

LAPORAN PENGERJAAN TUGAS GROUP PROJECT FP

Kelompok 10

PMA C



DISUSUN OLEH:

Nabila Kumala Gantari (5026211016)

Vasya Ayu Karmina (5026211091)

Zahrina Candrakanti (5026211100)

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

SURABAYA

2023

Daftar Isi

I. Overview Tugas.....	4
II. Deskripsi Permasalahan.....	4
III. Metodologi.....	4
A. Preprocessing Data.....	5
1. Import Library.....	5
2. Load Dataset.....	6
3. Display Data Type Information.....	6
4. Data Duplicate Check.....	7
6. Data Type Adjustment.....	8
● Change Data Type to Integer.....	8
● Datetime Formatting.....	8
7. Outlier Values Check.....	9
8. Normalisasi dataset.....	9
a. RNN.....	9
b. LSTM.....	10
c. GRU.....	11
9. Time Series (Sequence).....	12
a. RNN.....	12
b. LSTM.....	12
● Skenario 80 % Train.....	12
● Skenario 70% Train.....	13
● Skenario 60 % Train.....	13
c. GRU.....	14
B. Pembagian dataset.....	14
C. Deskripsi Model Deep Learning (RNN, LSTM, GRU).....	15
1. RNN.....	15
2. LSTM.....	16
3. GRU.....	16
D. Pemodelan Support Vector Regression (SVR).....	17
E. Evaluasi Model.....	18
F. Forecasting 12 Periode Berikutnya.....	18
G. Analisis Hasil dan Grafik.....	18
IV. Hasil Eksperimen dan Perbandingan.....	19
1. SVR.....	20
● Skenario 80 % Train.....	20
● Skenario 70 % Train.....	27
● Skenario 60 % Train.....	33
2. RNN.....	41
● Skenario 80 % Train.....	41
● Skenario 70 % Train.....	48

• Skenario 60 % Train.....	55
3. LSTM.....	63
• Skenario 80 % Train.....	63
• Skenario 70 % Train.....	70
• Skenario 60 % Train.....	78
4. GRU.....	86
• Skenario 80 % Train.....	86
• Skenario 70 % Train.....	92
• Skenario 60 % Train.....	98
V. Hasil Forecast Untuk 12 Periode.....	105
1. SVR.....	106
• Skenario 80 % Train.....	106
• Skenario 70 % Train.....	107
• Skenario 60 % Train.....	108
2. RNN.....	109
• Skenario 80 % Train.....	109
• Skenario 70 % Train.....	110
• Skenario 60 % Train.....	110
3. LSTM.....	111
• Skenario 80 % Train.....	111
• Skenario 70 % Train.....	112
• Skenario 60 % Train.....	113
4. GRU.....	114
• Skenario 80 % Train.....	114
• Skenario 70 % Train.....	115
• Skenario 60 % Train.....	115
VI. Hasil Analisis dan Kesimpulan.....	116

I. Overview Tugas

Pada Final Project ini, dilakukan peramalan (*forecasting*) menggunakan metode deep learning (RNN, LSTM, GRU) dan Support Vector Regression (SVR) untuk variabel ‘total_visitor’ dengan mempertimbangkan ‘total_accomodation’ sebagai faktor yang mempengaruhi. Penelitian ini menggunakan dataset provinsi NTT yang mencakup pertumbuhan pengunjung dan akomodasi selama sembilan tahun terakhir, dari tahun 2014 hingga 2022. Dataset tersebut terdiri dari tiga kolom, yaitu ‘datetime’, ‘total_visitor’, dan ‘total_accomodation’. Peramalan dilakukan pada tiga skenario berbeda untuk setiap metode dengan pembagian dataset train-test menggunakan rasio 80%-20%, 70%-30%, dan 60%-40%.

II. Deskripsi Permasalahan

Permasalahan yang diangkat dalam Final Project ini melibatkan prediksi jumlah pengunjung ke Provinsi Nusa Tenggara Timur (NTT) di masa depan, sebuah aspek yang sangat krusial untuk perencanaan dan pengembangan sektor pariwisata di wilayah tersebut. Antisipasi terhadap fluktuasi jumlah pengunjung dapat memberikan wawasan yang berharga bagi pihak berwenang dalam mengambil keputusan strategis terkait infrastruktur, promosi pariwisata, dan pengembangan ekonomi lokal. Faktor-faktor yang dapat memengaruhi jumlah pengunjung ke suatu daerah sangat kompleks, dan dalam konteks ini, penelitian akan memberikan fokus pada ketersediaan akomodasi sebagai salah satu variabel kunci. Pengelolaan dan peningkatan ketersediaan akomodasi diharapkan dapat menjadi strategi yang efektif untuk menarik dan mempertahankan minat pengunjung, sekaligus memberikan landasan yang kuat untuk perencanaan jangka panjang di NTT.

Selama penelitian, analisis mendalam terhadap hubungan antara ketersediaan akomodasi dan jumlah pengunjung akan dilakukan. Metodologi penelitian ini kemungkinan akan melibatkan pengumpulan data historis, analisis tren, dan penggunaan model prediktif untuk meramalkan potensi pertumbuhan pariwisata di NTT. Dengan demikian, penelitian ini diharapkan dapat memberikan kontribusi signifikan dalam pemahaman faktor-faktor yang mempengaruhi pariwisata di provinsi tersebut, memberikan landasan yang kuat untuk pengembangan strategi yang berkelanjutan dan efektif dalam mendukung pertumbuhan sektor pariwisata di masa depan.

Data yang digunakan dalam penelitian ini adalah dataset provinsi NTT yang mencakup pertumbuhan pengunjung dan akomodasi selama sembilan tahun terakhir, dari tahun 2014 hingga 2022. Dataset tersebut terdiri dari tiga kolom, yaitu ‘datetime’, ‘total_visitor’, dan ‘total_accomodation’. Metode yang digunakan dalam penelitian ini adalah deep learning (RNN, LSTM, GRU) dan Support Vector Regression (SVR). Ketiga metode tersebut dipilih karena memiliki kemampuan untuk mempelajari pola data yang kompleks. Peramalan dilakukan pada tiga skenario berbeda untuk setiap metode dengan pembagian dataset train-test menggunakan rasio 80%-20%, 70%-30%, dan 60%-40%.

III. Metodologi

Metodologi yang digunakan dalam peramalan pengunjung pada dataset provinsi NTT melibatkan langkah-langkah sebagai berikut:

A. Preprocessing Data

1. Import Library

Untuk melakukan pemeriksaan tipe data dan analisis awal data, pertama-tama, kita perlu mengimpor library yang diperlukan dalam lingkup analisis data. Sebagai contoh, kita dapat menggunakan library seperti pandas untuk manipulasi data dan pengecekan tipe data seperti dibawah ini.

```
[ ] import numpy as np
import sklearn.metrics as sm
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
from sklearn.svm import SVR
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle
from scipy import stats
import pyarrow as pa
from sklearn import metrics
from sklearn.metrics import f1_score, roc_auc_score
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import make_column_transformer
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
import tensorflow as tf
import sklearn.preprocessing
from sklearn.metrics import r2_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
from tensorflow.keras.callbacks import EarlyStopping
from keras.layers import Dense, Dropout, SimpleRNN, LSTM, GRU
from keras.models import Sequential
from tensorflow import keras
```

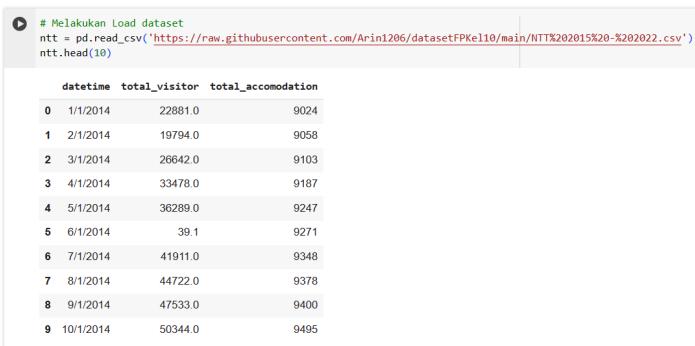
Gambar 1. Import Library

- numpy dan pandas digunakan untuk manipulasi dan analisis data numerik.
- matplotlib dan seaborn digunakan untuk visualisasi data.
- datetime digunakan untuk manipulasi data tanggal dan waktu.
- SVR (Support Vector Regression) dari sklearn.svm digunakan untuk model regresi.
- Modul-modul lain seperti StandardScaler, shuffle, stats, pyarrow, metrics, f1_score, roc_auc_score, make_pipeline, Pipeline, SimpleImputer, make_column_transformer, StratifiedKFold, cross_val_score, GridSearchCV, RandomizedSearchCV, tf (TensorFlow), MinMaxScaler, train_test_split, mean_absolute_error, mean_squared_error, EarlyStopping, Dense, Dropout, SimpleRNN, LSTM, GRU, Sequential, dan keras digunakan untuk preprocessing data dan pembuatan model machine learning.
- train_test_split digunakan untuk membagi data menjadi set pelatihan dan pengujian.
- mean_squared_error digunakan untuk mengukur kinerja model dengan menghitung rata-rata kuadrat dari perbedaan antara nilai prediksi dan nilai sebenarnya.
- Modul-modul di atas digunakan untuk membangun model neural network dengan TensorFlow dan Keras.

- SimpleImputer digunakan untuk menangani nilai yang hilang.
- StandardScaler dan MinMaxScaler digunakan untuk penskalaan fitur.
- make_column_transformer digunakan untuk membuat transformer khusus untuk setiap kolom.
- StratifiedKFold digunakan untuk validasi silang.
- GridSearchCV dan RandomizedSearchCV digunakan untuk penalaan parameter model.

2. Load Dataset

Setelah mengimpor library, langkah selanjutnya adalah membuat dataframe baru yang akan kita namai 'ntt' dan mengisi dataframe tersebut dengan data yang akan dianalisis.



```
# Melakukan Load dataset
ntt = pd.read_csv('https://raw.githubusercontent.com/Arin1206/datasetFPKe10/main/NTT%202015%20-%202022.csv')
ntt.head(10)
```

	datetime	total_visitor	total_accommodation
0	1/1/2014	22881.0	9024
1	2/1/2014	19794.0	9058
2	3/1/2014	26642.0	9103
3	4/1/2014	33478.0	9187
4	5/1/2014	36289.0	9247
5	6/1/2014	39.1	9271
6	7/1/2014	41911.0	9348
7	8/1/2014	44722.0	9378
8	9/1/2014	47533.0	9400
9	10/1/2014	50344.0	9495

Gambar 2. Load Dataset

pd.read_csv adalah fungsi Pandas yang digunakan untuk membaca data dari file CSV. Di sini, Anda memberikan URL file CSV sebagai argumen untuk membaca data dari sumber eksternal. File CSV yang dibaca memiliki nama 'NTT 2015 - 2022.csv'. Data yang diambil dari URL tersebut kemudian disimpan dalam DataFrame yang dinamai 'ntt'.

3. Display Data Type Information

Setelah membuat dataframe, langkah selanjutnya adalah melakukan pengecekan terhadap missing values. Pemeriksaan ini penting untuk memastikan bahwa data yang digunakan dalam analisis tidak memiliki nilai yang hilang atau tidak lengkap, sehingga memastikan konsistensi dan akurasi hasil analisis.

```
[ ] ntt.info()
```

#	Column	Non-Null Count	Dtype
0	datetime	108 non-null	object
1	total_visitor	108 non-null	float64
2	total_accomodation	108 non-null	int64

dtypes: float64(1), int64(1), object(1)
memory usage: 2.7+ KB

Gambar 3. Display Data Type Information

Fungsi info() memberikan ringkasan tentang struktur dataset, termasuk jumlah entri, tipe data, dan apakah terdapat nilai yang hilang.

4. Data Duplicate Check

Untuk memeriksa apakah terdapat data duplikat dalam DataFrame, Anda dapat menggunakan metode duplicated() dan sum() dari Pandas.

```
[ ] jumlah_baris_duplikat = ntt.duplicated().sum()
print("Jumlah baris yang memiliki data sama: {} baris".format(jumlah_baris_duplikat))
```

Jumlah baris yang memiliki data sama: 0 baris

Gambar 4. Duplicate Check

Hasil keluaran total_duplicates menunjukkan jumlah baris yang diidentifikasi sebagai duplikat dalam DataFrame 'ntt' hasilnya 0, itu berarti tidak ada data duplikat

5. Missing Values Check

Dengan menggunakan metode isnull(), setiap elemen dalam DataFrame 'ntt' diberi label True jika nilainya adalah NaN (nilai yang hilang), dan False jika tidak. Dengan menambahkan sum() pada hasil dari isnull(), mendapatkan jumlah nilai True (yang merepresentasikan nilai yang hilang) untuk setiap kolom.

```
[ ] jumlah_baris_duplikat = ntt.duplicated().sum()
print("Jumlah baris yang memiliki data sama: {} baris".format(jumlah_baris_duplikat))
```

Jumlah baris yang memiliki data sama: 0 baris

Gambar 5. Missing Value

Dari hasil query di atas menghasilkan jumlah baris yang memiliki nilai null sebanyak 0 baris yang sudah di total dengan setiap kolom yang ada.

6. Data Type Adjustment

- Change Data Type to Integer

Mengubah tipe data menjadi integer adalah proses konversi nilai-nilai dalam suatu dataset agar memiliki tipe data integer. Tipe data integer menyimpan nilai bilangan bulat tanpa desimal, yang biasanya digunakan untuk merepresentasikan data kategorikal, identifier, atau nilai yang hanya dapat berupa bilangan bulat. Proses ini menjadi penting ketika data yang semula memiliki tipe data lain (seperti float atau object) ingin diubah agar lebih sesuai dengan sifat data yang sebenarnya.

```
[7] # Mengubah kolom 'total_visitor' dari float menjadi integer
      ntt['total_visitor'] = ntt['total_visitor'].astype(int)

[8] ntt.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108 entries, 0 to 107
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   datetime          108 non-null    object  
 1   total_visitor     108 non-null    int64  
 2   total_accommodation 108 non-null  int64  
dtypes: int64(2), object(1)
memory usage: 2.7+ KB
```

Gambar 6. Data Type to Integer

Maka Dari itu, didapatkan perubahan pada kolom ‘total_visitor’ dan ‘total_accommodation’ menjadi integer yang ditunjukkan dengan fungsi.

- Datetime Formatting

Datetime formatting digunakan untuk mengubah representasi teks dari objek datetime menjadi format yang diinginkan. Objek datetime biasanya digunakan untuk menyimpan dan memanipulasi informasi tanggal dan waktu dalam pemrograman. Dengan datetime formatting, dapat mengontrol cara tanggal dan waktu ditampilkan atau disimpan dalam bentuk string.

```
[9] ntt['datetime'] = pd.to_datetime(ntt['datetime'], format='%m/%d/%Y')
      ntt.head()

      datetime  total_visitor  total_accommodation
0 2014-01-01        22881              9024
1 2014-02-01        19794              9058
2 2014-03-01        26642              9103
3 2014-04-01        33478              9187
4 2014-05-01        36289              9247
```

Gambar 7. Datetime Formatting

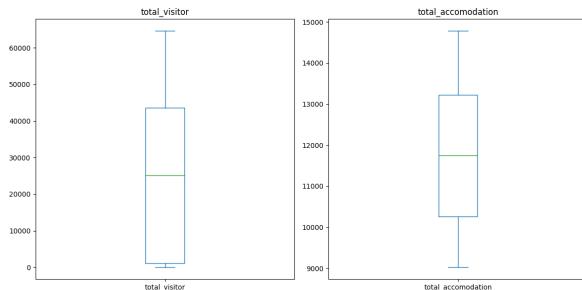
Dari eksekusi kode diatas didapatkan Format Datetime sudah berubah menjadi format sesuai yang telah ditentukan yaitu tahun-bulan-tanggal

7. Outlier Values Check

Selanjutnya, analisis outlier dapat dilakukan untuk mengidentifikasi dan menangani nilai ekstrim yang mungkin mempengaruhi hasil analisis secara signifikan. Outlier merupakan nilai yang jauh berbeda dari nilai-nilai lain dalam dataset dan dapat memengaruhi distribusi data secara keseluruhan.

```
data_check = ['total_visitor', 'total_accomodation']
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))
for i, data in enumerate(data_check):
    ntt[data].plot(kind='box', ax=axes[i])
    axes[i].set_title(data)
plt.tight_layout()
plt.show()
```

Gambar 8. Outlier Values Check



Gambar 9. Box Plot Outlier

Didapatkan dari pengecekan Outlier menggunakan boxplot diatas, bahwa dataset NTT sendiri tidak memiliki outlier yang harus ditangani sehingga persebaran data dianggap normal atau standard.

8. Normalisasi dataset

Normalisasi dataset adalah suatu proses dalam analisis data yang bertujuan untuk mengubah nilai-nilai dalam dataset ke dalam rentang yang seragam atau skala yang standar. Tujuan utama normalisasi adalah untuk memastikan bahwa variabel-variabel dalam dataset memiliki distribusi yang serupa atau setidaknya dapat dibandingkan secara konsisten, sehingga mencegah variabel dengan nilai yang sangat besar atau sangat kecil mendominasi analisis.

a. RNN

Dilihat dari distribusi data pada Outlier Checks pada praproses data RNN diperlukan normalisasi data pada dataset NTT agar rentang tidak terlalu berbeda jauh satu sama lainnya dan dapat mudah diproses yaitu dengan menggunakan code sebagai berikut

```
[213] # Normalize the data
scaler = MinMaxScaler()
dataset_scaled = ntt.copy()
for column in ntt.columns:
    dataset_scaled[[column]] = scaler.fit_transform(ntt[[column]])
```

Gambar 10. RNN Scaled

Dari gambar di atas didapatkan dataframe terbaru bernama dataset_scaled, dimana hasil dari pemanggilan fungsi MinMaxScaler untuk scaling dataframe ntt.copy. MinMaxScaler adalah salah satu metode normalisasi yang digunakan dalam pengolahan data dan analisis data. Tujuan utama dari MinMaxScaler adalah untuk mentransformasi nilai-nilai dalam dataset sehingga mereka dapat dinyatakan dalam rentang tertentu, seringkali dalam rentang 0 hingga 1. Teknik ini berguna untuk menangani variabel-variabel dengan skala yang berbeda dan menjaga konsistensi antara variabel-variabel tersebut. Maka untuk hasilnya sebagai berikut, menunjukkan bahwa valuenya sudah berubah menjadi rentang 0-1

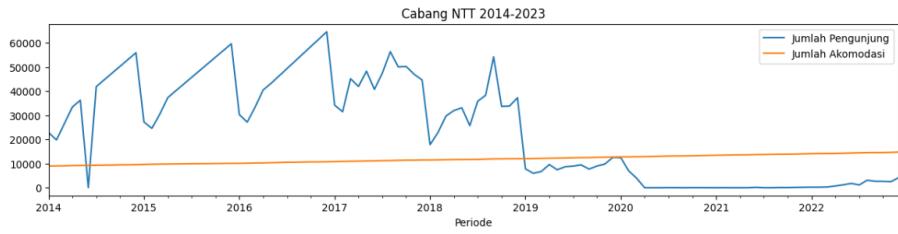
[214] dataset_scaled		
	total_visitor	total_accomodation
datetime		
2014-01-01	0.353981	0.000000
2014-02-01	0.306224	0.005905
2014-03-01	0.412166	0.013720
2014-04-01	0.517923	0.028308
2014-05-01	0.561410	0.038729
...
2022-08-01	0.048268	0.958145
2022-09-01	0.041337	0.964050
2022-10-01	0.040935	0.965960
2022-11-01	0.038847	0.978638
2022-12-01	0.066523	1.000000

108 rows × 2 columns

Gambar 11. Hasil RNN Scaled

b. LSTM

Sebelum melakukan normalisasi, dilakukan pengecekan terhadap persebaran data sesuai rentang waktu, dengan hasil yang menunjukkan bahwa terdapat penurunan grafik secara drastis pada beberapa periode waktu, terutama ketika periode waktu merujuk pada masa pandemi, mulai tahun 2020.



Gambar 12. Persebaran Data

Pada metode LSTM, proses normalisasi dapat langsung dilakukan dengan menggunakan kode berikut. Pada normalisasi ini tidak disebutkan rentang nominal normalisasi karena sebaran data yang kurang baik.

```
# Normalize the data
scaler_lstm = MinMaxScaler()
ntt_scaled = scaler_lstm.fit_transform(ntt_lstm)

ntt_scaled
```

Gambar 13. Coding Normalisasi LSTM

```
array([[3.53981343e-01, 0.00000000e+00],
       [3.06223797e-01, 5.90482807e-03],
       [4.12166030e-01, 1.37200417e-02],
       [5.17922616e-01, 2.83084404e-02],
       [5.61410294e-01, 3.87287253e-02],
       [6.04897972e-04, 4.28968392e-02],
       [6.48385650e-01, 5.62695380e-02],
       [6.91873327e-01, 6.14796804e-02],
```

Gambar 14. Hasil Normalisasi LSTM

Dengan menggunakan kode tersebut, didapatkan *output* berupa *array* dengan nominal hasil normalisasi.

c. GRU

Dilihat dari distribusi data pada Outlier Checks pada pra proses data GRU diperlukan normalisasi data pada dataset NTT agar rentang tidak terlalu berbeda jauh satu sama lainnya dan dapat mudah diproses yaitu dengan menggunakan code sebagai berikut

```
[1210] # Normalize the data
scaler = StandardScaler()
dataset_scaled = ntt.copy()
for column in ntt.columns:
    dataset_scaled[[column]] = scaler.fit_transform(ntt[[column]])
```

Gambar 15. GRU Scaled

Dari gambar diatas didapatkan dataframe terbaru bernama *dataset_scaled*, dimana hasil dari pemanggilan fungsi *MinMaxScaler* untuk scaling dataframe *ntt.copy*. *MinMaxScaler* adalah salah satu metode normalisasi yang digunakan dalam pengolahan data dan analisis data. Tujuan utama dari

MinMaxScaler adalah untuk mentransformasi nilai-nilai dalam dataset sehingga mereka dapat dinyatakan dalam rentang tertentu, seringkali dalam rentang 0 hingga 1. Teknik ini berguna untuk menangani variabel-variabel dengan skala yang berbeda dan menjaga konsistensi antara variabel-variabel tersebut.

