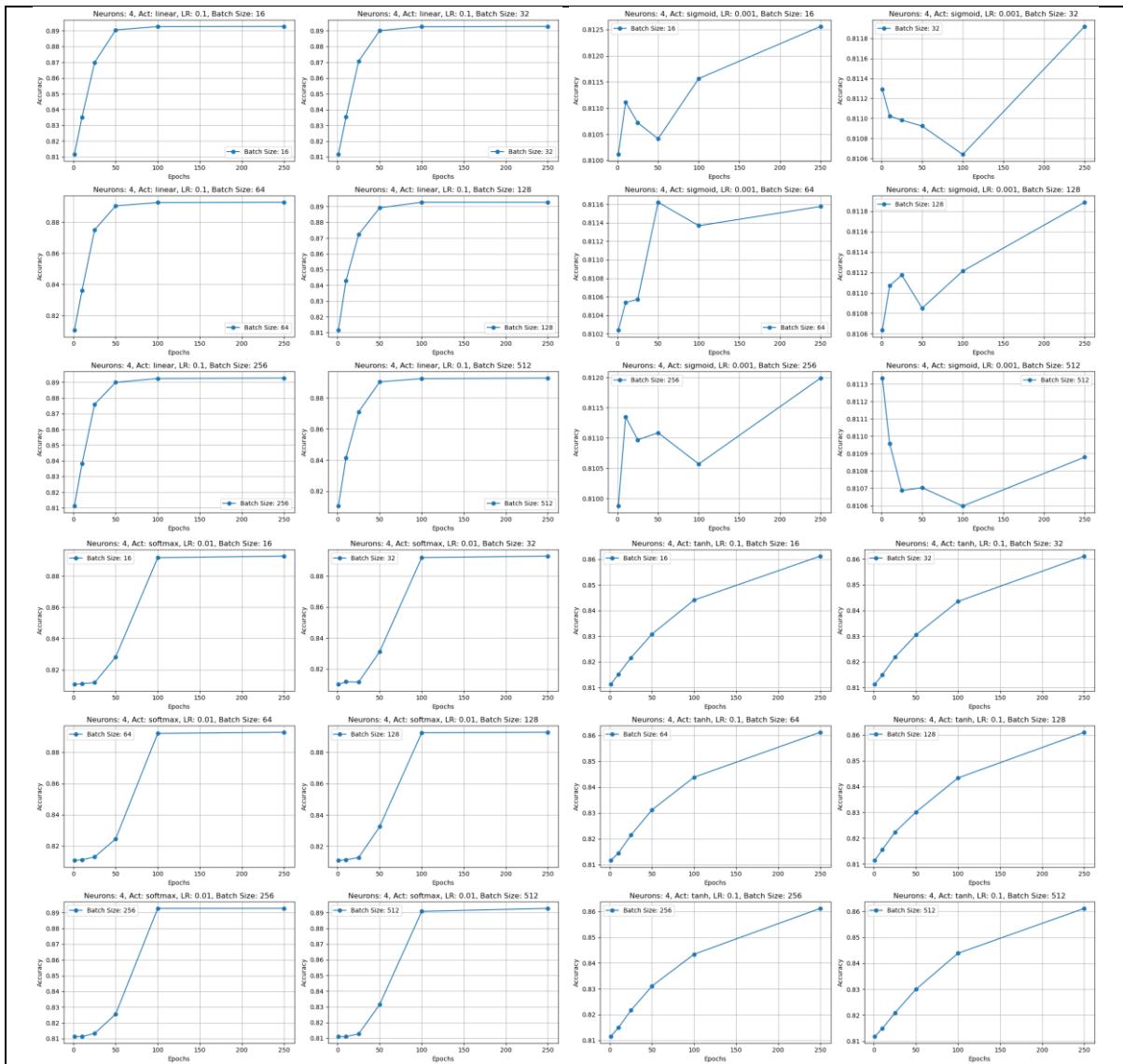
                plt.figure(figsize=(12, 12))
                for i, batch_size in enumerate(batch_sizes, 1):
                    plt.subplot(3, 2, i) # Buat grid 3x2
                    data = filtered_data[filtered_data['batch_size'] == batch_size]
                    plt.plot(data['epochs'], data['accuracy'], marker='o', label=f'Batch Size: {batch_size}')
                    plt.title(f'Neurons: {neuron}, Act: {activation}, LR: {lr}, Batch Size: {batch_size}')
                    plt.xlabel('Epochs')
                    plt.ylabel('Accuracy')
                    plt.legend()
                    plt.grid()

                plt.tight_layout()
                plt.show()