9. Time Series (Sequence)

a. RNN

Pada metode RNN, pembuatan *time sequence* dilakukan sebelum skenario *splitting* data. Hal tersebut dikarenakan langkah ini dilakukan sama pada semua kondisi pembagian data menjadi data *train* dan *test*.

```
# Define sequence length and features
sequence_length = 6 # Number of time steps in each sequence
num_features = len(dataset_scaled.columns)

# Create sequences and corresponding labels
sequences = []
labels = []
for i in range(len(dataset_scaled) - sequence_length):
    # Mendapatkan 5 data periode sebelumnya
    seq = dataset_scaled.iloc[i:i+sequence_length]
    # Mendapatkan data periode ke-6 sebagai label, cuma kolom total_visitor
    label = dataset_scaled.iloc[i+sequence_length, 1] # '_tempm' column index, kalau prediksi lebih dari satu, ganti angkanya
    sequences.append(seq)
    labels.append(label)
```

Gambar 16. Coding Pembuatan Time Series RNN

Pada kode tersebut, panjang sekuens yang ditetapkan sebesar 6 periode sebelumnya. Hal tersebut dilakukan untuk menyesuaikan jumlah data *test* sehingga dapat memenuhi jumlah minimum peramalan untuk 12 periode waktu berikutnya.

b. LSTM

Pada metode LSTM, pembuatan *time sequence* dilakukan pada masing-masing skenario *splitting* data. Hal tersebut dikarenakan langkah ini dilakukan menyesuaikan kondisi pembagian data menjadi data *train* dan *test*.

- Skenario 80 % Train

```
# convert an array of values into a dataset matrix
def create_dataset(data, look_back):
    dataX, dataY = [], []
    for i in range(len(data)-look_back):
        dataX.append(data[i:(i+look_back), 0:data.shape[1]])
        dataY.append(data[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

# reshape into X=t and Y=t+1
look_back = 10
X_train80, y_train80 = create_dataset(train80, look_back)
X_test20, y_test20 = create_dataset(test20, look_back)
```

Gambar 17. Coding Pembuatan Time Series LSTM Skenario 1

Pada kode tersebut, panjang sekuens yang ditetapkan ditunjukkan dengan variabel ‘look_back’ dengan besaran 10 periode sebelumnya.

Hal tersebut dilakukan untuk menyesuaikan jumlah data *test* sehingga dapat memenuhi jumlah minimum peramalan untuk 12 periode waktu berikutnya.

- Skenario 70% Train

```
# convert an array of values into a dataset matrix
def create_dataset(data, look_back):
    dataX, dataY = [], []
    for i in range(len(data)-look_back-1):
        dataX.append(data[i:(i+look_back), 0:data.shape[1]])
        dataY.append(data[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

# reshape into X=t and Y=t+1
look_back = 12
X_train70, y_train70 = create_dataset(train70, look_back)
X_test30, y_test30 = create_dataset(test30, look_back)
```

Gambar 18. Coding Pembuatan Time Series LSTM Skenario 2

Pada kode tersebut, panjang sekuens yang ditetapkan ditunjukkan dengan variabel ‘look_back’ dengan besaran 12 periode sebelumnya, menyesuaikan kondisi persebaran data. Namun, pada skenario ini terdapat sedikit perbedaan dengan skenario sebelumnya, dimana *range* yang digunakan untuk skenario ini adalah hasil dari jumlah baris data dikurangi panjang sekuens dan satu. Hal tersebut dilakukan untuk membuat pembagian data pada *train* dan *test* masing-masing bernilai genap sehingga dapat dikembalikan menjadi *array* dua dimensi seperti kondisi awal data.

- Skenario 60 % Train

```
# convert an array of values into a dataset matrix
def create_dataset(data, look_back):
    dataX, dataY = [], []
    for i in range(len(data)-look_back):
        dataX.append(data[i:(i+look_back), 0:data.shape[1]])
        dataY.append(data[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

# reshape into X=t and Y=t+1
look_back = 12
X_train60, y_train60 = create_dataset(train60, look_back)
X_test40, y_test40 = create_dataset(test40, look_back)
```

Gambar 19. Coding Pembuatan Time Series LSTM Skenario 3

Pada kode tersebut, panjang sekuens yang ditetapkan ditunjukkan dengan variabel ‘look_back’ dengan besaran 12 periode sebelumnya, menyesuaikan kondisi persebaran data. Tidak ada kondisi khusus pada skenario ini karena jumlah data *test* yang cukup banyak dan bernilai genap setelah dibagi.

c. GRU

Pada metode GRU, pembuatan *time sequence* dilakukan pada praproses data. Hal tersebut dikarenakan langkah ini dilakukan menyesuaikan kondisi pembagian data menjadi data *train* dan *test*.

```
[1211] # Define sequence length and features
sequence_length = 6 # Number of time steps in each sequence
num_features = len(dataset_scaled.columns)

# Create sequences and corresponding labels
sequences = []
labels = []
for i in range(len(dataset_scaled) - sequence_length):
    # Mendapatkan 5 data periode sebelumnya
    seq = dataset_scaled.iloc[i:i+sequence_length]
    # Mendapatkan data periode ke-6 sebagai label, cuma kolom total_visitor
    label = dataset_scaled.iloc[i+sequence_length, 1] # '_temp' column index, kalau prediksi lebih dari satu, ganti angkanya
    sequences.append(seq)
    labels.append(label)
```

Gambar 20. Coding Pembuatan Time Series GRU

Dalam penjelasan di atas, ditentukan panjang urutan (sequence length) dan jumlah fitur dalam dataset. Kemudian, menggunakan loop untuk membuat urutan data yang terdiri dari periode waktu sebelumnya sebanyak panjang urutan. Setiap urutan memiliki 6 langkah waktu. Selanjutnya, kita mendapatkan data pada langkah waktu ke-7 sebagai label, yang pada kasus ini adalah nilai 'total_visitor'. Urutan dan label ini kemudian digunakan untuk melatih model dalam analisis time series.

B. Pembagian dataset

Memisahkan dataset menjadi bagian training dan testing adalah langkah kritis dalam pengembangan model machine learning untuk memastikan model tersebut dapat generalisasi dengan baik pada data baru yang belum pernah dilihat. Rasio pemisahan antara training dan testing mempengaruhi seberapa baik model dapat dinilai dan diprediksi pada data baru. Berikut adalah penjelasan secara naratif untuk rasio pemisahan 80%-20%, 70%-30%, dan 60%-40%:

1. Rasio 80%-20%:

Pemisahan dengan rasio 80%-20% berarti 80% dari data akan digunakan untuk melatih (training) model, sementara 20% sisanya akan digunakan untuk menguji (testing) kinerja model. Rasio ini umum digunakan ketika dataset cukup besar, dan kita ingin memastikan model memiliki volume data yang cukup untuk belajar. Model akan dilatih pada data yang lebih besar dan diuji pada subset yang cukup besar pula untuk memberikan evaluasi yang cukup kredibel terhadap kinerja model.

2. Rasio 70%-30%:

Dengan rasio 70%-30%, sebanyak 70% data akan digunakan untuk melatih model, sedangkan 30% sisanya akan digunakan untuk menguji model. Rasio ini umumnya digunakan ketika dataset ukurannya moderat dan kita masih ingin memiliki volume data yang signifikan untuk melatih model. Dengan menggunakan lebih banyak data untuk training, diharapkan model dapat belajar dengan lebih baik.

3. Rasio 60%-40%:

Dengan rasio 60%-40%, sebanyak 60% data akan digunakan untuk melatih model, sementara 40% sisanya akan digunakan untuk menguji model. Rasio ini mungkin dipilih ketika dataset relatif kecil, dan memastikan model mendapatkan cukup variasi dari data testing untuk mengukur kemampuan generalisasi. Meskipun ukuran training set lebih kecil, kita tetap memiliki data yang cukup untuk melatih model dengan baik.

C. Deskripsi Model Deep Learning (RNN, LSTM, GRU)

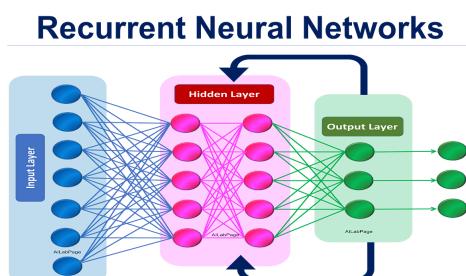
Dalam upaya membangun model rekurensi untuk forecasting jumlah pengunjung ('total_visitor') berdasarkan ketersediaan akomodasi ('total_accommodation'), kami memanfaatkan tiga jenis arsitektur rekurensi yang umum digunakan: RNN (Recurrent Neural Network), LSTM (Long Short-Term Memory), dan GRU (Gated Recurrent Unit). Model-model ini dirancang untuk menangkap pola temporal dan hubungan sekuensial dalam data waktu-seri, seperti data kunjungan wisata dengan ketersediaan akomodasi.

Pertama-tama, kami membangun model RNN, yang merupakan arsitektur dasar untuk mengatasi ketergantungan waktu dalam data. Kemudian, untuk meningkatkan kemampuan memori jangka panjang, kami menerapkan model LSTM. LSTM memiliki keunggulan dalam menangani masalah "vanishing gradient" yang sering dihadapi oleh RNN. Selanjutnya, kami mengevaluasi model GRU yang juga efektif dalam menangkap pola temporal, namun dengan arsitektur yang lebih sederhana dibandingkan LSTM.

Berikut merupakan Langkah-langkah dan penjelasan singkat mengenai metode analisis Dataset ntt dengan 3 model Deep Learning yaitu sebagai berikut :

1. RNN

Recurrent Neural Network (RNN) adalah suatu jenis arsitektur dalam dunia kecerdasan buatan yang dirancang khusus untuk menangani data sekuensial atau time series. Keunikan utama RNN terletak pada kemampuannya untuk mengolah informasi dalam konteks waktu, di mana setiap langkah input tidak hanya mempengaruhi langkah saat ini, tetapi juga dapat memiliki dampak pada langkah-langkah berikutnya.



Gambar 21. Konsep RNN

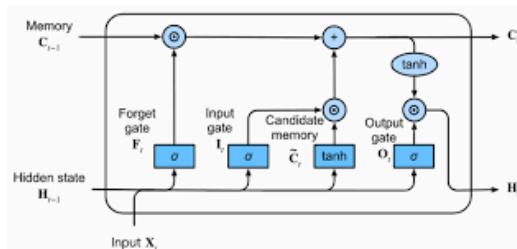
Pada umumnya, manusia tidak membuat keputusan secara isolatif setiap saat. Sebaliknya, kita cenderung mempertimbangkan pengalaman masa lalu dalam proses pengambilan keputusan. Analogi ini mencerminkan prinsip dasar

dari pengembangan Recurrent Neural Network (RNN). Mirip dengan cara manusia berpikir, RNN tidak mengabaikan informasi dari masa lalu saat memproses data. Inilah yang membedakan RNN dari jenis Neural Network lainnya.

Dengan singkatnya, RNN merupakan bagian dari keluarga Neural Network yang dirancang khusus untuk mengolah data yang bersifat berkesinambungan atau berurutan (sequential data). RNN mencapai kemampuan ini dengan menggunakan looping di dalam arsitekturnya, yang secara otomatis memungkinkan penyimpanan informasi dari masa lalu.

2. LSTM

Long Short-Term Memory (LSTM) adalah salah satu metode *deep learning* yang memungkinkan informasi untuk tetap bertahan yang menggunakan model jaringan saraf (*neural network* yang bersifat sekuensial). Metode ini merupakan salah satu jenis dari metode RNN, dengan tambahan bahwa model ini dapat mengatasi permasalahan gradien yang menghilang pada RNN. Ketika metode RNN memiliki kekurangan dalam objek ingatan berjangka panjang, LSTM telah dirancang secara eksplisit untuk mengatasi permasalahan tersebut. Dibandingkan dengan RNN, LSTM dapat menghindari masalah ketergantungan jangka panjang.



Gambar 22. Konsep LSTM

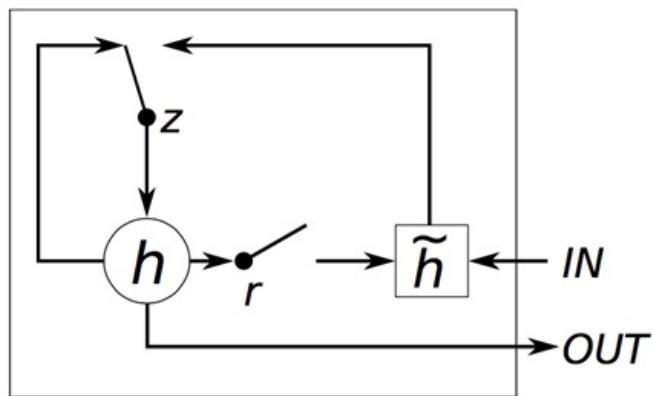
Berbeda dengan metode *neural network* tradisional, LSTM melibatkan koneksi *feedback* yang memungkinkan model tersebut untuk mengolah seluruh baris data. Hal tersebut membuat model ini efektif dalam memahami dan meramalkan pola dalam data berurutan, termasuk data *time series* yang terurut berdasarkan periode waktu. Selain itu, dalam desain arsitekturnya, LSTM memiliki tambahan sel memori dibandingkan dengan metode RNN biasa. Dengan ini, dapat disimpulkan bahwa LSTM merupakan metode *deep learning* yang memodelkan jaringan saraf layaknya RNN, namun dengan adanya tambahan beberapa fitur sehingga dapat mengolah data dengan lebih baik.

3. GRU

GRU (Gated Recurrent Unit) pertama kali diperkenalkan oleh Chung et al. pada tahun 2014 dengan tujuan utama untuk meningkatkan kemampuan setiap unit rekuren dalam menangkap ketergantungan pada skala waktu yang berbeda secara adaptif. Analoginya mirip dengan cara manusia membuat keputusan saat ini; tidak semua informasi dari masa lalu perlu (bahkan terkadang sebaiknya tidak) digunakan. Sebagai contoh, ketika kita memutuskan apa yang

akan dimakan sekarang, informasi mengenai jadwal ujian tengah semester dari masa lalu mungkin tidak memiliki kontribusi besar terhadap keputusan tersebut.

Dalam arsitektur GRU, terdapat dua komponen kunci yang disebut sebagai "gate," yaitu reset gate dan update gate. Ketika kita mengambil keputusan seperti dalam analogi sebelumnya, reset gate pada GRU bertanggung jawab untuk menentukan cara menggabungkan input baru dengan informasi masa lalu, sementara update gate menentukan seberapa banyak informasi masa lalu yang harus dipertahankan. Dengan adanya kedua gate ini, GRU dapat mengatur alur informasi dengan lebih adaptif, memungkinkan model untuk lebih efektif menangkap pola-pola penting dalam data sequential.

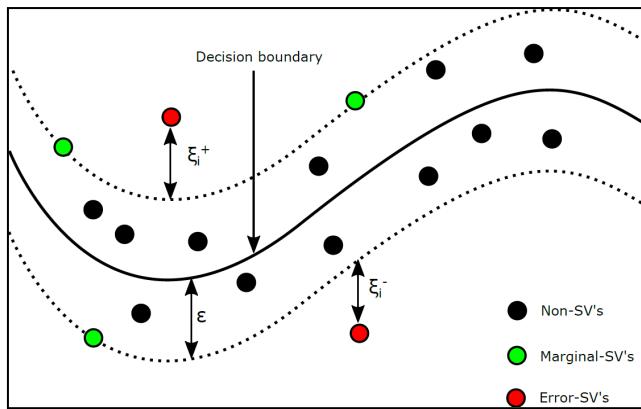


Gambar 23. Konsep GRU

Pada ilustrasi di atas, simbol r mewakili reset gates, sementara z merepresentasikan update gates. Di sisi lain, h dan \tilde{h} merujuk pada aktivasi dan aktivasi kandidat, yang berperan sebagai fungsi aktivasi. Meskipun GRU telah diperkenalkan belum lama, sehingga masih belum banyak dieksplorasi secara mendalam, namun memiliki potensi besar untuk pengembangan lebih lanjut. Terdapat banyak peluang untuk meningkatkan efisiensi dan akurasi GRU sehingga dapat menjadi model RNN yang lebih efektif dan handal.

D. Pemodelan Support Vector Regression (SVR)

Dalam rangka melakukan perbandingan antara model Support Vector Regression (SVR) dan metode deep learning, dalam Final Project ini, akan dibangun model SVR untuk memprediksi jumlah pengunjung berdasarkan fitur-fitur tertentu, seperti ketersediaan akomodasi, cuaca, atau atribut lainnya yang relevan.



Gambar 24. SVR Model

Support Vector membantu dalam mengidentifikasi kesesuaian terdekat antara titik-titik data dengan fungsi yang digunakan untuk menggambarkannya.

E. Evaluasi Model

Menilai performa setiap model dilakukan dengan menggunakan sejumlah metrik evaluasi, termasuk Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), dan Mean Absolute Error (MAE) pada kedua tahap pengujian, yaitu data training dan testing.

Proses pengukuran kinerja ini membantu dalam memahami sejauh mana model mampu memberikan prediksi yang akurat dan seberapa baik model tersebut dapat menggeneralisasi pada data baru yang belum pernah dilihat. Evaluasi dilakukan dengan membandingkan hasil prediksi model dengan nilai sebenarnya, dan metrik-metrik tersebut memberikan gambaran holistik tentang seberapa baik model mampu menyesuaikan diri dengan data latih dan memberikan prediksi yang dapat diandalkan pada data uji.

F. Forecasting 12 Periode Berikutnya

Memanfaatkan model terbaik di antara RNN, LSTM, GRU, dan SVR dengan melakukan prediksi jumlah pengunjung ('total_visitor') untuk periode 12 bulan kedepan. Pilihan model terbaik didasarkan pada hasil evaluasi performa, dan model tersebut akan digunakan untuk menghasilkan prediksi yang optimal terkait dengan trend dan pola temporal data time series. Proses ini akan memberikan gambaran proaktif terhadap perkiraan jumlah pengunjung di masa mendatang, memanfaatkan keunggulan model yang telah terbukti melalui evaluasi kinerja.

G. Analisis Hasil dan Grafik

Setelah memperoleh proyeksi 12 periode mendatang, langkah berikutnya akan melibatkan pembuatan visualisasi grafik yang merepresentasikan hasil prediksi dan evaluasi model untuk setiap skenario dan metode yang telah dianalisis. Grafik ini akan memberikan gambaran yang jelas dan terukur terkait performa model terbaik dalam memprediksi jumlah pengunjung pada periode yang akan datang.

Melalui penggunaan visualisasi ini, kita dapat dengan mudah membandingkan hasil prediksi dengan data aktual serta memahami sejauh mana model mampu

mengikuti pola tren yang sebenarnya. Selain itu, visualisasi grafik juga dapat membantu dalam mengidentifikasi kekuatan dan kelemahan masing-masing metode serta memberikan pandangan yang komprehensif terkait kinerja model secara keseluruhan. Dengan demikian, analisis ini tidak hanya memberikan hasil terbaik dari setiap skenario, tetapi juga memberikan pandangan yang mendalam melalui representasi visual yang efektif.

H. Kesimpulan dan Analisis Hasil

Menarik kesimpulan dan menganalisis hasil dari setiap metode dan skenario yang telah diuji memberikan gambaran komprehensif tentang kinerja model dalam memprediksi jumlah pengunjung. Evaluasi model pada skenario berbeda dan dengan menggunakan metode yang beragam menghasilkan pemahaman yang mendalam tentang kelebihan dan kelemahan masing-masing pendekatan. Dalam analisis ini, dapat disimpulkan bahwa metode rekurensi seperti RNN, LSTM, dan GRU menunjukkan kemampuan yang baik dalam menangkap pola temporal dalam data time series, namun masing-masing memiliki karakteristik dan keunggulan tertentu. Sebagai contoh, LSTM dan GRU, dengan kemampuan memori jangka panjangnya, mungkin lebih efektif dalam menangani ketergantungan waktu yang kompleks.

Pada setiap skenario, hasil prediksi model dievaluasi menggunakan metrik yang relevan seperti Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), dan Mean Absolute Error (MAE). Evaluasi tersebut membantu dalam mengukur sejauh mana model mampu memberikan prediksi yang akurat dan sesuai dengan data aktual. Analisis perbandingan hasil terbaik dari setiap skenario dan metode memberikan wawasan yang berharga untuk memilih pendekatan yang paling efektif dan sesuai dengan karakteristik dataset. Kesimpulan ini menjadi dasar untuk rekomendasi terkait dengan pilihan model terbaik yang dapat digunakan untuk prediksi jumlah pengunjung di masa depan, memberikan kontribusi signifikan dalam perencanaan dan pengembangan sektor pariwisata.

IV. Hasil Eksperimen dan Perbandingan

Dalam tahap Evaluasi Hasil Eksperimen dan Perbandingan ini, akan dievaluasi seberapa efektif model deep learning yang diimplementasikan setelah melalui beberapa tahapan praproses data dan pembagian data training dan testing dalam tiga skenario yang berbeda (80% Train, 70% Train, 60% Train). Evaluasi ini dilakukan pada tiga model berbeda, yaitu RNN, LSTM, dan GRU, serta model SVM (Support Vector Regression). Penilaian kinerja model menggunakan beberapa metrik evaluasi, termasuk Mean Absolute Percentage Error (MAPE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), dan Mean Absolute Error (MAE). Analisis ini akan memberikan gambaran komprehensif mengenai sejauh mana kemampuan prediktif masing-masing model dalam konteks skenario dan pembagian data latih.

1. SVR

- Skenario 80 % Train
 - a. Splitting Train and Test Data

Pertama, kode menentukan panjang dataset (`len(y)`) dan kemudian menghitung jumlah data yang akan digunakan sebagai data pelatihan (`ntt_train80svr`) dengan mengambil 80% dari panjang total dataset. Setelah itu, dilakukan pemisahan antara data pelatihan dan data pengujian menggunakan slicing pada array `X` dan `y` berdasarkan perhitungan sebelumnya. Sebagai contoh, `X_train80svr` dan `y_train80svr` berisi 80% pertama dari data `X` dan `y`, sedangkan `X_test20svr` dan `y_test20svr` berisi 20% sisanya.

```
[343] # Assuming 80-20 train-test split
       ntt_train80svr = int(0.8 * len(y))
       X_train80svr, y_train80svr = X[:ntt_train80svr], y[:ntt_train80svr]
       X_test20svr, y_test20svr = X[ntt_train80svr:], y[ntt_train80svr:]

[344] #Mengubah nilai input agar sesuai
       X_train80svr = X_train80svr.values.reshape(-1, 1)
       X_test20svr = X_test20svr.values.reshape(-1,1)
       y_train80svr = y_train80svr.values.reshape(-1, 1)
       y_test20svr = y_test20svr.values.reshape(-1,1)

[346] # Display results
       lengths = {
           'X_train': len(X_train80svr),
           'y_train': len(y_train80svr),
           'X_test': len(X_test20svr),
           'y_test': len(y_test20svr)
       }

       # Convert lengths to a Pandas DataFrame
       lengths_df = pd.DataFrame.from_dict(lengths, orient='index', columns=['Length'])

       print(lengths_df)

          Length
X_train      86
y_train      86
X_test       22
y_test       22
```

Gambar 25. Splitting Dataset

Pembagian dataset menjadi data pelatihan dan pengujian penting untuk mengukur kinerja model. Data pelatihan digunakan untuk melatih model, sedangkan data pengujian digunakan untuk menguji sejauh mana model mampu melakukan generalisasi pada data yang belum pernah dilihat sebelumnya. dapat disesuaikan tergantung pada karakteristik dataset dan kebutuhan spesifik dari suatu proyek machine learning

b. Hyperparameter Tuning

Auto tuning merupakan implementasi dari proses pencarian parameter terbaik (hyperparameter tuning) menggunakan GridSearchCV pada model Support Vector Regression (SVR) dengan kernel linear atau rbf. Proses ini dilakukan untuk mencari kombinasi parameter yang memberikan performa terbaik dalam hal prediksi. Parameter-parameter yang diuji mencakup parameter C, epsilon, dan jenis kernel (linear atau rbf).

```
[587] # Define the parameter grid for GridSearchCV
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'epsilon': [0.01, 0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf']
}

# Create GridSearchCV instance
grid_search = GridSearchCV(
    estimator=SVR(),
    param_grid=param_grid,
    refit=True,
    verbose=3,
    cv=3,
    error_score='raise'
)

# Fit the model with the best parameters
grid_search.fit(X_train80svr, y_train80svr)

# Menampilkan parameter terbaik
print("Parameter terbaik:", grid_search.best_params_)
```

Gambar 26. Coding Hyperparameter Tuning SVR

Pertama, didefinisikan param_grid yang berisi daftar nilai yang akan diuji untuk setiap parameter. Misalnya, C diuji pada nilai [0.01, 0.1, 1, 10, 100], epsilon pada nilai [0.01, 0.1, 1, 10, 100], dan kernel pada jenis ['linear', 'rbf']. Selanjutnya, kita membuat instance dari GridSearchCV dengan menggunakan model SVR, parameter grid yang telah didefinisikan sebelumnya, dan konfigurasi lainnya seperti cv (jumlah lipatan cross-validation) dan verbose untuk menampilkan informasi selama proses pencarian.

Proses fitting dilakukan dengan memanggil metode fit() pada instance GridSearchCV. Setelah pencarian selesai, kita mendapatkan parameter terbaik dengan memanggil atribut best_params_ dari objek GridSearchCV. Dengan informasi parameter terbaik ini, kita dapat mengoptimalkan model SVR untuk data kita. GridSearchCV secara otomatis akan melatih ulang model dengan parameter terbaik pada seluruh data pelatihan setelah proses pencarian selesai (refit=True).

c. Create Model

Pembuatan Model akan menggunakan grid search untuk menemukan hiperparameter optimal untuk model SVR, diikuti oleh pembuatan dan pelatihan model SVR baru menggunakan parameter terbaik pada dataset latih tertentu. Variabel best_param mengandung kombinasi parameter terbaik yang ditemukan selama proses grid search. Grid search adalah teknik penalaan hiperparameter di mana sejumlah nilai hiperparameter yang telah ditentukan sebelumnya diuji secara menyeluruh untuk menemukan kombinasi yang memberikan kinerja model terbaik. Pada kasus ini, nampaknya telah diterapkan pada model SVR.

```
[588] best_param = grid_search.best_params_
# Create SVR model
regressor = SVR(**best_param)

# Fit the SVR model
regressor.fit(X_train80svr, y_train80svr)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y
y = column_or_1d(y, warn=True)
SVR(C=0.01, epsilon=100, kernel='linear')
```

Gambar 27. Coding Create Model SVR

Selanjutnya, model SVR baru (regressor) dibuat dengan menggunakan parameter terbaik yang ditemukan dari grid search. Model SVR difit (disesuaikan) dengan data latih ($X_{train80svr}$ dan $y_{train80svr}$). Langkah ini melibatkan pelatihan model pada fitur input ($X_{train80svr}$) dan nilai target yang sesuai ($y_{train80svr}$), memungkinkan algoritma SVR untuk mempelajari pola-pola mendasar dalam data.

d. Prediction

Prediksi dilakukan menggunakan model Support Vector Regression (SVR) yang telah di-fit sebelumnya dengan data latih (training data). Dengan menggunakan `regressor.predict($X_{test20svr}$)`, model SVR memprediksi nilai target untuk data uji ($X_{test20svr}$). Kemudian, nilai prediksi yang diperoleh dibulatkan ke nilai integer menggunakan `np.round` dan diubah menjadi tipe data integer dengan `astype(int)`. Hasil prediksi ini dapat digunakan untuk evaluasi performa model atau keperluan analisis lebih lanjut.

```
[589] # Prediksi dengan data test
y_pred80svr = np.round(regressor.predict(X_test20svr)).astype(int)

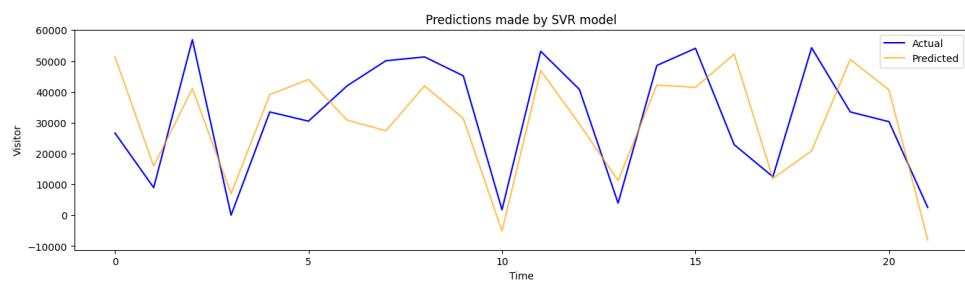
[590] y_pred80svr
array([51384, 15918, 41030, 7006, 39085, 44032, 30846, 27384, 41927,
       31391, -5101, 46896, 29553, 11259, 42195, 41446, 52229, 11911,
      20887, 50487, 40624, -7986])

[591] def plotting_actual_vs_pred(y_test20svr, y_pred, title):
    plt.figure(figsize=(16, 4))
    plt.plot(y_test20svr, color='blue', label='Actual')
    plt.plot(y_pred, alpha=0.7, color='orange',
             label='Predicted')
    plt.title(title)
    plt.xlabel('Time')
    plt.ylabel('Visitor')
    plt.legend()
    plt.show()

[592] plotting_actual_vs_pred(y_test20svr, y_pred80svr, "Predictions made by SVR model")
```

Gambar 28. Coding Prediction SVR

Plot berikut akan menampilkan perbandingan antara data aktual dan prediksi berdasarkan model SVR dengan pembagian data training dan test yang telah dilakukan.



Gambar 29. Plot Hasil Prediksi SVR

e. Evaluate Model (Train)

Prediksi dilakukan menggunakan data latih (train data), dan hasil prediksi tersebut dibandingkan dengan nilai sebenarnya dari target variabel pada data latih. Metrik evaluasi yang umum digunakan antara lain Mean Squared

Error (MSE), Mean Absolute Error (MAE), dan Mean Absolute Percentage Error (MAPE).

```
[594] # Prediksi dengan data yang sudah dinormalisasi
y_pred80svr2 = np.round(regressor.predict(X_train80svr)).astype(int)

[595] y_pred80svr2

array([-6726, -4172, 48766, 49589, 712, 6119, 30098, 19669, 40741,
       45507, -7206, 40004, 220, 24467, -3092, 7487, 28859, 43648,
       21966, 47196, -1586, 15298, 4602, 48211, 7786, 34137, 14924,
       42846, -3295, 49846, -36, 2614, 23633, 1321, 1652, 3383,
       20694, 35943, 20128, 44684, 35772, 26561, 46437, 16869, 20320,
       34885, 25717, 43242, 38946, 33710, 32855, 10009, -2879, 23302,
       13364, 5435, 12541, 10521, 26390, 2935, 9122, 8074, -7089,
       11558, 4003, 42376, 17222, -6020, 22981, 37888, -2441, 25044,
       -506, 32086, 25610, 34372, 28378, 48446, 18632, -9301, 36841,
       13855, 23868, 51865, 19872, 18087])
```

Gambar 30. Hasil Prediksi SVR

```
[600] # Evaluate the model
mse_80_train = mean_squared_error(y_train80svr, y_pred80svr2)
mae_80_train = mean_absolute_error(y_train80svr, y_pred80svr2)
mape_80_train = mean_absolute_percentage_error(y_train80svr, y_pred80svr2)

print(f"Mean Absolute Error (MAE): {round(mae_80_train, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_80_train, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_80_train, 2)}%")

Mean Absolute Error (MAE): 9733.55
Mean Squared Error (MSE): 170505689.01
Mean Absolute Percentage Error (MAPE): 1.5978876212785574e+18%
```

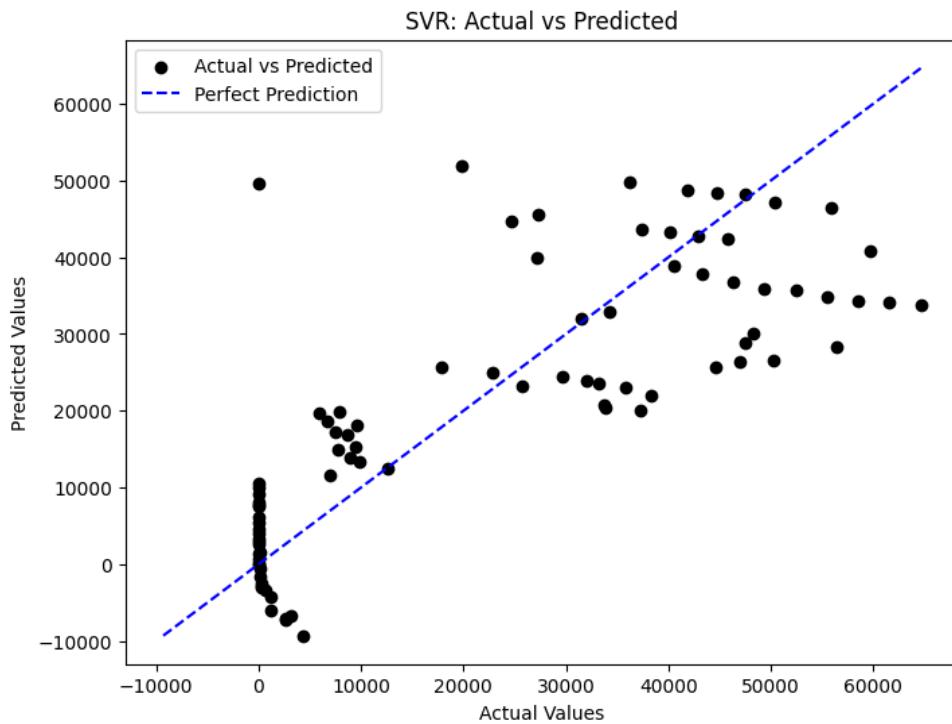
Gambar 31. Evaluasi Model SVR

Scatter plot membantu untuk memvisualisasikan perbandingan antara nilai aktual dan nilai prediksi dari model Support Vector Regression (SVR). Setiap titik dalam scatter plot merepresentasikan pasangan nilai aktual dan nilai prediksi untuk setiap sampel pada data latih.

```
❶ # Scatter plot to visualize predicted vs actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_train80svr, y_pred80svr2, color='black', label='Actual vs Predicted')
# Plotting the diagonal line (perfect prediction)
max_value = max(max(y_train80svr), max(y_pred80svr2))
min_value = min(min(y_train80svr), min(y_pred80svr2))
plt.plot([min_value, max_value], [min_value, max_value], color='blue', linestyle='--', label='Perfect Prediction')

plt.title('SVR: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
```

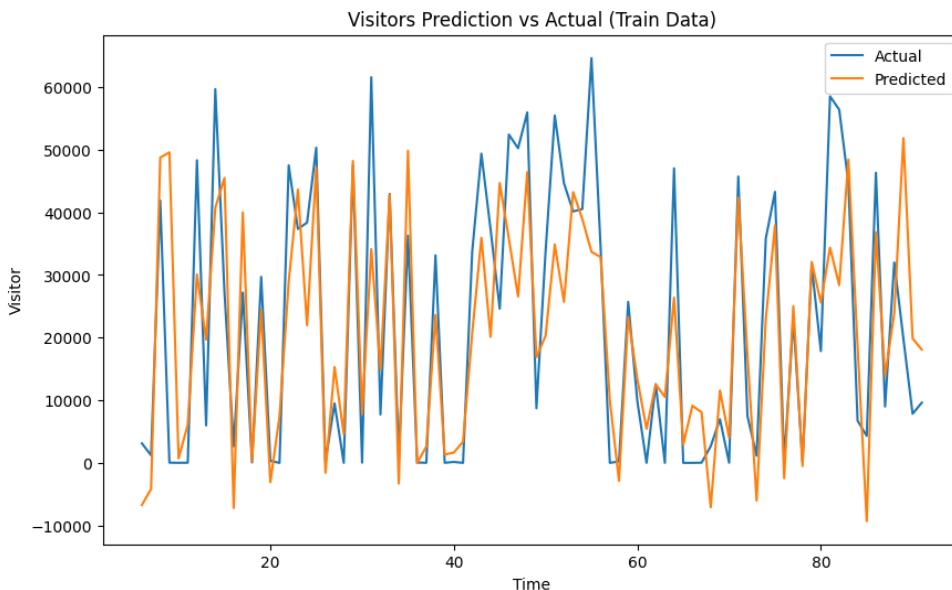
Gambar 32. Coding Plot Perbandingan SVR



Gambar 33. Gambar Hasil Perbandingan SVR

Dengan adanya garis diagonal biru dengan pola putus-putus, plot ini memberikan gambaran visual tentang seberapa baik prediksi model SVR dibandingkan dengan nilai aktual. Jika semua titik sejajar dengan garis diagonal, itu menunjukkan model memiliki performa yang sempurna. Scatter plot ini membantu secara intuitif memahami sejauh mana model SVR mampu mereplikasi nilai sebenarnya pada data latih.

Berikut adalah grafik yang menunjukkan perbandingan antara data aktual dan prediksi dari data latih



Gambar 34. Grafik Perbandingan SVR

f. Evaluate Model (Test)

Prediksi dilakukan menggunakan data uji (test data), dan hasil prediksi tersebut dibandingkan dengan nilai sebenarnya dari target variabel pada data latih. Metrik evaluasi yang umum digunakan antara lain Mean Squared Error (MSE), Mean Absolute Error (MAE), dan Mean Absolute Percentage Error (MAPE).

```
[589] # Prediksi dengan data test
y_pred80svr = np.round(regressor.predict(X_test20svr)).astype(int)

[590] y_pred80svr
array([51384, 15918, 41030, 7006, 39085, 44032, 30846, 27384, 41927,
       31391, -5101, 46896, 29553, 11259, 42195, 41446, 52229, 11911,
      20887, 50487, 40624, -7986])
```

Gambar 35. Hasil Prediksi SVR

```
[843] # Evaluate the model
mse_80_test = mean_squared_error(y_test20svr, y_pred80svr)
mae_80_test = mean_absolute_error(y_test20svr, y_pred80svr)
mape_80_test = mean_absolute_percentage_error(y_test20svr, y_pred80svr)

print(f"Mean Absolute Error (MAE): {round(mae_80_test, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_80_test, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_80_test, 2)}%")

Mean Absolute Error (MAE): 12842.91
Mean Squared Error (MSE): 229233230.91
Mean Absolute Percentage Error (MAPE): 12.11%
```

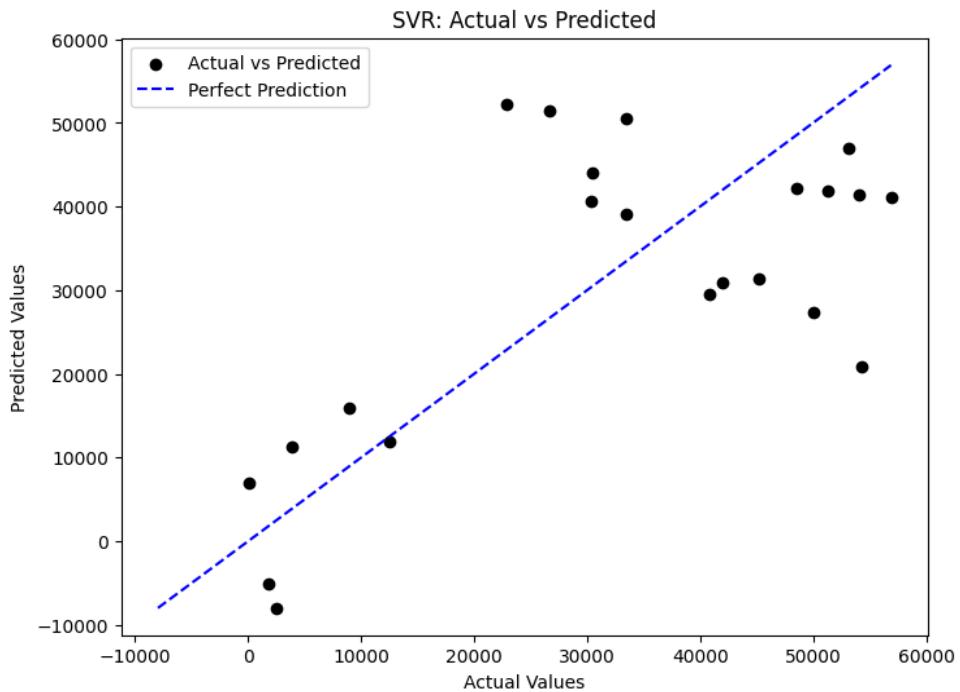
Gambar 36. Evaluasi Model Prediksi SVR

Scatter plot membantu untuk memvisualisasikan perbandingan antara nilai aktual dan nilai prediksi dari model Support Vector Regression (SVR). Setiap titik dalam scatter plot merepresentasikan pasangan nilai aktual dan nilai prediksi untuk setiap sampel pada data latih.

```
[844] # Scatter plot to visualize predicted vs actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test20svr, y_pred80svr, color='black', label='Actual vs Predicted')
# Plotting the diagonal line (perfect prediction)
max_value = max(max(y_test20svr), max(y_pred80svr))
min_value = min(min(y_test20svr), min(y_pred80svr))
plt.plot([min_value, max_value], [min_value, max_value], color='blue', linestyle='--', label='Perfect Prediction')

plt.title('SVR: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
```

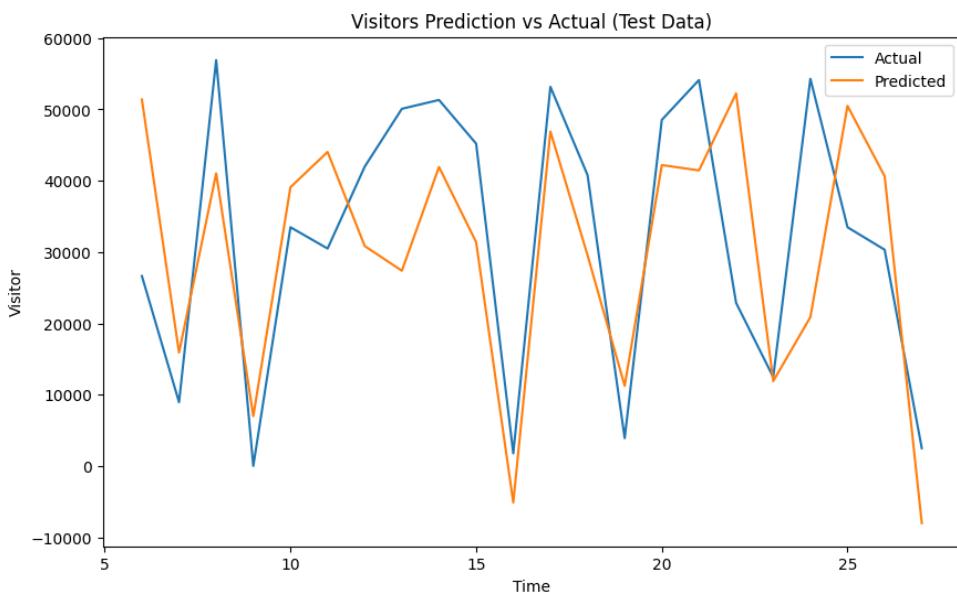
Gambar 37. Coding Plot Perbandingan Prediksi SVR



Gambar 38. Hasil Plot Perbandingan Prediksi SVR

Dengan adanya garis diagonal biru dengan pola putus-putus, plot ini memberikan gambaran visual tentang seberapa baik prediksi model SVR dibandingkan dengan nilai aktual. Jika semua titik sejajar dengan garis diagonal, itu menunjukkan model memiliki performa yang sempurna. Scatter plot ini membantu secara intuitif memahami sejauh mana model SVR mampu mereplikasi nilai sebenarnya pada data latih.