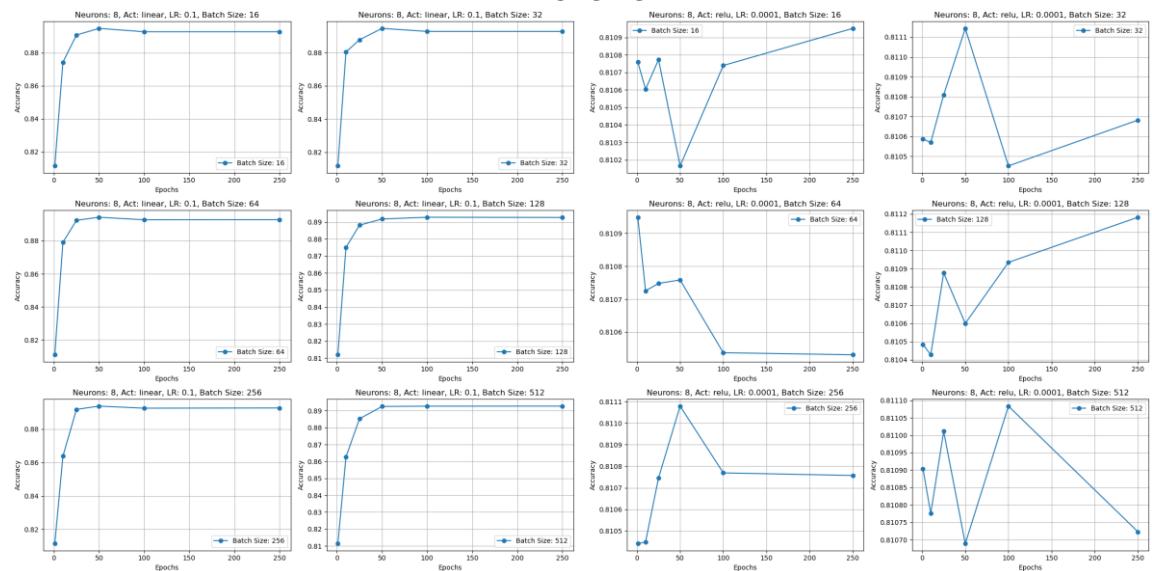
    ✓ 1m 4.2s
```

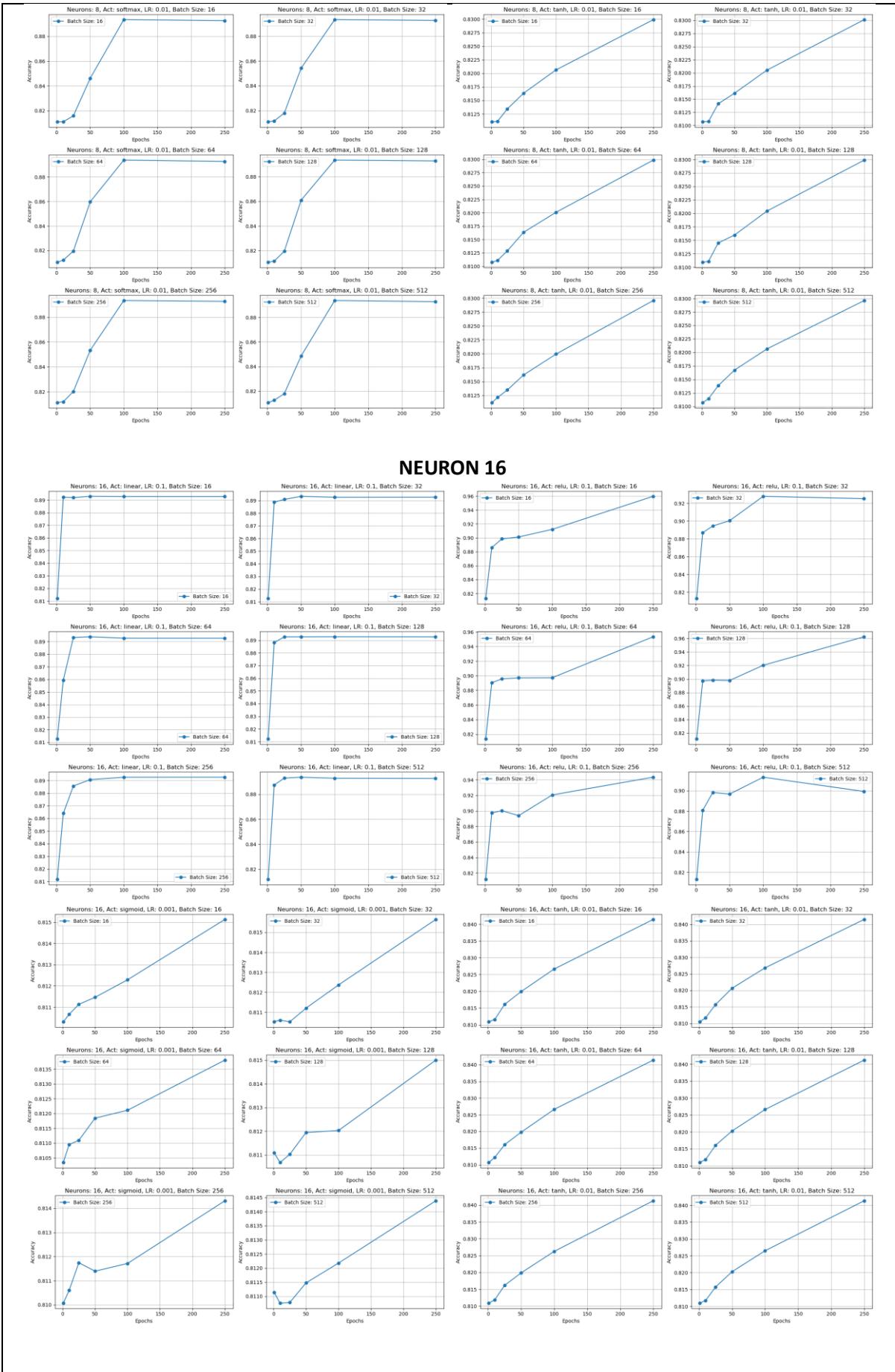
Output:

NEURON 4

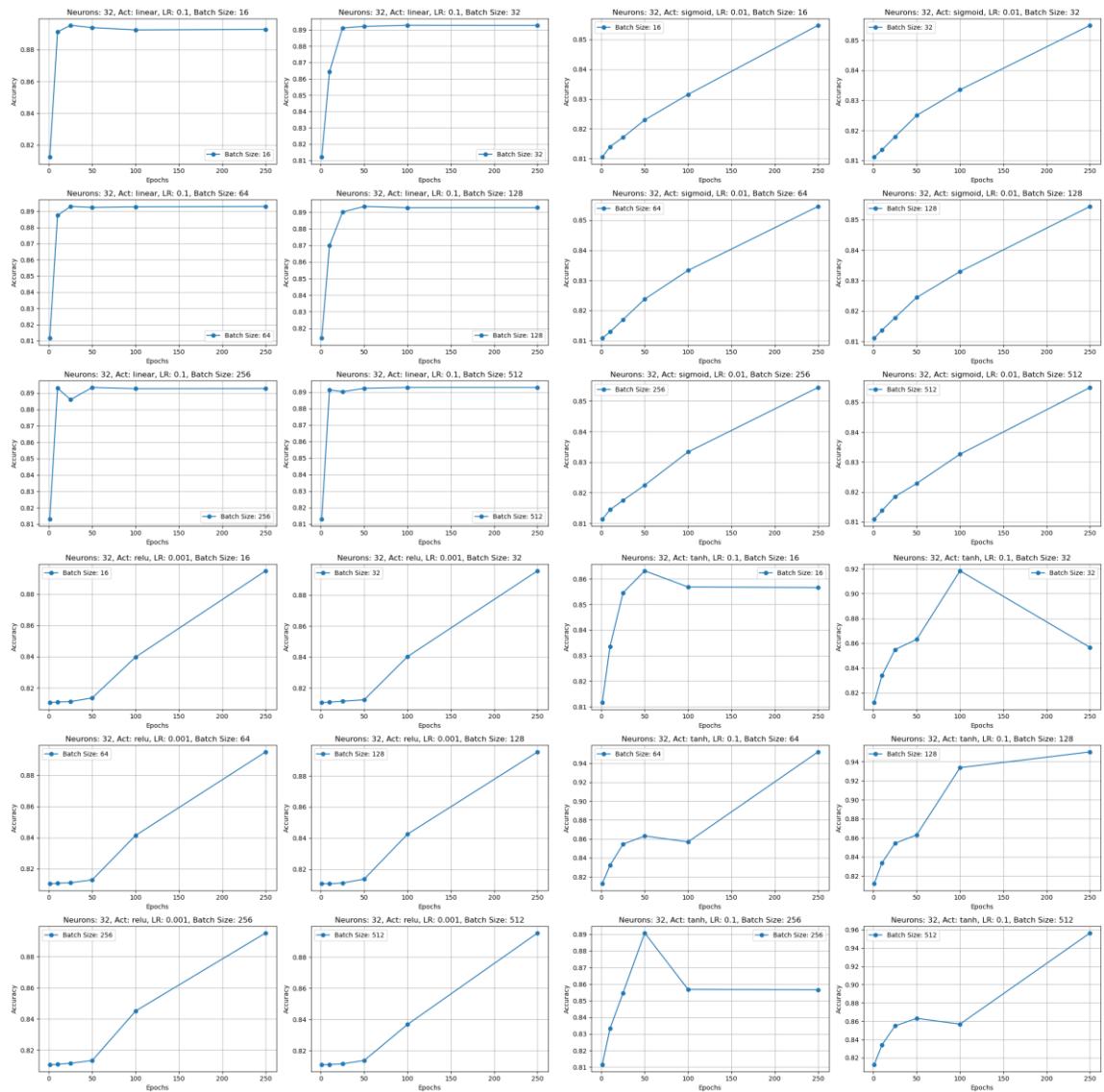


NEURON 8

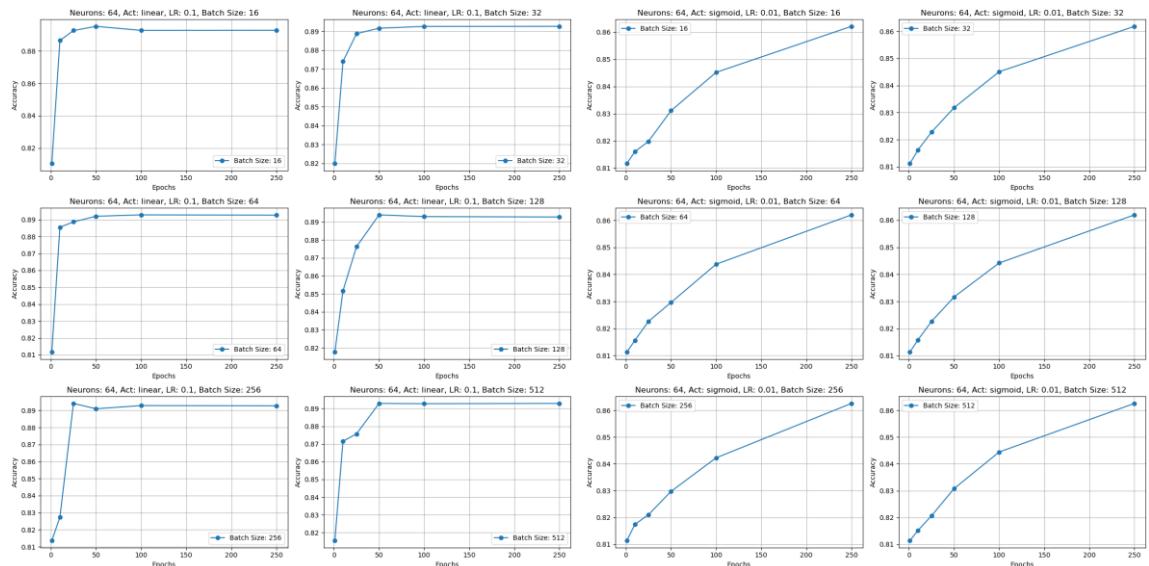


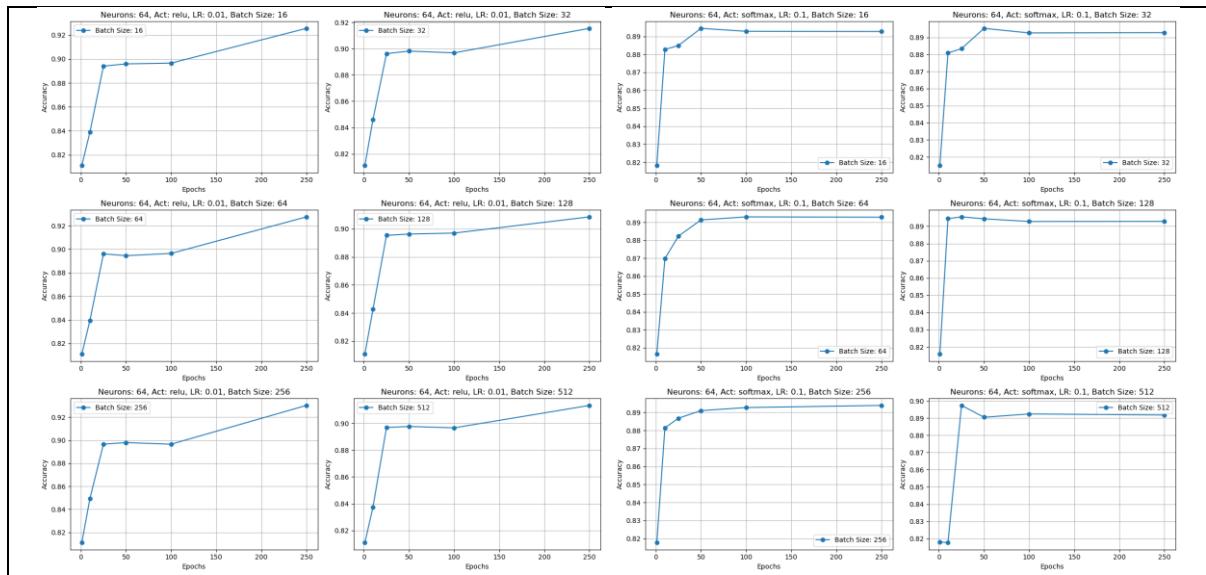


NEURON 32



NEURON 64





Analisis:

Beberapa grafik akurasi terlihat sama meskipun ada perubahan pada kombinasi parameter model dapat terjadi karena beberapa alasan. Pertama, perubahan ukuran batch mungkin tidak terlalu mempengaruhi akurasi jika model sudah cukup stabil dalam proses pembelajaran atau dataset tidak memiliki kompleksitas yang tinggi. Kedua, jika model terlalu sederhan (misalnya dengan jumlah neuron yang terbatas atau lapisan tersembunyi yang sedikit) maka model tersebut tidak memiliki kapasitas yang cukup untuk menangkap pola-pola kompleks dalam data. Dalam kasus ini, meskipun ada variasi pada parameter seperti ukuran batch atau fungsi aktivasi, model tetap tidak dapat meningkatkan akurasinya secara signifikan. Ketiga yaitu pemilihan learning rate yang kurang optimal dalam data juga bisa menyebabkan performa model tetap stagnan meskipun ada perubahan parameter.