Berikut adalah grafik yang menunjukkan perbandingan antara data aktual dan prediksi dari data uji



Gambar 39. Grafik Perbandingan Prediksi SVR

- Skenario 70 % Train
- a. Splitting Train and Test Data

Pertama, kode menentukan panjang dataset (`len(y)`) dan kemudian menghitung jumlah data yang akan digunakan sebagai data pelatihan (`ntt_train70svr`) dengan mengambil 70% dari panjang total dataset. Setelah itu, dilakukan pemisahan antara data pelatihan dan data pengujian menggunakan slicing pada array `X` dan `y` berdasarkan perhitungan sebelumnya. Sebagai contoh, `X_train70svr` dan `y_train70svr` berisi 70% pertama dari data `X` dan `y`, sedangkan `X_test30svr` dan `y_test30svr` berisi 30% sisanya.

```
[848] # Assuming 80-20 train-test split
      ntt_train70svr = int(0.7 * len(y))
      X_train70svr, y_train70svr = X[:ntt_train70svr], y[:ntt_train70svr]
      X_test30svr, y_test30svr = X[ntt_train70svr:], y[ntt_train70svr:]

[849] #Mengubah nilai input agar sesuai
      X_train70svr = X_train70svr.values.reshape(-1, 1)
      X_test30svr = X_test30svr.values.reshape(-1,1)

[850] # Display results
      lengths = {
          'X_train': len(X_train70svr),
          'y_train': len(y_train70svr),
          'X_test': len(X_test30svr),
          'y_test': len(y_test30svr)
      }

      # Convert lengths to a Pandas DataFrame
      lengths_df = pd.DataFrame.from_dict(lengths, orient='index', columns=['Length'])

      print(lengths_df)

      Length
      X_train    75
      y_train    75
      X_test     33
      y_test     33
```

Gambar 40. Splitting Dataset

Pembagian dataset menjadi data pelatihan dan pengujian penting untuk mengukur kinerja model. Data pelatihan digunakan untuk melatih model, sedangkan data pengujian digunakan untuk menguji sejauh mana model mampu melakukan generalisasi pada data yang belum pernah dilihat sebelumnya. dapat disesuaikan tergantung pada karakteristik dataset dan kebutuhan spesifik dari suatu proyek machine learning

b. Hyperparameter Tuning

Auto tuning merupakan implementasi dari proses pencarian parameter terbaik (hyperparameter tuning) menggunakan GridSearchCV pada model Support Vector Regression (SVR) dengan kernel linear atau rbf. Proses ini dilakukan untuk mencari kombinasi parameter yang memberikan performa terbaik dalam hal prediksi. Parameter-parameter yang diuji mencakup parameter C, epsilon, dan jenis kernel (linear atau rbf).

```
[851] # Define the parameter grid for GridSearchCV
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'epsilon': [0.01, 0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf']
}

# Create GridSearchCV instance
grid_search = GridSearchCV(
    estimator=SVR(),
    param_grid=param_grid,
    refit=True,
    verbose=3,
    cv=3,
    error_score='raise'
)

# Fit the model with the best parameters
grid_search.fit(X_train70svr, y_train70svr)

# Menampilkan parameter terbaik
print("Parameter terbaik:", grid_search.best_params_)
```

Gambar 41. Coding Hyperparameter Tuning SVR

Pertama, didefinisikan param_grid yang berisi daftar nilai yang akan diuji untuk setiap parameter. Misalnya, C diuji pada nilai [0.01, 0.1, 1, 10, 100], epsilon pada nilai [0.01, 0.1, 1, 10, 100], dan kernel pada jenis ['linear', 'rbf']. Selanjutnya, kita membuat instance dari GridSearchCV dengan menggunakan model SVR, parameter grid yang telah didefinisikan sebelumnya, dan konfigurasi lainnya seperti cv (jumlah lipatan cross-validation) dan verbose untuk menampilkan informasi selama proses pencarian.

Proses fitting dilakukan dengan memanggil metode fit() pada instance GridSearchCV. Setelah pencarian selesai, kita mendapatkan parameter terbaik dengan memanggil atribut best_params_ dari objek GridSearchCV. Dengan informasi parameter terbaik ini, kita dapat mengoptimalkan model SVR untuk data kita. GridSearchCV secara otomatis akan melatih ulang model dengan parameter terbaik pada seluruh data pelatihan setelah proses pencarian selesai (refit=True).

c. Create Model

Pembuatan Model akan menggunakan grid search untuk menemukan hiperparameter optimal untuk model SVR, diikuti oleh pembuatan dan pelatihan model SVR baru menggunakan parameter terbaik pada dataset latih tertentu. Variabel best_param mengandung kombinasi parameter terbaik yang ditemukan selama proses grid search. Grid search adalah teknik penalaan hiperparameter di mana sejumlah nilai hiperparameter yang telah ditentukan sebelumnya diuji secara menyeluruh untuk menemukan kombinasi yang memberikan kinerja model terbaik. Pada kasus ini, nampaknya telah diterapkan pada model SVR.

```
[852] best_param = grid_search.best_params_
# Create SVR model
regressor = SVR(**best_param)

# Fit the SVR model
regressor.fit(X_train70svr, y_train70svr)

SVR(C=0.1, epsilon=100, kernel='linear')
```

Gambar 42. Create Model SVR

Selanjutnya, model SVR baru (regressor) dibuat dengan menggunakan parameter terbaik yang ditemukan dari grid search. Model SVR difit (diseduaikan) dengan data latih ($X_{train70svr}$ dan $y_{train70svr}$). Langkah ini melibatkan pelatihan model pada fitur input ($X_{train70svr}$) dan nilai target yang sesuai ($y_{train70svr}$), memungkinkan algoritma SVR untuk mempelajari pola-pola mendasar dalam data.

d. Prediction

Prediksi dilakukan menggunakan model Support Vector Regression (SVR) yang telah di-fit sebelumnya dengan data latih (training data). Dengan menggunakan `regressor.predict(X_test30svr)`, model SVR memprediksi nilai target untuk data uji ($X_{test30svr}$). Kemudian, nilai prediksi yang diperoleh dibulatkan ke nilai integer menggunakan `np.round` dan diubah menjadi tipe data integer dengan `astype(int)`. Hasil prediksi ini dapat digunakan untuk evaluasi performa model atau keperluan analisis lebih lanjut.

```
[853] # Prediksi dengan data yang sudah dinormalisasi
y_pred70svr = np.round(regressor.predict(X_test30svr)).astype(int)

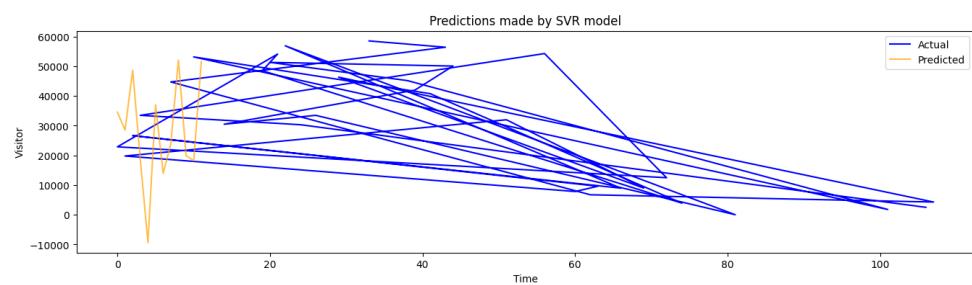
[854] y_pred70svr
array([34533, 28510, 48673, 18718, -9348, 37013, 13919, 23979, 52109,
       19963, 18170, 51626, 15991, 41222, 7036, 39268, 44239, 30890,
       27511, 42124, 31537, -5128, 47116, 29691, 11310, 42392, 41641,
       52474, 11964, 20983, 50724, 40814, -8627])

[898] def plotting_actual_vs_pred(y_test30svr, y_pred, title):
    plt.figure(figsize=(16, 4))
    plt.plot(y_test30svr, color='blue', label='Actual')
    plt.plot(y_pred, alpha=0.7, color='orange', label='Predicted')
    plt.title(title)
    plt.xlabel('Time')
    plt.ylabel('Visitor')
    plt.legend()
    plt.show()

[896] plotting_actual_vs_pred(y_test30svr, y_pred70svr, "Predictions made by SVR model")
```

Gambar 43. Coding Perbandingan Train SVR

Plot berikut akan menampilkan perbandingan antara data aktual dan prediksi berdasarkan model SVR dengan pembagian data training dan test yang telah dilakukan.



Gambar 44. Grafik Perbandingan SVR

e. Evaluate Model (Train)

Prediksi dilakukan menggunakan data latih (train data), dan hasil prediksi tersebut dibandingkan dengan nilai sebenarnya dari target variabel pada data latih. Metrik evaluasi yang umum digunakan antara lain Mean Squared

Error (MSE), Mean Absolute Error (MAE), dan Mean Absolute Percentage Error (MAPE).

```
[857] # Prediksi dengan data yang sudah dinormalisasi
y_pred70svr2 = np.round(regressor.predict(X_train70svr)).astype(int)

❶ y_pred70svr2

❷ array([-6760, -4194, 48995, 49822, 712, 6145, 30238, 19759, 40932,
       45721, -7243, 40191, 219, 24580, -3110, 7520, 28993, 43852,
       22068, 47417, -1596, 15368, 4621, 48437, 7820, 34297, 14992,
       43047, -3314, 50080, -39, 2624, 23743, 1324, 1657, 3397,
       20790, 36111, 20221, 44894, 35939, 26684, 46655, 16946, 20414,
       35048, 25836, 43444, 39128, 33867, 33008, 10053, -2895, 23410,
       13425, 5458, 12598, 10569, 26513, 2946, 9162, 8110, -7125,
       11610, 4019, 42575, 17301, -6052, 23088, 38065, -2455, 25160,
       -512, 32235, 25729])
```

Gambar 45. Hasil Prediksi SVR

```
[859] # Evaluate the model
mse_70_train = mean_squared_error(y_train70svr, y_pred70svr2)
mae_70_train = mean_absolute_error(y_train70svr, y_pred70svr2)
mape_70_train = mean_absolute_percentage_error(y_train70svr, y_pred70svr2)

print(f"Mean Absolute Error (MAE): {round(mae_70_train, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_70_train, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_70_train, 2)}%")

Mean Absolute Error (MAE): 9072.44
Mean Squared Error (MSE): 153466296.41
Mean Absolute Percentage Error (MAPE): 1.8403509517286794e+18%
```

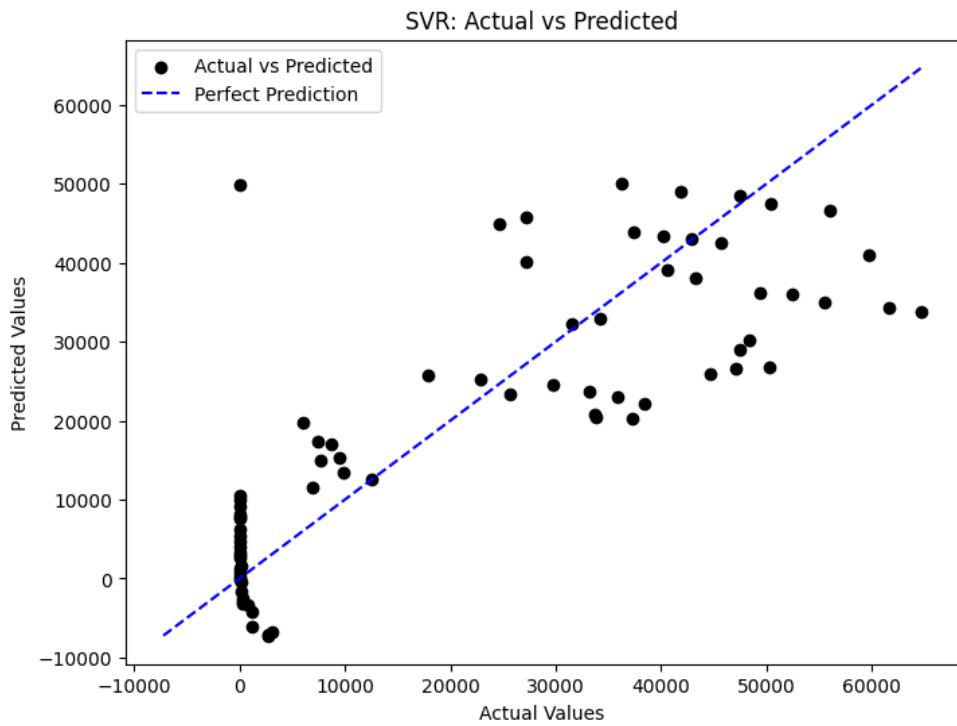
Gambar 46. Evaluasi Prediksi SVR

Scatter plot membantu untuk memvisualisasikan perbandingan antara nilai aktual dan nilai prediksi dari model Support Vector Regression (SVR). Setiap titik dalam scatter plot merepresentasikan pasangan nilai aktual dan nilai prediksi untuk setiap sampel pada data latih.

```
[860] # Scatter plot to visualize predicted vs actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_train70svr, y_pred70svr2, color='black', label='Actual vs Predicted')
# Plotting the diagonal line (perfect prediction)
max_value = max(max(y_train70svr), max(y_pred70svr2))
min_value = min(min(y_train70svr), min(y_pred70svr2))
plt.plot([min_value, max_value], [min_value, max_value], color='blue', linestyle='--', label='Perfect Prediction')

plt.title('SVR: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
```

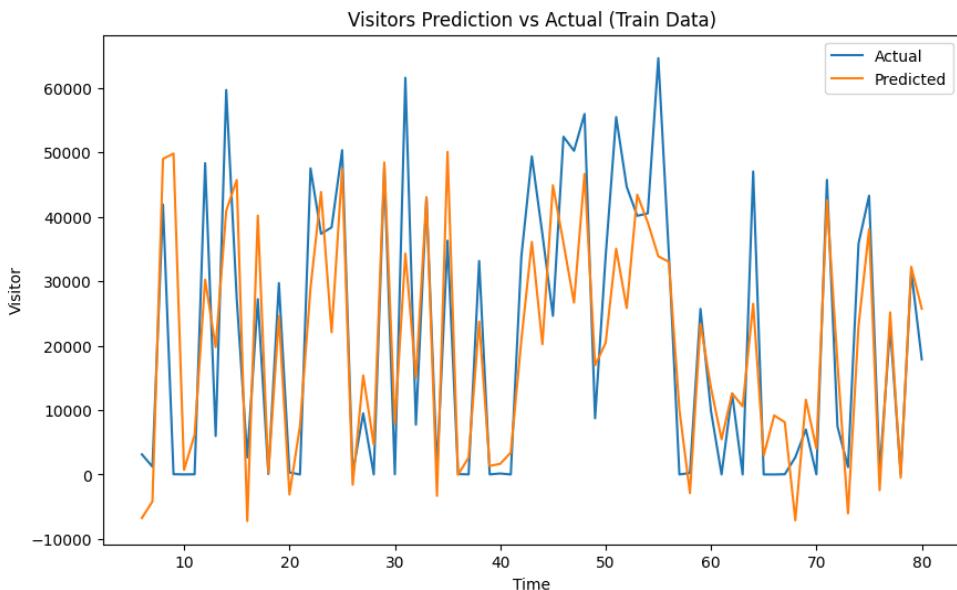
Gambar 47. Coding Plot Perbandingan SVR



Gambar 48. Plot Perbandingan Train SVR

Dengan adanya garis diagonal biru dengan pola putus-putus, plot ini memberikan gambaran visual tentang seberapa baik prediksi model SVR dibandingkan dengan nilai aktual. Jika semua titik sejajar dengan garis diagonal, itu menunjukkan model memiliki performa yang sempurna. Scatter plot ini membantu secara intuitif memahami sejauh mana model SVR mampu mereplikasi nilai sebenarnya pada data latih.

Berikut adalah grafik yang menunjukkan perbandingan antara data aktual dan prediksi dari data latih



Gambar 49. Grafik Perbandingan Train SVR

f. Evaluate Model (Test)

Prediksi dilakukan menggunakan data uji (test data), dan hasil prediksi tersebut dibandingkan dengan nilai sebenarnya dari target variabel pada data latih. Metrik evaluasi yang umum digunakan antara lain Mean Squared Error (MSE), Mean Absolute Error (MAE), dan Mean Absolute Percentage Error (MAPE).

```
[853] # Prediksi dengan data yang sudah dinormalisasi
y_pred70svr = np.round(regressor.predict(X_test30svr)).astype(int)

[854] y_pred70svr
array([34533, 28510, 48673, 18718, -9348, 37013, 13919, 23979, 52109,
       19963, 18170, 51626, 15991, 41222, 7036, 39268, 44239, 30990,
       27511, 42124, 31537, -5128, 47116, 29691, 11310, 42392, 41641,
      52474, 11964, 20983, 50724, 40814, -8027])
```

Gambar 50. Hasil Prediksi SVR

```
[862] # Evaluate the model
mse_70_test = mean_squared_error(y_test30svr, y_pred70svr)
mae_70_test = mean_absolute_error(y_test30svr, y_pred70svr)
mape_70_test = mean_absolute_percentage_error(y_test30svr, y_pred70svr)

print(f"Mean Absolute Error (MAE): {round(mae_70_test, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_70_test, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_70_test, 2)}%")

Mean Absolute Error (MAE): 13306.39
Mean Squared Error (MSE): 248207120.45
Mean Absolute Percentage Error (MAPE): 8.44%
```

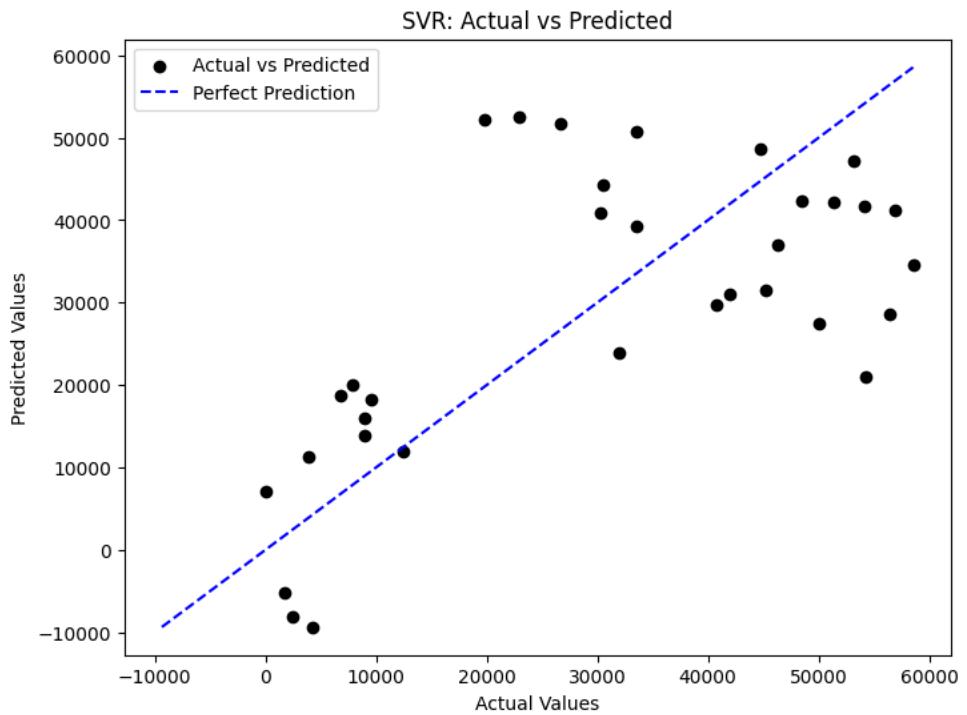
Gambar 51. Evaluasi Prediksi SVR

Scatter plot membantu untuk memvisualisasikan perbandingan antara nilai aktual dan nilai prediksi dari model Support Vector Regression (SVR). Setiap titik dalam scatter plot merepresentasikan pasangan nilai aktual dan nilai prediksi untuk setiap sampel pada data latih.

```
[863] # Scatter plot to visualize predicted vs actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test30svr, y_pred70svr, color='black', label='Actual vs Predicted')
# Plotting the diagonal line (perfect prediction)
max_value = max(max(y_test30svr), max(y_pred70svr))
min_value = min(min(y_test30svr), min(y_pred70svr))
plt.plot([min_value, max_value], [min_value, max_value], color='blue', linestyle='--', label='Perfect Prediction')

plt.title('SVR: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
```

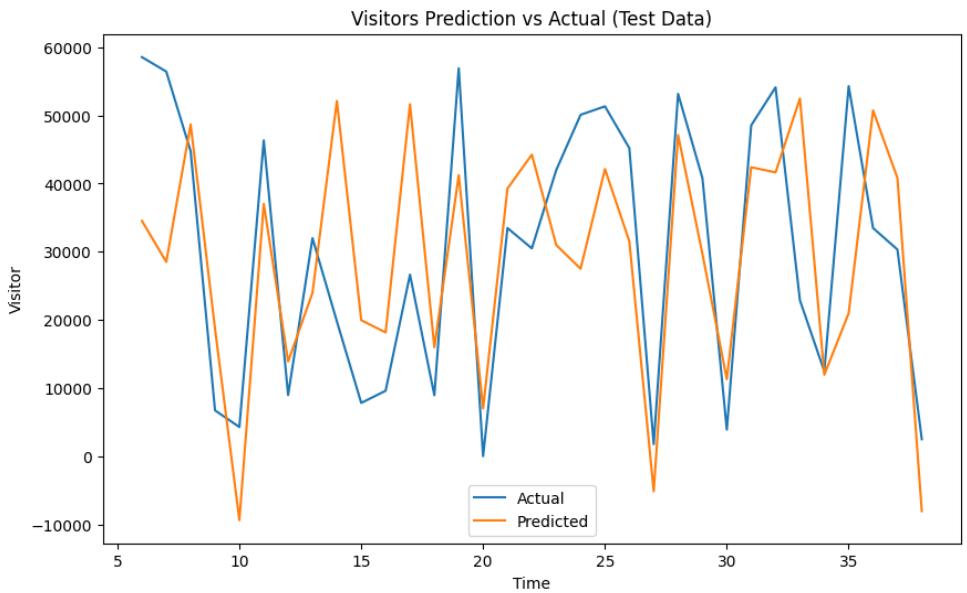
Gambar 52. Coding Plot Perbandingan Prediksi SVR



Gambar 53. Plot Perbandingan Prediksi SVR

Dengan adanya garis diagonal biru dengan pola putus-putus, plot ini memberikan gambaran visual tentang seberapa baik prediksi model SVR dibandingkan dengan nilai aktual. Jika semua titik sejajar dengan garis diagonal, itu menunjukkan model memiliki performa yang sempurna. Scatter plot ini membantu secara intuitif memahami sejauh mana model SVR mampu mereplikasi nilai sebenarnya pada data latih.

Berikut adalah grafik yang menunjukkan perbandingan antara data aktual dan prediksi dari data uji



Gambar 54. Hasil Grafik Perbandingan Prediksi SVR

- Skenario 60 % Train
- a. Splitting Train and Test Data

Pertama, kode menentukan panjang dataset (`len(y)`) dan kemudian menghitung jumlah data yang akan digunakan sebagai data pelatihan (`ntt_train60svr`) dengan mengambil 60% dari panjang total dataset. Setelah itu, dilakukan pemisahan antara data pelatihan dan data pengujian menggunakan slicing pada array `X` dan `y` berdasarkan perhitungan sebelumnya. Sebagai contoh, `X_train70svr` dan `y_train60svr` berisi 60% pertama dari data `X` dan `y`, sedangkan `X_test40svr` dan `y_test40svr` berisi 40% sisanya.

```
[959] # Assuming 80-20 train-test split
      ntt_train60svr = int(0.6 * len(y))
      X_train60svr, y_train60svr = X[:ntt_train60svr], y[:ntt_train60svr]
      X_test40svr, y_test40svr = X[ntt_train60svr:], y[ntt_train60svr:]

[960] # Mengubah nilai input agar sesuai
      X_train60svr = X_train60svr.values.reshape(-1, 1)
      X_test40svr = X_test40svr.values.reshape(-1,1)

[962] # Display results
      lengths = {
          'X_train': len(X_train60svr),
          'y_train': len(y_train60svr),
          'X_test': len(X_test40svr),
          'y_test': len(y_test40svr)
      }

      # Convert lengths to a Pandas DataFrame
      lengths_df = pd.DataFrame.from_dict(lengths, orient='index', columns=['Length'])

      print(lengths_df)
      Length
      X_train    64
      y_train    64
      X_test     44
      y_test     44
```

Gambar 55. Splitting Dataset

Pembagian dataset menjadi data pelatihan dan pengujian penting untuk mengukur kinerja model. Data pelatihan digunakan untuk melatih model, sedangkan data pengujian digunakan untuk menguji sejauh mana model mampu melakukan generalisasi pada data yang belum pernah dilihat sebelumnya. dapat disesuaikan tergantung pada karakteristik dataset dan kebutuhan spesifik dari suatu proyek machine learning

b. Hyperparameter Tuning

Auto tuning merupakan implementasi dari proses pencarian parameter terbaik (hyperparameter tuning) menggunakan GridSearchCV pada model Support Vector Regression (SVR) dengan kernel linear atau rbf. Proses ini dilakukan untuk mencari kombinasi parameter yang memberikan performa terbaik dalam hal prediksi. Parameter-parameter yang diuji mencakup parameter C, epsilon, dan jenis kernel (linear atau rbf).

```
[963] # Define the parameter grid for GridSearchCV
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'epsilon': [0.01, 0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf']
}

# Create GridSearchCV instance
grid_search = GridSearchCV(
    estimator=SVR(),
    param_grid=param_grid,
    refit=True,
    verbose=3,
    cv=3,
    error_score='raise'
)

# Fit the model with the best parameters
grid_search.fit(X_train60svr, y_train60svr)

# Menampilkan parameter terbaik
print("Parameter terbaik:", grid_search.best_params_)
```

Gambar 56. Coding Hyperparameter Tuning SVR

Pertama, didefinisikan param_grid yang berisi daftar nilai yang akan diuji untuk setiap parameter. Misalnya, C diuji pada nilai [0.01, 0.1, 1, 10, 100], epsilon pada nilai [0.01, 0.1, 1, 10, 100], dan kernel pada jenis ['linear', 'rbf']. Selanjutnya, kita membuat instance dari GridSearchCV dengan menggunakan model SVR, parameter grid yang telah didefinisikan sebelumnya, dan konfigurasi lainnya seperti cv (jumlah lipatan cross-validation) dan verbose untuk menampilkan informasi selama proses pencarian.

Proses fitting dilakukan dengan memanggil metode fit() pada instance GridSearchCV. Setelah pencarian selesai, kita mendapatkan parameter terbaik dengan memanggil atribut best_params_ dari objek GridSearchCV. Dengan informasi parameter terbaik ini, kita dapat mengoptimalkan model SVR untuk data kita. GridSearchCV secara otomatis akan melatih ulang model dengan parameter terbaik pada seluruh data pelatihan setelah proses pencarian selesai (refit=True).

c. Create Model

Pembuatan Model akan menggunakan grid search untuk menemukan hiperparameter optimal untuk model SVR, diikuti oleh pembuatan dan pelatihan model SVR baru menggunakan parameter terbaik pada dataset latih tertentu. Variabel best_param mengandung kombinasi parameter terbaik yang ditemukan selama proses grid search. Grid search adalah teknik penalaan hiperparameter di mana sejumlah nilai hiperparameter yang telah ditentukan sebelumnya diuji secara menyeluruh untuk menemukan kombinasi yang memberikan kinerja model terbaik. Pada kasus ini, nampaknya telah diterapkan pada model SVR.

```
[964] best_param = grid_search.best_params_
# Create SVR model
regressor = SVR(**best_param)

# Fit the SVR model
regressor.fit(X_train60svr, y_train60svr)

▼
      SVR
SVR(C=0.01, epsilon=0.01, kernel='linear')
```

Gambar 57. Coding Create Model SVR

Selanjutnya, model SVR baru (regressor) dibuat dengan menggunakan parameter terbaik yang ditemukan dari grid search. Model SVR difit (diseduaikan) dengan data latih ($X_{train60svr}$ dan $y_{train60svr}$). Langkah ini melibatkan pelatihan model pada fitur input ($X_{train60svr}$) dan nilai target yang sesuai ($y_{train60svr}$), memungkinkan algoritma SVR untuk mempelajari pola-pola mendasar dalam data.

d. Prediction

Prediksi dilakukan menggunakan model Support Vector Regression (SVR) yang telah di-fit sebelumnya dengan data latih (training data). Dengan menggunakan `regressor.predict(X_test40svr)`, model SVR memprediksi nilai target untuk data uji ($X_{test40svr}$). Kemudian, nilai prediksi yang diperoleh dibulatkan ke nilai integer menggunakan `np.round` dan diubah menjadi tipe data integer dengan `astype(int)`. Hasil prediksi ini dapat digunakan untuk evaluasi performa model atau keperluan analisis lebih lanjut.

```
[965] # Prediksi dengan data yang sudah dinormalisasi
y_pred60svr = np.round(regressor.predict(X_test40svr)).astype(int)

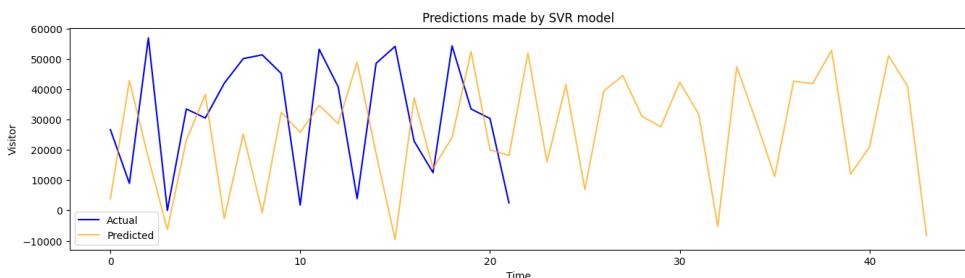
[966] y_pred60svr
array([ 3907, 42799, 17305, -6252, 23142, 38250, -2623, 25232, -663,
       32370, 25806, 34687, 28611, 48951, 18734, -9576, 37189, 13893,
       24041, 52417, 19990, 18182, 51929, 15983, 41435, 6951, 39463,
       44478, 31113, 27604, 42344, 31666, -5320, 47380, 29803, 11261,
      42615, 41857, 52785, 11922, 21019, 51019, 41023, -8244])

[967] def plotting_actual_vs_pred(y_test40svr, y_pred, title):
    plt.figure(figsize=(16, 4))
    plt.plot(y_test40svr, color='blue', label='Actual')
    plt.plot(y_pred, alpha=0.7, color='orange',
             label='Predicted')
    plt.title(title)
    plt.xlabel('Time')
    plt.ylabel('Visitor')
    plt.legend()
    plt.show()

[968] plotting_actual_vs_pred(y_test40svr, y_pred60svr, "Predictions made by SVR model")
```

Gambar 58. Coding Plot Perbandingan SVR

Plot berikut akan menampilkan perbandingan antara data aktual dan prediksi berdasarkan model SVR dengan pembagian data training dan test yang telah dilakukan.



e. Evaluate Model (Train)

Prediksi dilakukan menggunakan data latih (train data), dan hasil prediksi tersebut dibandingkan dengan nilai sebenarnya dari target variabel pada data latih. Metrik evaluasi yang umum digunakan antara lain Mean Squared

Error (MSE), Mean Absolute Error (MAE), dan Mean Absolute Percentage Error (MAPE).

```
[970] # Prediksi dengan data yang sudah dinormalisasi
y_pred60svr2 = np.round(regressor.predict(X_train60svr)).astype(int)

[971] y_pred60svr2

array([-6966, -4378, 49276, 50110, 572, 6052, 30355, 19785, 41142,
       45972, -7454, 40395, 73, 24648, -3284, 7438, 29099, 44088,
       22113, 47684, -1757, 15355, 4514, 48713, 7741, 34449, 14976,
       43276, -3490, 50370, -187, 2499, 23803, 1189, 1525, 3279,
       20824, 36279, 20250, 45139, 36106, 26770, 46915, 16947, 20445,
       35207, 25915, 43676, 39323, 34016, 33149, 9994, -3067, 23467,
       13395, 5359, 12561, 10514, 26597, 2824, 9095, 8034, -7335,
       11564])
```

Gambar 59. Hasil Prediksi Train SVR

```
[973] # Evaluate the model
mse_60_train = mean_squared_error(y_train60svr, y_pred60svr2)
mae_60_train = mean_absolute_error(y_train60svr, y_pred60svr2)
mape_60_train = mean_absolute_percentage_error(y_train60svr, y_pred60svr2)

print(f"Mean Absolute Error (MAE): {round(mae_60_train, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_60_train, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_60_train, 2)}%")

Mean Absolute Error (MAE): 9747.11
Mean Squared Error (MSE): 173004613.95
Mean Absolute Percentage Error (MAPE): 2.1340025359318385e+18%
```

Gambar 60. Evaluate Model Train SVR

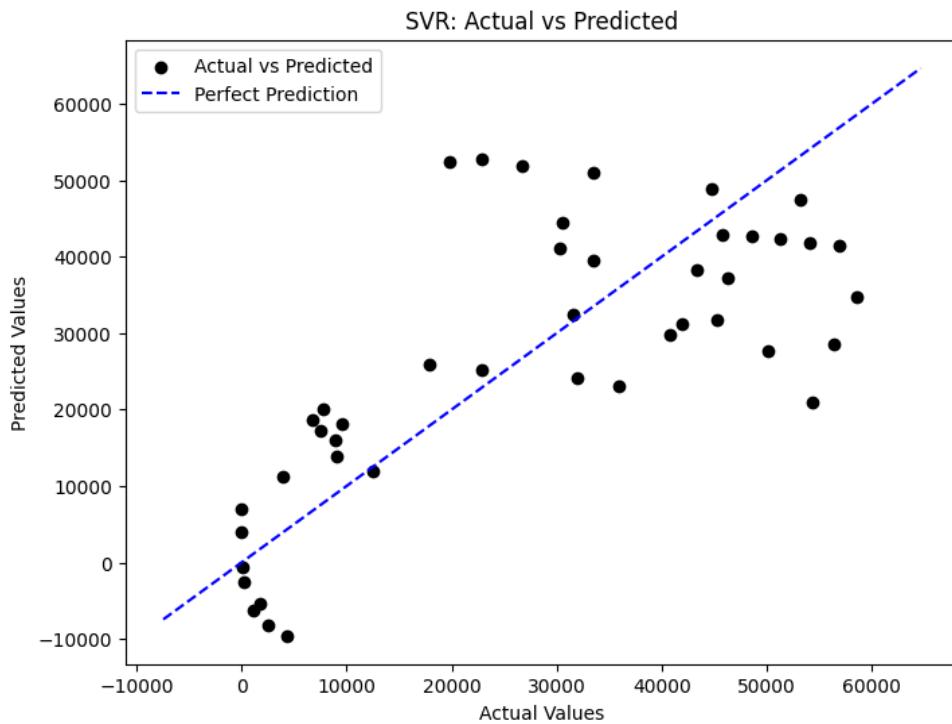
Scatter plot membantu untuk memvisualisasikan perbandingan antara nilai aktual dan nilai prediksi dari model Support Vector Regression (SVR). Setiap titik dalam scatter plot merepresentasikan pasangan nilai aktual dan nilai prediksi untuk setiap sampel pada data latih.

```
[974] # Scatter plot to visualize predicted vs actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test40svr, y_pred60svr, color='black', label='Actual vs Predicted')
# Plotting the diagonal line (perfect prediction)
max_value = max(max(y_train60svr), max(y_pred60svr2))
min_value = min(min(y_train60svr), min(y_pred60svr2))
plt.plot([min_value, max_value], [min_value, max_value], color='blue', linestyle='--', label='Perfect Prediction')

plt.title('SVR: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
```

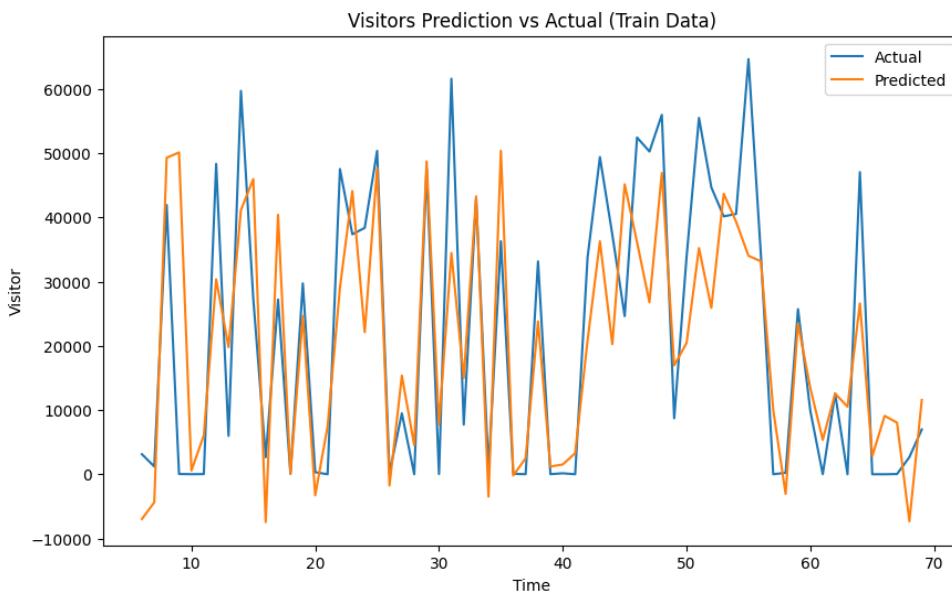
Gambar 61. Coding Plot Perbandingan SVR

Dengan adanya garis diagonal biru dengan pola putus-putus, plot ini memberikan gambaran visual tentang seberapa baik prediksi model SVR dibandingkan dengan nilai aktual. Jika semua titik sejajar dengan garis diagonal, itu menunjukkan model memiliki performa yang sempurna. Scatter plot ini membantu secara intuitif memahami sejauh mana model SVR mampu mereplikasi nilai sebenarnya pada data latih.



Gambar 62. Hasil Prediksi Train SVR

Berikut adalah grafik yang menunjukkan perbandingan antara data aktual dan prediksi dari data latih



Gambar 63. Hasil Grafik Perbandingan SVR

f. Evaluate Model (Test)

Prediksi dilakukan menggunakan data uji (test data), dan hasil prediksi tersebut dibandingkan dengan nilai sebenarnya dari target variabel pada data latih. Metrik evaluasi yang umum digunakan antara lain Mean Squared Error (MSE), Mean Absolute Error (MAE), dan Mean Absolute Percentage Error (MAPE).

```
[965] # Prediksi dengan data yang sudah dinormalisasi
y_pred60svr = np.round(regressor.predict(X_test40svr)).astype(int)

[966] y_pred60svr
array([ 3907, 42799, 17305, -6252, 23142, 38250, -2623, 25232, -663,
       32370, 25806, 34687, 28611, 48951, 18734, -9576, 37189, 13893,
      24041, 52417, 19990, 18182, 51929, 15983, 41435, 6951, 39463,
     44478, 31113, 27604, 42344, 31666, -5320, 47380, 29803, 11261,
     42615, 41857, 52785, 11922, 21019, 51019, 41023, -8244])
```

Gambar 64

```
[976] # Evaluate the model
mse_60_test = mean_squared_error(y_test40svr, y_pred60svr)
mae_60_test = mean_absolute_error(y_test40svr, y_pred60svr)
mape_60_test = mean_absolute_percentage_error(y_test40svr, y_pred60svr)

print(f"Mean Absolute Error (MAE): {round(mae_60_test, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_60_test, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_60_test, 2)}%")

Mean Absolute Error (MAE): 11279.61
Mean Squared Error (MSE): 197032349.57
Mean Absolute Percentage Error (MAPE): 10.27%
```

Gambar 65

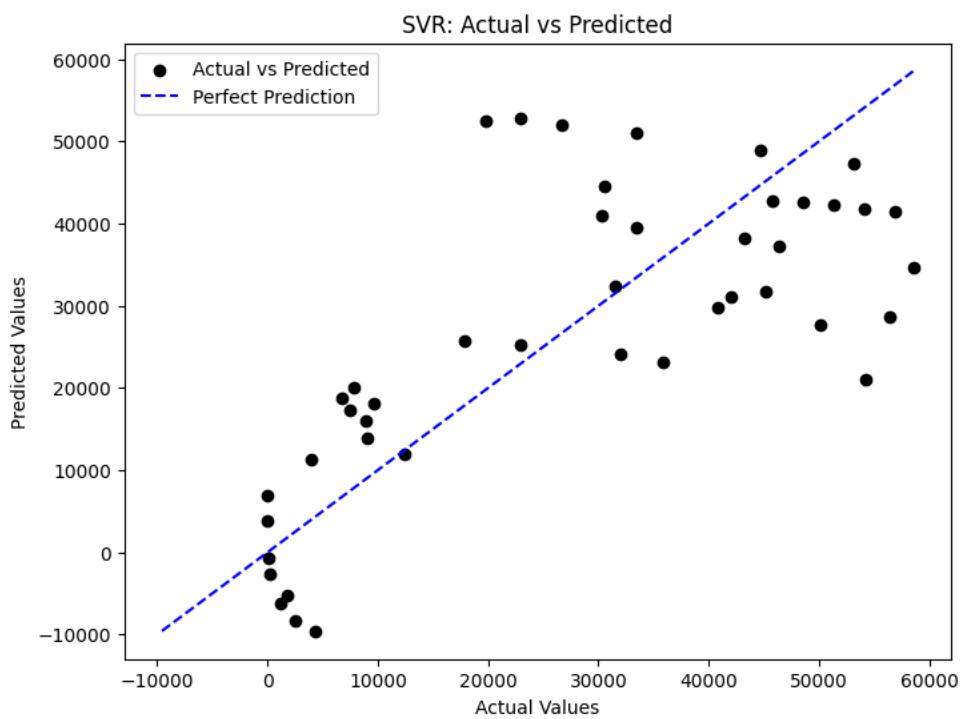
Scatter plot membantu untuk memvisualisasikan perbandingan antara nilai aktual dan nilai prediksi dari model Support Vector Regression (SVR). Setiap titik dalam scatter plot merepresentasikan pasangan nilai aktual dan nilai prediksi untuk setiap sampel pada data latih.

```
[977] # Scatter plot to visualize predicted vs actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test40svr, y_pred60svr, color='black', label='Actual vs Predicted')
# Plotting the diagonal line (perfect prediction)
max_value = max(max(y_test40svr), max(y_pred60svr))
min_value = min(min(y_test40svr), min(y_pred60svr))
plt.plot([min_value, max_value], [min_value, max_value], color='blue', linestyle='--', label='Perfect Prediction')

plt.title('SVR: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
```

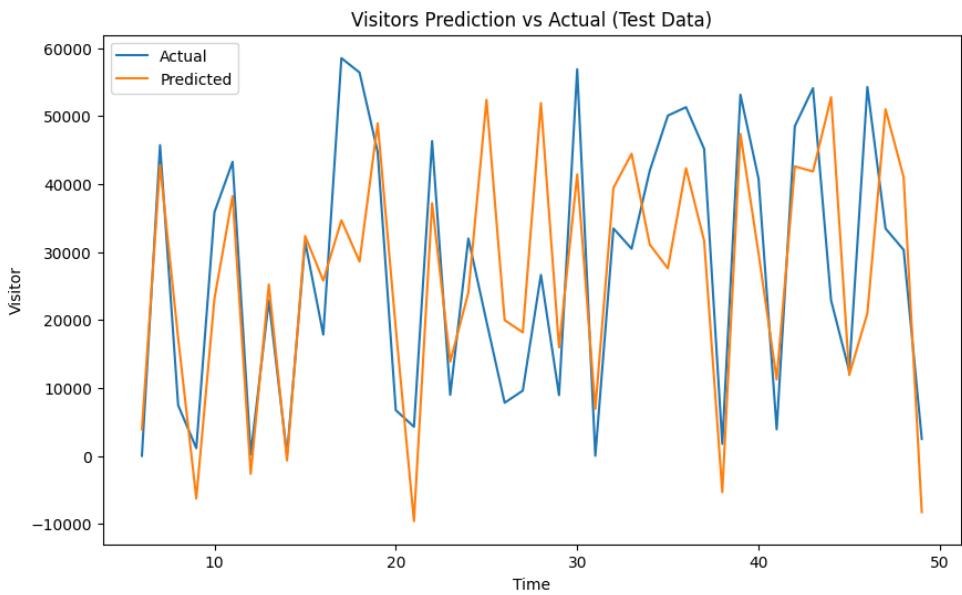
Gambar 66

Dengan adanya garis diagonal biru dengan pola putus-putus, plot ini memberikan gambaran visual tentang seberapa baik prediksi model SVR dibandingkan dengan nilai aktual. Jika semua titik sejajar dengan garis diagonal, itu menunjukkan model memiliki performa yang sempurna. Scatter plot ini membantu secara intuitif memahami sejauh mana model SVR mampu mereplikasi nilai sebenarnya pada data latih.