```
# Load results from CSV files
results_df_1 = pd.read_csv("mlp_regression_hidden_layer_1.csv")
results_df_2 = pd.read_csv("mlp_regression_hidden_layer_2.csv")
results_df_3 = pd.read_csv("mlp_regression_hidden_layer_3.csv")

# Sort and display top 10 results for each hidden layer configuration
print("\nTop 10 Results for Hidden Layer 1")
print(results_df_1.sort_values(by='mse', ascending=True).head(10))

print("\nTop 10 Results for Hidden Layer 2")
print(results_df_2.sort_values(by='mse', ascending=True).head(10))

print("\nTop 10 Results for Hidden Layer 3")
print(results_df_3.sort_values(by='mse', ascending=True).head(10))
```

Output:

Top 10 Results for Hidden Layer 1							
	layers	neurons	activation	epochs	lr	batch_size	mae \
2860	1	64	tanh	250	0.1	256	57.655377
2715	1	64	relu	250	0.1	128	62.499385
2861	1	64	tanh	250	0.1	512	58.499928
2858	1	64	tanh	250	0.1	64	58.457275
2859	1	64	tanh	250	0.1	128	59.801395
2857	1	64	tanh	250	0.1	32	60.650358
2856	1	64	tanh	250	0.1	16	60.796276
2712	1	64	relu	250	0.1	16	72.907483
2280	1	32	tanh	250	0.1	16	68.69963
2283	1	32	tanh	250	0.1	128	68.503149
	mse	r2	accuracy				
2860	7277.350874	0.770180	0.940987				
2715	7463.959941	0.764287	0.936029				
2861	7483.853722	0.763658	0.940123				
2858	7498.687426	0.763190	0.940167				
2859	7762.487235	0.754859	0.938791				
2857	7972.639923	0.748222	0.937922				
2856	8074.758508	0.744998	0.937772				
2712	10547.196472	0.666917	0.925376				
2280	10575.236878	0.6666032	0.929775				
2283	10681.106604	0.662688	0.929884				
Top 10 Results for Hidden Layer 2							
	layers	neurons	activation	epochs	lr	batch_size	mae \
3291	2	64	relu	250	0.1	128	30.599988
3288	2	64	relu	250	0.1	16	31.301783
3289	2	64	relu	250	0.1	32	31.938192

Analisis:

Kode ini bertujuan untuk memuat dan menganalisis hasil eksperimen model dengan jumlah lapisan tersembunyi yang berbeda, yaitu 1, 2, dan 3 lapisan. Pertama, data hasil eksperimen dimuat dari tiga file CSV yang berisi konfigurasi model yang berbeda, masing-masing mewakili satu jumlah lapisan tersembunyi. Kemudian, untuk setiap konfigurasi lapisan tersembunyi, data diurutkan berdasarkan nilai Mean Squared Error (MSE) di mana MSE yang lebih rendah menunjukkan model dengan kesalahan prediksi yang lebih kecil. Setelah itu, kode ini menampilkan 10 hasil terbaik berdasarkan nilai MSE terkecil untuk setiap konfigurasi lapisan tersembunyi. Tujuan dari langkah ini adalah untuk mengevaluasi dan membandingkan performa model dengan jumlah lapisan tersembunyi yang berbeda.

```
# Combine all results into a single DataFrame
combined_results = pd.concat([results_df_1, results_df_2, results_df_3])

# Sort combined results by MSE and display top 10
top_10_results = combined_results.sort_values(by='mse', ascending=True).head(10)

# Display the top 10 results
print("Top 10 Results Across All Hidden Layer Configurations:")
print(top_10_results)
```

✓ 0.0s

Python

Output:

Top 10 Results Across All Hidden Layer Configurations:							
	layers	neurons	activation	epochs	lr	batch_size	mae
3290	3	64	relu	250	0.1	64	28.756235
3291	3	64	relu	250	0.1	128	29.089365
3288	3	64	relu	250	0.1	16	29.016921
3291	2	64	relu	250	0.1	128	30.599988
2572	3	32	relu	250	0.1	256	29.478266
3288	2	64	relu	250	0.1	16	31.301783
3292	3	64	relu	250	0.1	256	32.694788
3289	2	64	relu	250	0.1	32	31.938192
2571	3	32	relu	250	0.1	128	33.217072
3292	2	64	relu	250	0.1	256	33.181610
	mse	r2	accuracy				
3290	1931.679271	0.938997	0.970567				
3291	1982.754161	0.937384	0.970226				
3288	2031.185352	0.935855	0.970300				
3291	2160.824417	0.931761	0.968680				
2572	2163.219329	0.931685	0.969828				
3288	2250.596476	0.928926	0.967961				
3292	2293.161691	0.927581	0.966536				
3289	2308.485306	0.927098	0.967310				
2571	2386.579862	0.924631	0.966001				
3292	2481.493419	0.921634	0.966037				

Analisis:

Kode ini memuat hasil dari tiga file CSV yang berisi data konfigurasi model dengan lapisan tersembunyi yang berbeda (1, 2, dan 3 lapisan). Setiap dataset diurutkan berdasarkan nilai Mean Squared Error (MSE), dan 10 hasil teratas ditampilkan untuk setiap konfigurasi lapisan tersembunyi. Selanjutnya, hasil dari ketiga konfigurasi digabungkan ke dalam satu DataFrame dan diurutkan kembali berdasarkan nilai MSE untuk menampilkan 10 hasil terbaik secara keseluruhan. Ini memungkinkan perbandingan antara hasil model yang berbeda berdasarkan jumlah lapisan tersembunyi.

```

# Subplot 1: MSE
plt.subplot(1, 2, 1)
sns.barplot(x=top_10_results.index, y=top_10_results['mse'], palette="pastel")
plt.title("Top 10 Results: Mean Squared Error (MSE)")
plt.xlabel("Configuration Index")
plt.ylabel("MSE")
plt.xticks(rotation=45)

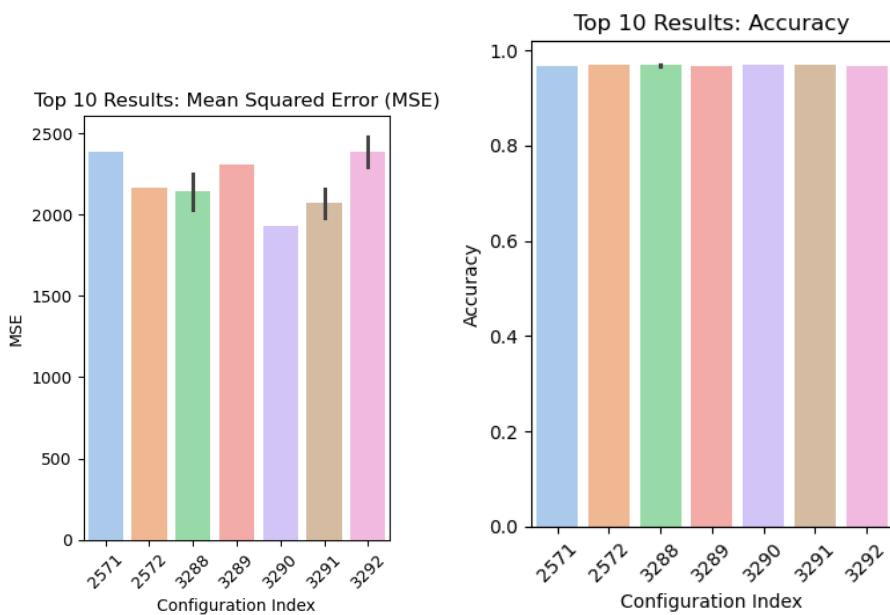
# Adjust layout and display the plots
plt.tight_layout()
plt.show()
✓ 0.1s

# Subplot 2: Accuracy
plt.subplot(1, 2, 2)
sns.barplot(x=top_10_results.index, y=top_10_results['accuracy'], palette="pastel")
plt.title("Top 10 Results: Accuracy")
plt.xlabel("Configuration Index")
plt.ylabel("Accuracy")
plt.xticks(rotation=45)

# Adjust layout and display the plots
plt.tight_layout()
plt.show()
✓ 0.1s

```

Output:



Analisis:

Kode ini membuat dua subplot yang menampilkan perbandingan hasil model berdasarkan dua metrik: Mean Squared Error (MSE) dan Accuracy. Pada subplot pertama, grafik batang menggambarkan peringkat 10 konfigurasi terbaik berdasarkan nilai MSE, yang menunjukkan seberapa besar kesalahan kuadrat rata-rata antara prediksi dan nilai aktual. Pada subplot kedua, grafik batang menggambarkan peringkat 10 konfigurasi terbaik berdasarkan akurasi, yang mengukur seberapa dekat prediksi model dengan nilai aktual. Kedua grafik ini diatur dengan rotasi label pada sumbu x untuk meningkatkan keterbacaan, dan layout ditata agar lebih rapi dan jelas.