Gambar 67

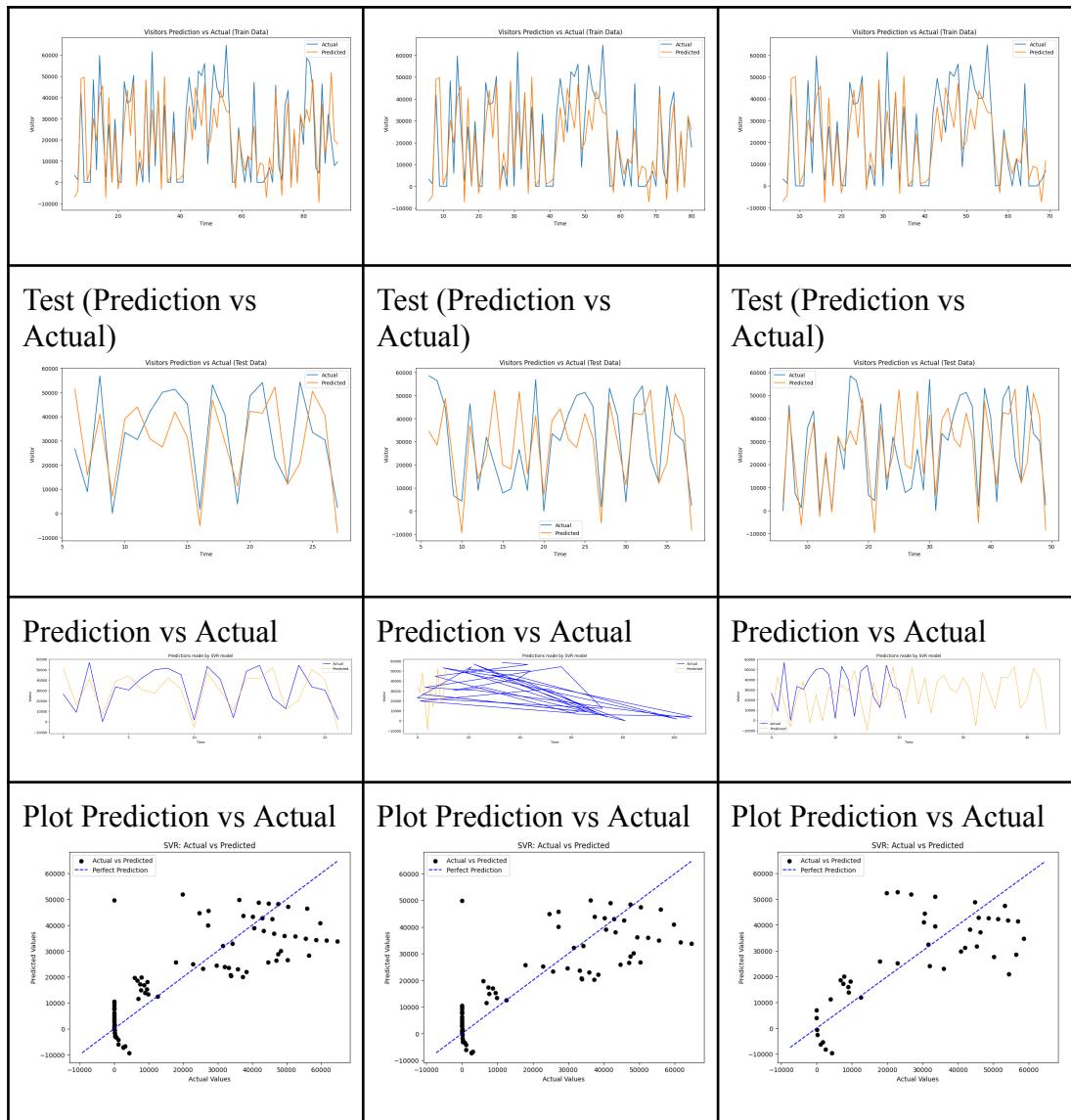
Berikut adalah grafik yang menunjukkan perbandingan antara data aktual dan prediksi dari data uji



Gambar 68

Tabel Perbandingan Grafik Antar Skenario

Scenario 80% Train	Scenario 70% Train	Scenario 60% Train
Train (Prediction vs Actual)	Train (Prediction vs Actual)	Train (Prediction vs Actual)



Scenario	Model	MSE	MAE	MAPE (%)
Scenario 1	RNN Train 80	1.70506e+08	9733.55	1.59789e+18
Scenario 1	RNN Test 80	2.29233e+08	12842.9	12.1119
Scenario 2	RNN Train 70	1.53466e+08	9072.44	1.84035e+18
Scenario 2	RNN Test 70	2.48207e+08	13306.4	8.44264
Scenario 3	RNN Train 60	1.73005e+08	9747.11	2.134e+18
Scenario 3	RNN Test 60	1.97032e+08	11279.6	10.2696

MAPE sering kali efektif untuk menganalisis kumpulan data yang besar dan membutuhkan penggunaan nilai kumpulan data selain nol. Nilai MAPE lebih mudah diinterpretasikan dibandingkan alat ukur yang lain karena berupa nilai persen yang dapat terukur skalanya. Berdasarkan nilai MAPE, skenario terbaik

pada model SVR adalah skenario 2 dengan pembagian 70% data latih (train) dan 30% data uji (test).

2. RNN

Setelah Praproses data dalam model RNN yang mencakup load dataset hingga melakukan scaling dengan menggunakan MinMaxScaler, dilakukan proses selanjutnya yaitu melakukan analisis terhadap pembagian Train dan test Data baik melakukan pembagian sesuai skenario, membuat Model RNN hingga memunculkan hasil data Forecast yang telah terbuat dengan memperhitungkan MAPE,MAE, dan MSE.

- Skenario 80 % Train

Selanjutnya yaitu, melakukan analisis RNN dengan Skenario 80 % Train dan 20 % test dengan melalui beberapa tahapan dibawah ini :

- a. Splitting Train and Test Data

Membagi data pelatihan dan pengujian dengan rasio 80% dan 20% serta menetapkan hasilnya ke dalam variabel baru, seperti yang ditunjukkan di bawah ini:

```
[ ] train_ratio = 0.8
# train_ratio = 0.7
# train_ratio = 0.6

[ ] train_size = int(train_ratio * len(sequences))

X_train, X_test = sequences[:train_size], sequences[train_size:]
y_train, y_test = labels[:train_size].reshape(-1,1), labels[train_size:].reshape(-1,1)

print("Train X shape:", X_train.shape)
print("Train Y shape:", y_train.shape)
print("Test X shape:", X_test.shape)
print("Test Y shape:", y_test.shape)

Train X shape: (81, 6, 2)
Train Y shape: (81, 1)
Test X shape: (21, 6, 2)
Test Y shape: (21, 1)
```

Gambar 69. Splitting train & Test RNN 80 %

Dalam penjelasan di atas, yaitu menghitung ukuran data pelatihan berdasarkan proporsi tertentu dari total urutan. Selanjutnya, menggunakan nilai tersebut untuk memisahkan urutan menjadi data pelatihan dan pengujian yang menjadi variabel baru Variabel X_train, X_test, y_train, dan y_test kemudian digunakan untuk menyimpan data dan label yang sesuai.Lalu, melakukan print untuk menampilkan ukuran dari data pelatihan dan pengujian untuk memastikan proses pemisahan berjalan dengan baik.

- b. Create Model

Tahap Selanjutnya, yaitu memanggil model RNN yang diimport dari library tensorflow untuk membuat kerangka kerja model RNN dengan menggunakan beberapa parameter yang telah ditentukan pada kode berikut.

```
[ ] # Inisiasi pembuatan model RNN
model = Sequential()

# Model RNN
model.add(SimpleRNN(40,activation="tanh",return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.15))
model.add(SimpleRNN(40,activation="tanh",return_sequences=True))
model.add(Dropout(0.15))
model.add(SimpleRNN(40,activation="tanh",return_sequences=False))
model.add(Dropout(0.15))
model.add(Dense(1))
```

Gambar 70. Splitting train & Test RNN 80 %

Kode tersebut mengawali proses pembuatan model RNN dengan menggunakan framework Keras. Model ini dirancang untuk melakukan prediksi pada suatu rangkaian data time series. Berikut adalah tahap-tahap konfigurasi model, yaitu yang pertama menambahkan Layer RNN Pertama, Menambahkan layer RNN pertama dengan 40 unit dan fungsi aktivasi tangen hiperbolik (tanh). `return_sequences=True` menunjukkan bahwa layer ini mengembalikan urutan sekuensial. `input_shape=(X_train.shape[1], X_train.shape[2])` menentukan bentuk input model berdasarkan dimensi data pelatihan.

yang kedua, menambahkan Dropout Layer Pertama untuk menambahkan layer dropout pertama dengan tingkat dropout sebesar 0.15. Dropout digunakan untuk mencegah overfitting dengan secara acak mengabaikan sebagian unit selama pelatihan. Lalu Tambahkan Layer RNN Kedua untuk menambahkan layer RNN kedua dengan 40 unit dan fungsi aktivasi tanh. `return_sequences=True` menunjukkan bahwa layer ini juga mengembalikan urutan sekuensial. keempat untuk menambahkan Dropout Layer Kedua dengan tingkat dropout sebesar 0.15.

Lalu ada Model Summary yang ditunjukkan pada gambar dibawah ini yang berfungsi sebagai rekapan hasil inisiasi diatas.

```
[ ] model.summary()

Model: "sequential_6"
=====
Layer (type)          Output Shape         Param #
=====
simple_rnn_18 (SimpleRNN)    (None, 6, 40)           1720
dropout_18 (Dropout)        (None, 6, 40)            0
simple_rnn_19 (SimpleRNN)    (None, 6, 40)           3240
dropout_19 (Dropout)        (None, 6, 40)            0
simple_rnn_20 (SimpleRNN)    (None, 40)              3240
dropout_20 (Dropout)        (None, 40)              0
dense_6 (Dense)            (None, 1)                41
=====
Total params: 8241 (32.19 KB)
Trainable params: 8241 (32.19 KB)
Non-trainable params: 0 (0.00 Byte)
```

Gambar 71. model Summary RNN

c. Fit Model

Proses `fit` pada model RNN merupakan tahap pelatihan, di mana model secara iteratif memperbarui parameter-parameter internalnya agar dapat memahami pola dan hubungan dalam data pelatihan. Fungsi ini

memanfaatkan algoritma pembelajaran yang diatur oleh optimizer, yang dalam kasus ini digunakan adalah algoritma Adam. Saat proses pelatihan, model mencoba meminimalkan nilai fungsi kerugian (loss function), yang mengukur seberapa baik prediksi model sesuai dengan nilai aktual. Dropout layers, yang ditambahkan pada model, membantu mencegah overfitting dengan secara acak mengabaikan sejumlah unit selama pelatihan. Setelah iterasi pelatihan selesai, model akan dapat membuat prediksi yang lebih akurat pada data yang tidak pernah dilihat sebelumnya. Keseluruhan proses ini bertujuan untuk menghasilkan model RNN yang dapat memahami dan memprediksi pola dalam data time series dengan lebih baik.

```
[3754] # Train model
model.compile(optimizer="adam", loss="MSE")
history = model.fit(X_train, y_train, epochs=10, batch_size=1000, verbose=2, validation_split=0.2)

Epoch 1/10
1/1 - 4s - loss: 0.4878 - val_loss: 0.2423 - 4s/epoch - 4s/step
Epoch 2/10
1/1 - 0s - loss: 0.2373 - val_loss: 0.0197 - 50ms/epoch - 50ms/step
Epoch 3/10
1/1 - 0s - loss: 0.3685 - val_loss: 0.0070 - 48ms/epoch - 48ms/step
Epoch 4/10
1/1 - 0s - loss: 0.3348 - val_loss: 0.0088 - 50ms/epoch - 50ms/step
Epoch 5/10
1/1 - 0s - loss: 0.2639 - val_loss: 0.0054 - 76ms/epoch - 76ms/step
Epoch 6/10
1/1 - 0s - loss: 0.1705 - val_loss: 9.7962e-04 - 64ms/epoch - 64ms/step
Epoch 7/10
1/1 - 0s - loss: 0.2583 - val_loss: 0.0011 - 66ms/epoch - 66ms/step
Epoch 8/10
1/1 - 0s - loss: 0.2879 - val_loss: 7.9177e-04 - 50ms/epoch - 50ms/step
Epoch 9/10
1/1 - 0s - loss: 0.3640 - val_loss: 0.0019 - 52ms/epoch - 52ms/step
Epoch 10/10
1/1 - 0s - loss: 0.2064 - val_loss: 0.0139 - 53ms/epoch - 53ms/step
```

Gambar 72. model Summary RNN

Untuk menunjukkan apakah model RNN memiliki kecocokan yang kuat dengan dataset dapat diukur dengan menggunakan r-squared seperti gambar dibawah ini, dimana r-squared 0.28 menunjukkan hubungan kuat dengan model

```
[3756] # Evaluasi Nilai R^2
from sklearn.metrics import r2_score #ini nanti dipindah ke atas biar ngga bingung
score = r2_score(y_test,predictions)
print("R-Squared Score of RNN model",score)

R-Squared Score of RNN model 0.2818064393453976
```

Gambar 73. model Summary RNN

Selanjutnya, yaitu Plot berikut akan menampilkan perbandingan antara data aktual dan prediksi berdasarkan model SVR dengan pembagian data training dan test yang telah dilakukan.

```
[3757] def plotting_actual_vs_pred(y_test, y_pred, title):
    plt.figure(figsize=(16, 4))
    plt.plot(y_test, color='blue', label='Actual')
    plt.plot(y_pred, alpha=0.7, color='orange',
             label='Predicted')
    plt.title(title)
    plt.xlabel('Time')
    plt.ylabel('Visitor')
    plt.legend()
    plt.show()

[3758] plotting_actual_vs_pred(y_test, predictions, "Predictions made by RNN model")
```

Gambar 14. Plot Predicted vs Actual RNN

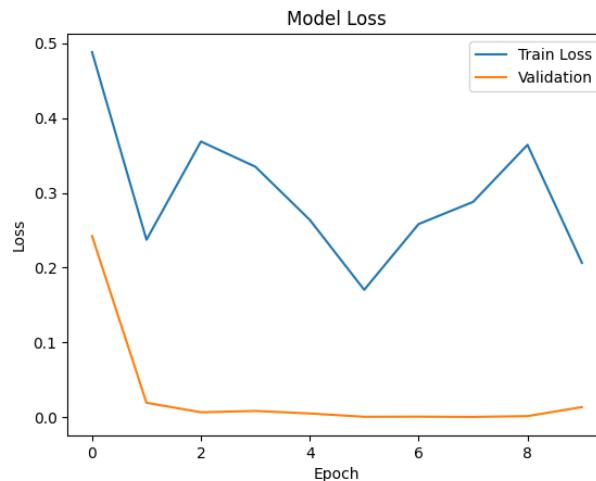
d. Loss Model

Tahap selanjutnya yaitu, mengenai Loss Model. Loss Model sendiri adalah fungsi matematis yang menyederhanakan perbedaan antara prediksi model dengan nilai sebenarnya. Sebagai contoh, dalam regresi, fungsi kerugian umumnya adalah Mean Squared Error (MSE), yang menghitung rata-rata dari kuadrat selisih antara prediksi dan ground truth. Proses pelatihan model melibatkan iteratif memperbarui parameter-parameter model untuk meminimalkan nilai loss ini. Sebagai penerapan loss model, maka dibuat seperti berikut, dimana loss model diambil dari Fit RNN diatas.

```
[3759] # Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train Loss', 'Validation'], loc='upper right')
plt.show()
```

Gambar 75. Loss Model

Berdasarkan plot pada gambar diatas, dapat diartikan sebagai pemberian label, legenda berdasarkan data history yang menghasilkan plot train loss dan val loss seperti berikut



Gambar 76. Hasil Loss Model

e. Prediction

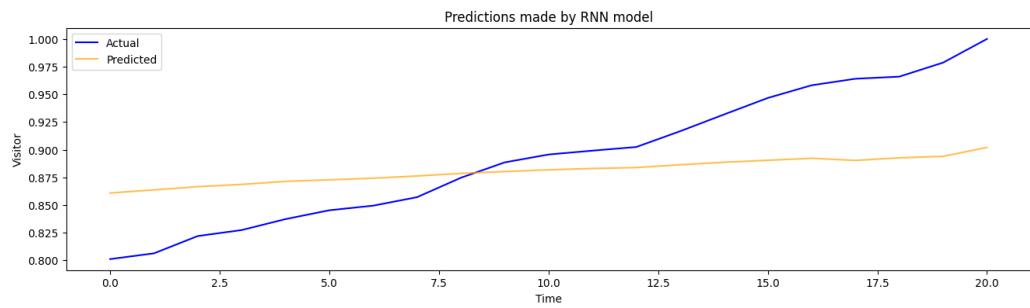
Selanjutnya yaitu, melakukan prediksi terhadap model untuk X_test, dimana prediction sendiri dalam RNN melibatkan mengalirkan input melalui model dan memperoleh output prediksi. Hasil ini kemudian dapat dibandingkan dengan nilai sebenarnya untuk mengevaluasi seberapa baik model dapat melakukan prediksi. Pada umumnya, hasil prediksi RNN dapat digunakan untuk berbagai aplikasi, termasuk prediksi time series, pengenalan pola, dan tugas-tugas lain yang melibatkan urutan data.

```
[3755] # Melakukan prediksi pada test set
predictions = model.predict(X_test)
```

1/1 [=====] - 1s 530ms/step

Gambar 77. Fungsi Predict RNN

Berdasarkan gambar diatas, maka prediksi berhasil dijalankan dengan menggunakan fungsi predict dan menghasilkan plot seperti dibawah ini.



Gambar 78. Hasil Prediksi RNN

f. Evaluate Model (Train)

Selanjutnya yaitu tahap prediksi pada model data train, dimana data train yang akan diprediksi yaitu X_train, dilanjut dengan penerapan fungsi scaler Inverse untuk mengembalikan nilai-nilai total_visitor dari y_train hingga memplot hasil Train Set.

```
[ ] # Membuat prediksi visitor menggunakan X_train
y_train_pred = model.predict(X_train)

3/3 [=====] - 0s 5ms/step

[ ] # Mengembalikan skala nilai visitor dari y_train
y_train_copies = np.repeat(y_train.reshape(-1, 1), X_train.shape[-1], axis=-1)
y_train_real = scaler.inverse_transform(y_train_copies)[:,1]
y_train_real

array([ 9348.,  9378.,  9400.,  9495.,  9523.,  9566.,  9653.,  9730.,
       9791.,  9827.,  9865.,  9902.,  9946.,  9963.,  9988.,  10033.,
      10072.,  10099.,  10118.,  10168.,  10254.,  10267.,  10366.,  10464.,
      10543.,  10564.,  10647.,  10695.,  10717.,  10757.,  10837.,  10909.,
      10974.,  11025.,  11095.,  11146.,  11211.,  11256.,  11349.,  11426.,
     11442.,  11505.,  11515.,  11568.,  11622.,  11678.,  11700.,  11731.,
     11761.,  11856.,  11957.,  11975.,  12010.,  12028.,  12052.,  12071.,
     12168.,  12219.,  12300.,  12333.,  12422.,  12480.,  12515.,  12615.,
     12661.,  12738.,  12797.,  12830.,  12858.,  12927.,  12975.,  13058.,
     13156.,  13183.,  13211.,  13256.,  13339.,  13403.,  13481.,  13537.,
     13595.])

[ ] # Mengembalikan skala nilai visitor dari y_train_pred
y_train_pred_copies = np.repeat(y_train_pred, X_train.shape[-1], axis=-1)
y_train_pred_real = scaler.inverse_transform(y_train_pred_copies)[:,1]
y_train_pred_real

array([ 9063.184,  8080.438,  7678.5244,  7796.6333,  9003.885,
       7100.956,  8476.162,  9170.351,  9564.966,  9192.057,
       8796.822,  9458.17,  9014.931,  8826.528,  8855.149,
       9010.668,  9021.572,  9045.548,  9052.425,  9739.292,
      10056.893,  9637.625,  9299.971,  9856.013,  9606.0205,
      9427.421,  9420.271,  9563.552,  9583.547,  9619.292,
      9579.816,  10252.634,  10578.523,  10091.22,  10052.015,
      10504.664,  10674.814,  10409.262,  10447.193,  10424.168,
      10750.373,  10718.004,  10748.746,  11399.381,  11589.976,
     11460.104,  11566.093,  12048.498,  11979.586,  11947.394,
     11919.645,  11688.204,  11886.664,  11818.111,  12010.961,
     12174.232,  12421.155,  12770.632,  12557.881,  13139.532,
     13460.035,  13445.885,  13392.828,  13570.086,  13511.244,
     13565.231,  13552.528,  13636.842,  13630.586,  13701.884,
     13712.348,  13737.886,  13856.494,  13859.691,  13942.023,
     13882.689,  13906.068,  13929.461,  13939.814,  13949.449,
     13960.9795], dtype=float32)
```

Gambar 79. Inverse RNN

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat

diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
[ ] # Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_80_train = mean_squared_error(y_train_real, y_train_pred_real)
mae_80_train = mean_absolute_error(y_train_real, y_train_pred_real)
mape_80_train = mean_absolute_percentage_error(y_train_real, y_train_pred_real)

print(f"Mean Absolute Error (MAE): {round(mae_80_train, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_80_train, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_80_train, 2)}%")

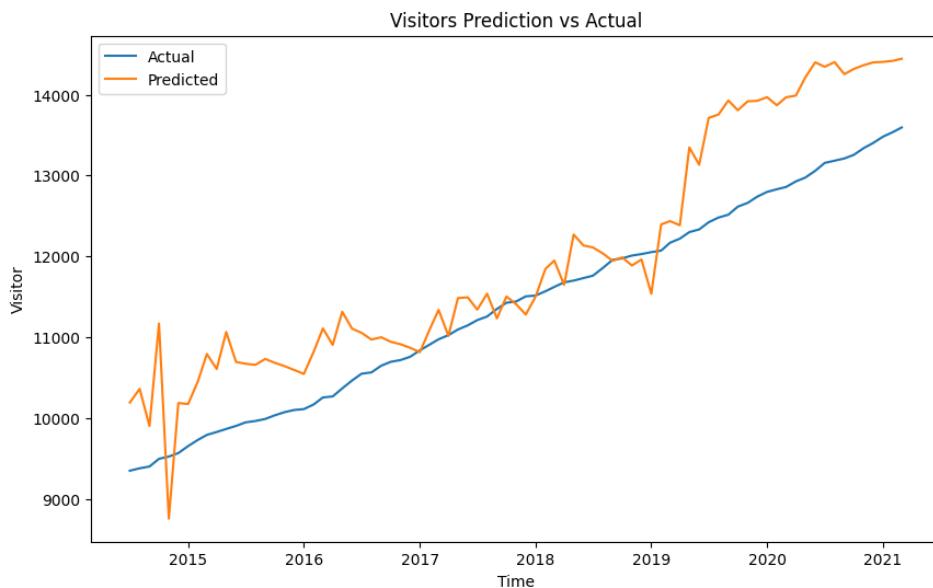
Mean Absolute Error (MAE): 722.64
Mean Squared Error (MSE): 703572.71
Mean Absolute Percentage Error (MAPE): 0.07%
```

Gambar 80. Hasil MSE, MAE, dan MAPE RNN

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model RNN mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model RNN dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.

```
❸ # Melakukan plotting hasil prediksi visitor terhadap data visitor asli dari train set
plt.figure(figsize=(10, 6))
plt.plot(ntt.index[sequence_length:y_train.shape[0]+sequence_length], y_train_real, label='Actual')
plt.plot(ntt.index[sequence_length:y_train.shape[0]+sequence_length], y_train_pred_real, label='Predicted')
plt.title('Visitors Prediction vs Actual')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()
```



Gambar 81. Perbandingan RNN

g. Evaluate Model (Test)

Selanjutnya yaitu tahap prediksi pada model data test, dimana data test yang akan diprediksi yaitu X_test, dilanjut dengan penerapan fungsi scaler Inverse untuk mengembalikan nilai-nilai total_visitor dari y_test hingga memplot hasil Test Set.

```
[ ] # Membuat prediksi visitor menggunakan X_test
y_test_pred = model.predict(X_test)

1/1 [=====] - 0s 23ms/step

[ ] # Mengembalikan skala nilai visitor dari y_test
y_test_copies = np.repeat(y_test.reshape(-1, 1), X_test.shape[-1], axis=-1)
y_test_real = scaler.inverse_transform(y_test_copies)[:,1]
y_test_real

array([13637., 13667., 13757., 13788., 13845., 13891., 13915., 13959.,
       14060., 14140., 14181., 14201., 14220., 14302., 14389., 14475.,
       14541., 14575., 14586., 14659., 14782.])

[ ] # Mengembalikan skala nilai visitor dari y_train_pred
y_test_pred_copies = np.repeat(y_test_pred, X_test.shape[-1], axis=-1)
y_test_pred_real = scaler.inverse_transform(y_test_pred_copies)[:,1]
y_test_pred_real

array([13980.848 , 13997.009 , 14013.819 , 14025.61 , 14041.228 ,
       14049.312 , 14057.634 , 14069.402 , 14082.789 , 14092.351 ,
       14101.326 , 14108.266 , 14113.154 , 14127.7295, 14140.663 ,
       14151.004 , 14161.203 , 14150.269 , 14163.691 , 14171.466 ,
       14217.787 ], dtype=float32)
```

Gambar 82. Splitting Data

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
[ ] # Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_80_test = mean_squared_error(y_test_real, y_test_pred_real)
mae_80_test = mean_absolute_error(y_test_real, y_test_pred_real)
mape_80_test = mean_absolute_percentage_error(y_test_real, y_test_pred_real)

print(f"Mean Absolute Error (MAE): {round(mae_80_test, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_80_test, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_80_test, 2)}%")

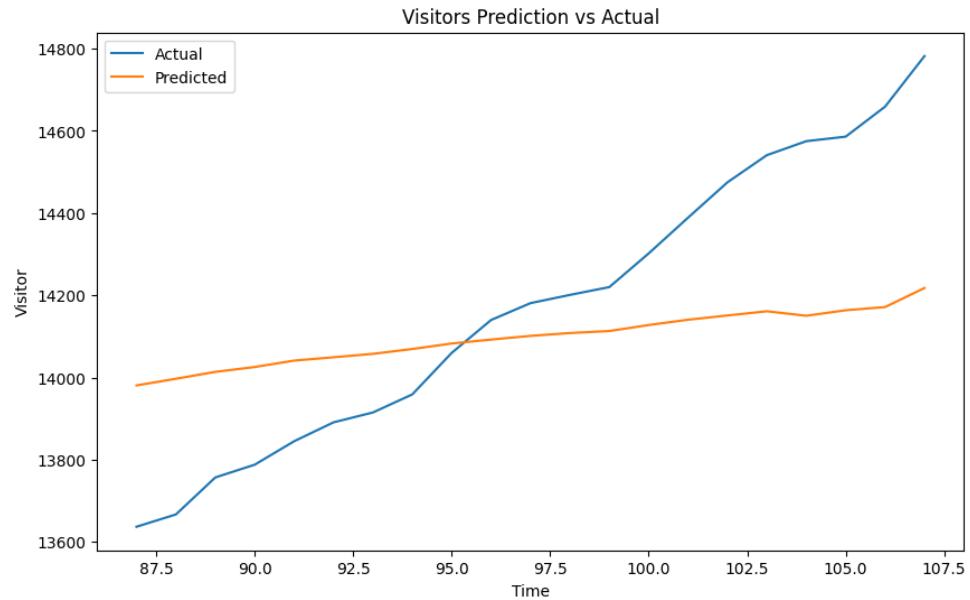
Mean Absolute Error (MAE): 245.27
Mean Squared Error (MSE): 82420.66
Mean Absolute Percentage Error (MAPE): 0.02%
```

Gambar 83.

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model RNN mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model RNN dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.

```
❸ # Melakukan plotting hasil prediksi visitor terhadap data visitor asli dari test set
plt.figure(figsize=(10, 6))
plt.plot(ntt.index[-y_test.shape[0]:], y_test_real, label='Actual')
plt.plot(ntt.index[-y_test.shape[0]:], y_test_pred_real, label='Predicted')
plt.title('Visitors Prediction vs Actual')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()
```



Gambar 84

- Skenario 70 % Train

Selanjutnya yaitu, melakukan analisis RNN dengan Skenario 70 % Train dan 30 % test dengan melalui beberapa tahapan dibawah ini :

- a. Splitting Train and Test Data

Membagi data pelatihan dan pengujian dengan rasio 70% dan 30% serta menetapkan hasilnya ke dalam variabel baru, seperti yang ditunjukkan di bawah ini:

```
[ ] ntt = ntt.set_index('datetime')

[ ] train_ratio = 0.7

[ ] train_size = int(train_ratio * len(sequences))
# pakai panjang sequences karena panjang dataset sudah berkurang 5 karena dibuat sequences

X_train, X_test = sequences[:train_size], sequences[train_size:]
y_train, y_test = labels[:train_size].reshape(-1,1), labels[train_size: ].reshape(-1,1)

print("Train X shape:", X_train.shape)
print("Train Y shape:", y_train.shape)
print("Test X shape:", X_test.shape)
print("Test Y shape:", y_test.shape)
```

Train X shape: (71, 6, 2)
 Train Y shape: (71, 1)
 Test X shape: (31, 6, 2)
 Test Y shape: (31, 1)

Gambar 85. Splitting train & Test RNN 70 %

Dalam penjelasan di atas, yaitu menghitung ukuran data pelatihan berdasarkan proporsi tertentu dari total urutan. Selanjutnya, menggunakan nilai tersebut untuk memisahkan urutan menjadi data pelatihan dan pengujian yang menjadi variabel baru Variabel X_train, X_test, y_train, dan y_test kemudian digunakan untuk menyimpan data dan label yang sesuai.Lalu, melakukan print untuk menampilkan ukuran dari data pelatihan dan pengujian untuk memastikan proses pemisahan berjalan dengan baik.

- b. Create Model

Tahap Selanjutnya, yaitu memanggil model RNN yang diimporkan dari library tensorflow untuk membuat kerangka kerja model RNN dengan menggunakan beberapa parameter yang telah ditentukan pada kode berikut.

```
[ ] # Inisiasi pembuatan model
model = Sequential()

# Membuat model RNN
# Activation ada relu, sigmoid, tanh
model.add(SimpleRNN(40,activation="tanh",return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.15))
model.add(SimpleRNN(40,activation="tanh",return_sequences=True))
model.add(Dropout(0.15))
model.add(SimpleRNN(40,activation="tanh",return_sequences=False))
model.add(Dropout(0.15))
model.add(Dense(1))
```

Gambar 86. Splitting train & Test RNN 80 %

Kode tersebut mengawali proses pembuatan model RNN dengan menggunakan framework Keras. Model ini dirancang untuk melakukan prediksi pada suatu rangkaian data time series. Berikut adalah tahap-tahap konfigurasi model, yaitu yang pertama menambahkan Layer RNN Pertama, Menambahkan layer RNN pertama dengan 40 unit dan fungsi aktivasi tangen hiperbolik (tanh). `return_sequences=True` menunjukkan bahwa layer ini mengembalikan urutan sekuensial. `input_shape=(X_train.shape[1], X_train.shape[2])` menentukan bentuk input model berdasarkan dimensi data pelatihan.

Untuk yang kedua, menambahkan Dropout Layer Pertama untuk menambahkan layer dropout pertama dengan tingkat dropout sebesar 0.15. Dropout digunakan untuk mencegah overfitting dengan secara acak mengabaikan sebagian unit selama pelatihan. Lalu Tambahkan Layer RNN Kedua untuk menambahkan layer RNN kedua dengan 40 unit dan fungsi aktivasi tanh. `return_sequences=True` menunjukkan bahwa layer ini juga mengembalikan urutan sekuensial. Keempat untuk menambahkan Dropout Layer Kedua dengan tingkat dropout sebesar 0.15.

Lalu ada Model Summary yang ditunjukkan pada gambar dibawah ini yang berfungsi sebagai rekapan hasil inisiasi diatas.

```
[ ] model.summary()

Model: "sequential_106"
=====
Layer (type)          Output Shape         Param #
=====
simple_rnn_318 (SimpleRNN)    (None, 6, 40)      1720
dropout_318 (Dropout)        (None, 6, 40)      0
simple_rnn_319 (SimpleRNN)    (None, 6, 40)      3240
dropout_319 (Dropout)        (None, 6, 40)      0
simple_rnn_320 (SimpleRNN)    (None, 40)         3240
dropout_320 (Dropout)        (None, 40)         0
dense_106 (Dense)           (None, 1)          41
=====
Total params: 8241 (32.19 KB)
Trainable params: 8241 (32.19 KB)
Non-trainable params: 0 (0.00 Byte)
```

Gambar 87. model Summary RNN 70%

c. Fit Model

Proses 'fit' pada model RNN merupakan tahap pelatihan, di mana model secara iteratif memperbarui parameter-parameter internalnya agar dapat

memahami pola dan hubungan dalam data pelatihan. Fungsi ini memanfaatkan algoritma pembelajaran yang diatur oleh optimizer, yang dalam kasus ini digunakan adalah algoritma Adam. Saat proses pelatihan, model mencoba meminimalkan nilai fungsi kerugian (loss function), yang mengukur seberapa baik prediksi model sesuai dengan nilai aktual. Dropout layers, yang ditambahkan pada model, membantu mencegah overfitting dengan secara acak mengabaikan sejumlah unit selama pelatihan. Setelah iterasi pelatihan selesai, model akan dapat membuat prediksi yang lebih akurat pada data yang tidak pernah dilihat sebelumnya. Keseluruhan proses ini bertujuan untuk menghasilkan model RNN yang dapat memahami dan memprediksi pola dalam data time series dengan lebih baik.

```
[ ] # Train model
# Pilihlah optimizer adam, huber, spatial_cossentropy
model.compile(optimizer="adam", loss="MSE")
history = model.fit(X_train, y_train, epochs=10, batch_size=1000, verbose=2, validation_split=0.2)

Epoch 1/10
1/1 - 3s - loss: 0.2773 - val_loss: 0.0838 - 3s/epoch - 3s/step
Epoch 2/10
1/1 - 0s - loss: 0.3762 - val_loss: 0.0276 - 42ms/epoch - 42ms/step
Epoch 3/10
1/1 - 0s - loss: 0.3535 - val_loss: 0.0180 - 39ms/epoch - 39ms/step
Epoch 4/10
1/1 - 0s - loss: 0.2320 - val_loss: 0.0671 - 37ms/epoch - 37ms/step
Epoch 5/10
1/1 - 0s - loss: 0.3323 - val_loss: 0.0576 - 37ms/epoch - 37ms/step
Epoch 6/10
1/1 - 0s - loss: 0.2996 - val_loss: 0.0143 - 38ms/epoch - 38ms/step
Epoch 7/10
1/1 - 0s - loss: 0.1228 - val_loss: 0.0037 - 37ms/epoch - 37ms/step
Epoch 8/10
1/1 - 0s - loss: 0.1788 - val_loss: 0.0090 - 40ms/epoch - 40ms/step
Epoch 9/10
1/1 - 0s - loss: 0.1771 - val_loss: 0.0149 - 39ms/epoch - 39ms/step
Epoch 10/10
1/1 - 0s - loss: 0.1761 - val_loss: 0.0170 - 39ms/epoch - 39ms/step
```

Gambar 88. model Summary RNN

Untuk menunjukkan apakah model RNN memiliki kecocokan yang kuat dengan dataset dapat diukur dengan menggunakan r-squared seperti gambar dibawah ini, dimana r-squared 0.28 menunjukkan hubungan kuat dengan model

```
[ ] # Evaluasi Nilai R^2
from sklearn.metrics import r2_score #ini nanti dipindah ke atas biar ngga bingung
score = r2_score(y_test,predictions)
print("R-Squared Score of RNN model",score)
```

R-Squared Score of RNN model -2.59575132496986

Gambar 89. model Summary RNN

Selanjutnya, yaitu Plot berikut akan menampilkan perbandingan antara data aktual dan prediksi berdasarkan model SVR dengan pembagian data training dan test yang telah dilakukan.

```
[3757] def plotting_actual_vs_pred(y_test, y_pred, title):
    plt.figure(figsize=(16, 4))
    plt.plot(y_test, color='blue', label='Actual')
    plt.plot(y_pred, alpha=0.7, color='orange',
             label='Predicted')
    plt.title(title)
    plt.xlabel('Time')
    plt.ylabel('Visitor')
    plt.legend()
    plt.show()

[3758] plotting_actual_vs_pred(y_test, predictions, "Predictions made by RNN model")
```

Gambar 90. Plot Predicted vs Actual RNN

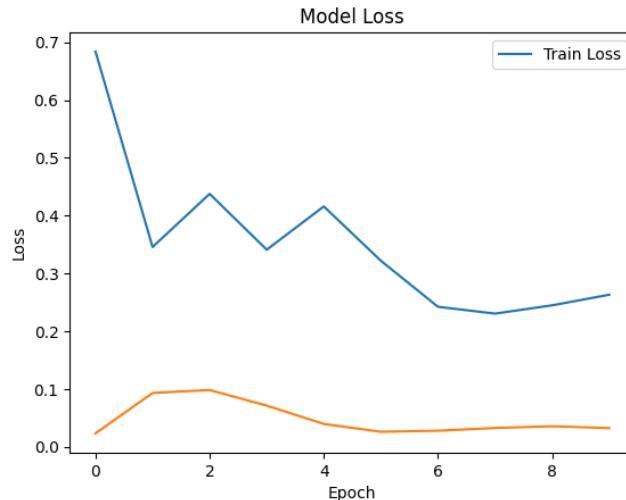
d. Loss Model

Tahap selanjutnya yaitu, mengenai Loss Model. Loss Model sendiri adalah fungsi matematis yang menyederhanakan perbedaan antara prediksi model dengan nilai sebenarnya. Sebagai contoh, dalam regresi, fungsi kerugian umumnya adalah Mean Squared Error (MSE), yang menghitung rata-rata dari kuadrat selisih antara prediksi dan ground truth. Proses pelatihan model melibatkan iteratif memperbarui parameter-parameter model untuk meminimalkan nilai loss ini. Sebagai penerapan loss model, maka dibuat seperti berikut, dimana loss model diambil dari Fit RNN diatas.

```
[3759] # Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train Loss', 'Validation'], loc='upper right')
plt.show()
```

Gambar 91. Loss Model

Berdasarkan plot pada gambar diatas, dapat diartikan sebagai pemberian label, legenda berdasarkan data history yang menghasilkan plot train loss dan val loss seperti berikut



Gambar 92. Hasil Loss Model

e. Prediction

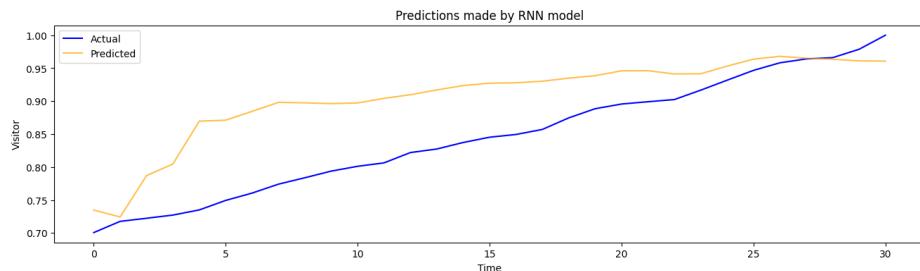
Selanjutnya yaitu, melakukan prediksi terhadap model untuk X_{test} , dimana prediction sendiri dalam RNN melibatkan mengalirkan input melalui model dan memperoleh output prediksi. Hasil ini kemudian dapat dibandingkan dengan nilai sebenarnya untuk mengevaluasi seberapa baik model dapat melakukan prediksi. Pada umumnya, hasil prediksi RNN dapat digunakan untuk berbagai aplikasi, termasuk prediksi time series, pengenalan pola, dan tugas-tugas lain yang melibatkan urutan data.

```
[63] # Membuat prediksi visitor menggunakan X_train  
y_train_pred = model.predict(X_train)
```

```
3/3 [=====] - 0s 7ms/step
```

Gambar 93. Fungsi Predict RNN

Berdasarkan gambar diatas, maka prediksi berhasil dijalankan dengan menggunakan fungsi predict dan menghasilkan plot seperti dibawah ini.



Gambar 94. Hasil Prediksi RNN

f. Evaluate Model (Train)

Selanjutnya yaitu tahap prediksi pada model data train, dimana data train yang akan diprediksi yaitu X_train, dilanjut dengan penerapan fungsi scaler Inverse untuk mengembalikan nilai-nilai total_visitor dari y_train hingga memplot hasil Train Set.

```
[63] # Membuat prediksi visitor menggunakan X_train  
y_train_pred = model.predict(X_train)
```

```
3/3 [=====] - 0s 7ms/step
```

```
❶ # Mengembalikan skala nilai visitor dari y_train  
y_train_copies = np.repeat(y_train.reshape(-1, 1), X_train.shape[-1], axis=-1)  
y_train_real = scaler.inverse_transform(y_train_copies)[:,1]  
  
❷ array([ 9348.,  9378.,  9400.,  9495.,  9523.,  9566.,  9653.,  9730.,  
    9791.,  9827.,  9865.,  9902.,  9946.,  9963.,  9988., 10033.,  
    10072., 10099., 10110., 10168., 10254., 10267., 10366., 10464.,  
    10548., 10564., 10647., 10695., 10717., 10757., 10837., 10909.,  
    10974., 11025., 11095., 11146., 11211., 11256., 11349., 11426.,  
    11442., 11505., 11515., 11568., 11622., 11678., 11700., 11731.,  
    11761., 11856., 11957., 11975., 12010., 12028., 12052., 12071.,  
    12168., 12219., 12300., 12333., 12422., 12480., 12515., 12615.,  
    12661., 12738., 12797., 12830., 12858., 12927., 12975.])
```

```
[65] # Mengembalikan skala nilai visitor dari y_train_pred  
y_train_pred_copies = np.repeat(y_train_pred, X_train.shape[-1], axis=-1)  
y_train_pred_real = scaler.inverse_transform(y_train_pred_copies)[:,1]  
y_train_pred_real
```

```
array([ 8273.22 ,  8679.375 ,  9202.714 ,  9426.193 ,  7788.382 ,  
    9876.476 ,  8684.852 ,  8339.211 ,  8190.3213,  8323.242 ,  
    9309.145 ,  8966.984 ,  9088.965 ,  9156.714 ,  9113.833 ,  
    9023.582 ,  8988.247 ,  8953.849 ,  8927.053 ,  8591.428 ,  
    8409.622 ,  8496.388 ,  9525.146 ,  9296.0285,  9338.456 ,  
    9351.369 ,  9315.116 ,  9194.731 ,  9185.538 ,  9166.173 ,  
    9148.978 ,  8808.687 ,  8643.908 ,  8731.873 ,  9825.218 ,  
    9460.183 ,  9465.551 ,  9597.052 ,  9468.054 ,  9761.437 ,  
    9359.757 ,  9330.122 ,  9444.235 ,  9218.02 ,  9225.226 ,  
    9658.859 ,  10542.952 ,  9936.602 ,  10884.443 ,  10489.54 ,  
    10494.519 ,  10532.841 ,  10295.889 ,  10087.956 ,  9714.861 ,  
    10462.062 ,  9491.746 ,  10532.781 ,  11617.154 ,  10752.536 ,  
    12959.287 ,  13200.645 ,  13290.911 ,  12981.416 ,  13269.153 ,  
    13213.521 ,  13289.073 ,  13239.247 ,  13391.38 ,  13298.33 ,  
    13369.17 ], dtype=float32)
```

Gambar 95. Inverse RNN

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
[ ] # Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_70_train = mean_squared_error(y_train_real, y_train_pred_real)
mae_70_train = mean_absolute_error(y_train_real, y_train_pred_real)
mape_70_train = mean_absolute_percentage_error(y_train_real, y_train_pred_real)

print(f"Mean Absolute Error (MAE): {round(mae_70_train, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_70_train, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_70_train, 2)}%")

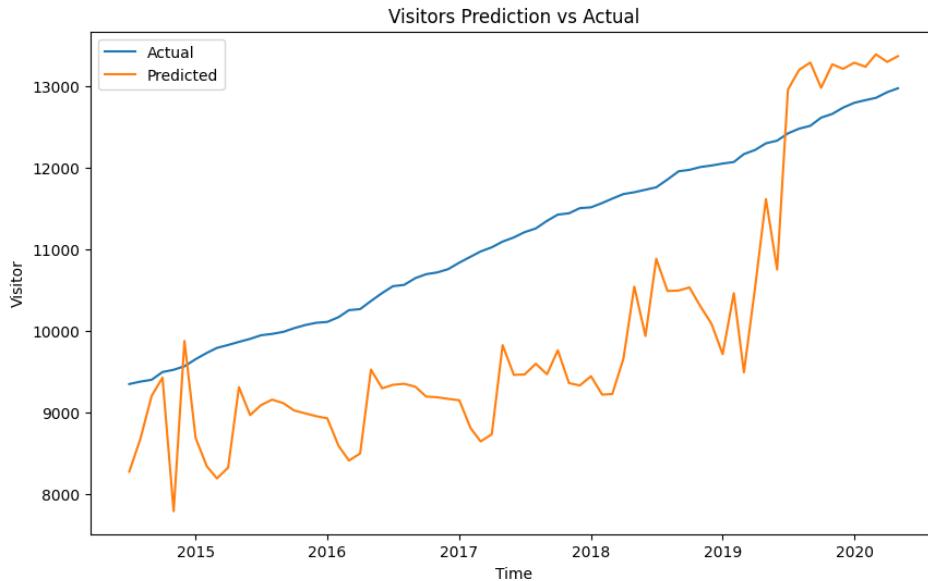
Mean Absolute Error (MAE): 1309.78
Mean Squared Error (MSE): 2100317.08
Mean Absolute Percentage Error (MAPE): 0.12%
```

Gambar 96. Hasil MSE, MAE, dan MAPE RNN

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model RNN mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model RNN dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.

```
[67] # Melakukan plotting hasil prediksi visitor terhadap data visitor asli dari train set
plt.figure(figsize(10, 6))
plt.plot(ntt.index[sequence_length:y_train.shape[0]+sequence_length], y_train_real, label='Actual')
plt.plot(ntt.index[sequence_length:y_train.shape[0]+sequence_length], y_train_pred_real, label='Predicted')
plt.title('Visitors Prediction vs Actual')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()
```



Gambar 97. Grafik Perbandingan RNN

g. Evaluate Model (Test)

Selanjutnya yaitu tahap prediksi pada model data test, dimana data test yang akan diprediksi yaitu X_test, dilanjut dengan penerapan fungsi scaler Inverse untuk mengembalikan nilai-nilai total_visitor dari y_test hingga memplot hasil Test Set.

```
[73] # Membuat prediksi visitor menggunakan X_test
y_test_pred = model.predict(X_test)

1/1 [=====] - 0s 26ms/step

[74] # Mengembalikan skala nilai visitor dari y_test
y_test_copies = np.repeat(y_test.reshape(-1, 1), X_test.shape[-1], axis=-1)
y_test_real = scaler.inverse_transform(y_test_copies)[:,1]
y_test_real

array([13058., 13156., 13183., 13211., 13256., 13339., 13403., 13481.,
       13537., 13595., 13637., 13667., 13757., 13788., 13845., 13891.,
       13915., 13959., 14060., 14140., 14181., 14201., 14220., 14302.,
       14389., 14475., 14541., 14575., 14586., 14659., 14782.])

[75] # Mengembalikan skala nilai visitor dari y_train_pred
y_test_pred_copies = np.repeat(y_test_pred, X_test.shape[-1], axis=-1)
y_test_pred_real = scaler.inverse_transform(y_test_pred_copies)[:,1]
y_test_pred_real

array([13254.54 , 13194.344 , 13555.644 , 13657.765 , 14031.499 ,
       14039.997 , 14116.811 , 14195.72 , 14191.3955, 14184.568 ,
       14190.635 , 14230.611 , 14262.147 , 14304.349 , 14341.98 ,
       14362.981 , 14366.611 , 14380.096 , 14406.96 , 14428.776 ,
       14471.005 , 14471.926 , 14443.922 , 14445.253 , 14511.974 ,
       14572.8125, 14597.916 , 14580.94 , 14572.333 , 14557.693 ,
       14556.163 ], dtype=float32)
```

Gambar 98.

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
[76] # Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_70_test = mean_squared_error(y_test_real, y_test_pred_real)
mae_70_test = mean_absolute_error(y_test_real, y_test_pred_real)
mape_70_test = mean_absolute_percentage_error(y_test_real, y_test_pred_real)

print(f"Mean Absolute Error (MAE): {round(mae_70_test, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_70_test, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_70_test, 2)}%")

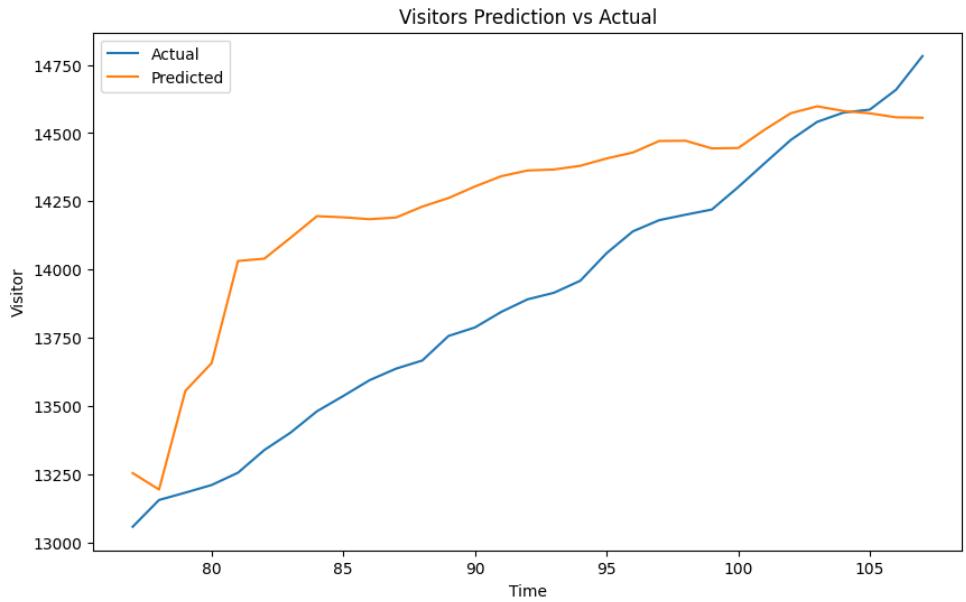
Mean Absolute Error (MAE): 366.84
Mean Squared Error (MSE): 186641.13
Mean Absolute Percentage Error (MAPE): 0.03%
```

Gambar 99. Evaluasi Model RNN

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model RNN mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model RNN dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.

```
❶ # Melakukan plotting hasil prediksi visitor terhadap data visitor asli dari test set
plt.figure(figsize=(10, 6))
plt.plot(ntt.index[-y_test.shape[0]:], y_test_real, label='Actual')
plt.plot(ntt.index[-y_test.shape[0]:], y_test_pred_real, label='Predicted')
plt.title('Visitors Prediction vs Actual')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()
```



Gambar 100.

- Skenario 60 % Train

- a. Splitting Train and Test Data

Membagi data pelatihan dan pengujian dengan rasio 70% dan 30% serta menetapkan hasilnya ke dalam variabel baru, seperti yang ditunjukkan di bawah ini:

```
[83] ntt = ntt.set_index('datetime')

[84] train_ratio = 0.6

[85] train_size = int(train_ratio * len(sequences))
    # pakai panjang sequences karena panjang dataset sudah berkurang 5 karena dibuat sequences

    X_train, X_test = sequences[:train_size], sequences[train_size:]
    y_train, y_test = labels[:train_size].reshape(-1,1), labels[train_size:].reshape(-1,1)

    print("Train X shape:", X_train.shape)
    print("Train Y shape:", y_train.shape)
    print("Test X shape:", X_test.shape)
    print("Test Y shape:", y_test.shape)

Train X shape: (61, 6, 2)
Train Y shape: (61, 1)
Test X shape: (41, 6, 2)
Test Y shape: (41, 1)
```

Gambar 101. Splitting train & Test RNN 70 %

Dalam penjelasan di atas, yaitu menghitung ukuran data pelatihan berdasarkan proporsi tertentu dari total urutan. Selanjutnya, menggunakan nilai tersebut untuk memisahkan urutan menjadi data pelatihan dan pengujian yang menjadi variabel baru Variabel X_train, X_test, y_train, dan y_test kemudian digunakan untuk menyimpan data dan label yang sesuai. Lalu, melakukan print untuk menampilkan ukuran dari data pelatihan dan pengujian untuk memastikan proses pemisahan berjalan dengan baik.

- b. Create Model

Tahap Selanjutnya, yaitu memanggil model RNN yang diimpor dari library tensorflow untuk membuat kerangka kerja model RNN dengan

menggunakan beberapa parameter yang telah ditentukan pada kode berikut.

```
[86] # Inisiasi pembuatan model
model = Sequential()

# Membuat model RNN
# Activation ada relu, sigmoid, tanh
model.add(SimpleRNN(40,activation="tanh",return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.15))
model.add(SimpleRNN(40,activation="tanh",return_sequences=True))
model.add(Dropout(0.15))
model.add(SimpleRNN(40,activation="tanh",return_sequences=False))
model.add(Dropout(0.15))
model.add(Dense(1))
```

Gambar 102. Splitting train & Test RNN 80 %

Kode tersebut mengawali proses pembuatan model RNN dengan menggunakan framework Keras. Model ini dirancang untuk melakukan prediksi pada suatu rangkaian data time series. Berikut adalah tahap-tahap konfigurasi model, yaitu yang pertama menambahkan Layer RNN Pertama, Menambahkan layer RNN pertama dengan 40 unit dan fungsi aktivasi tangen hiperbolik (tanh). `return_sequences=True` menunjukkan bahwa layer ini mengembalikan urutan sekuensial. `input_shape=(X_train.shape[1], X_train.shape[2])` menentukan bentuk input model berdasarkan dimensi data pelatihan.

Untuk yang kedua, menambahkan Dropout Layer Pertama untuk menambahkan layer dropout pertama dengan tingkat dropout sebesar 0.15. Dropout digunakan untuk mencegah overfitting dengan secara acak mengabaikan sebagian unit selama pelatihan. Lalu Tambahkan Layer RNN Kedua untuk menambahkan layer RNN kedua dengan 40 unit dan fungsi aktivasi tanh. `return_sequences=True` menunjukkan bahwa layer ini juga mengembalikan urutan sekuensial. Keempat untuk menambahkan Dropout Layer Kedua dengan tingkat dropout sebesar 0.15.

Lalu ada Model Summary yang ditunjukkan pada gambar dibawah ini yang berfungsi sebagai rekapan hasil inisiasi diatas.

```
[87] model.summary()

Model: "sequential_2"
-----

| Layer (type)             | Output Shape  | Param # |
|--------------------------|---------------|---------|
| simple_rnn_6 (SimpleRNN) | (None, 6, 40) | 1720    |
| dropout_6 (Dropout)      | (None, 6, 40) | 0       |
| simple_rnn_7 (SimpleRNN) | (None, 6, 40) | 3240    |
| dropout_7 (Dropout)      | (None, 6, 40) | 0       |
| simple_rnn_8 (SimpleRNN) | (None, 40)    | 3240    |
| dropout_8 (Dropout)      | (None, 40)    | 0       |
| dense_2 (Dense)          | (None, 1)     | 41      |


-----
Total params: 8241 (32.19 KB)
Trainable params: 8241 (32.19 KB)
Non-trainable params: 0 (0.00 Byte)
```

Gambar 103. model Summary RNN 70%

c. Fit Model

Proses `fit` pada model RNN merupakan tahap pelatihan, di mana model secara iteratif memperbarui parameter-parameter internalnya agar dapat

memahami pola dan hubungan dalam data pelatihan. Fungsi ini memanfaatkan algoritma pembelajaran yang diatur oleh optimizer, yang dalam kasus ini digunakan adalah algoritma Adam. Saat proses pelatihan, model mencoba meminimalkan nilai fungsi kerugian (loss function), yang mengukur seberapa baik prediksi model sesuai dengan nilai aktual. Dropout layers, yang ditambahkan pada model, membantu mencegah overfitting dengan secara acak mengabaikan sejumlah unit selama pelatihan. Setelah iterasi pelatihan selesai, model akan dapat membuat prediksi yang lebih akurat pada data yang tidak pernah dilihat sebelumnya. Keseluruhan proses ini bertujuan untuk menghasilkan model RNN yang dapat memahami dan memprediksi pola dalam data time series dengan lebih baik.

```
[ ] # Train model
# Pilihan optimizer adam, huber, spatial_crossentropy
model.compile(optimizer="adam", loss="MSE")
history = model.fit(X_train, y_train, epochs=10, batch_size=1000, verbose=2, validation_split=0.2)

Epoch 1/10
1/1 - 4s - loss: 0.2094 - val_loss: 0.0269 - 4s/epoch - 4s/step
Epoch 2/10
1/1 - 0s - loss: 0.4274 - val_loss: 0.0165 - 68ms/epoch - 68ms/step
Epoch 3/10
1/1 - 0s - loss: 0.3015 - val_loss: 0.0148 - 44ms/epoch - 44ms/step
Epoch 4/10
1/1 - 0s - loss: 0.2220 - val_loss: 0.1061 - 48ms/epoch - 48ms/step
Epoch 5/10
1/1 - 0s - loss: 0.2193 - val_loss: 0.2444 - 46ms/epoch - 46ms/step
Epoch 6/10
1/1 - 0s - loss: 0.2675 - val_loss: 0.2484 - 44ms/epoch - 44ms/step
Epoch 7/10
1/1 - 0s - loss: 0.2870 - val_loss: 0.1538 - 42ms/epoch - 42ms/step
Epoch 8/10
1/1 - 0s - loss: 0.1261 - val_loss: 0.0686 - 41ms/epoch - 41ms/step
Epoch 9/10
1/1 - 0s - loss: 0.2151 - val_loss: 0.0152 - 61ms/epoch - 61ms/step
Epoch 10/10
1/1 - 0s - loss: 0.1953 - val_loss: 0.0058 - 63ms/epoch - 63ms/step
```

Gambar 104. model Summary RNN

Untuk menunjukkan apakah model RNN memiliki kecocokan yang kuat dengan dataset dapat diukur dengan menggunakan r-squared seperti gambar dibawah ini, dimana r-squared 0.28 menunjukkan hubungan kuat dengan model

```
[90] # Evaluasi Nilai R^2
from sklearn.metrics import r2_score #ini nanti dipindah ke atas biar ngga bingung
score = r2_score(y_test,predictions)
print("R-Squared Score of RNN model",score)

R-Squared Score of RNN model -3.513042955295992
```

Gambar 105. model Summary RNN

Selanjutnya, yaitu Plot berikut akan menampilkan perbandingan antara data aktual dan prediksi berdasarkan model SVR dengan pembagian data training dan test yang telah dilakukan.

```
[3757] def plotting_actual_vs_pred(y_test, y_pred, title):
    plt.figure(figsize=(16, 4))
    plt.plot(y_test, color='blue', label='Actual')
    plt.plot(y_pred, alpha=0.7, color='orange',
    label='Predicted')
    plt.title(title)
    plt.xlabel('Time')
    plt.ylabel('Visitor')
    plt.legend()
    plt.show()

[3758] plotting_actual_vs_pred(y_test, predictions, "Predictions made by RNN model")
```

Gambar 106. Plot Predicted vs Actual RNN

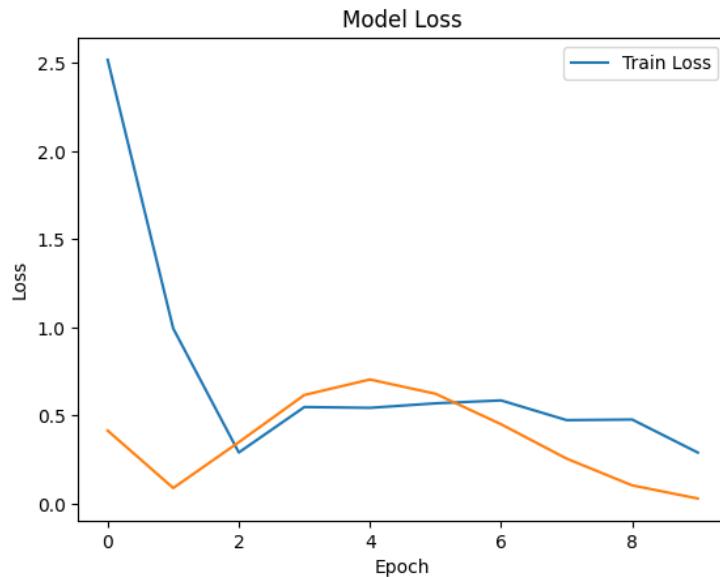
d. Loss Model

Tahap selanjutnya yaitu, mengenai Loss Model. Loss Model sendiri adalah fungsi matematis yang menyederhanakan perbedaan antara prediksi model dengan nilai sebenarnya. Sebagai contoh, dalam regresi, fungsi kerugian umumnya adalah Mean Squared Error (MSE), yang menghitung rata-rata dari kuadrat selisih antara prediksi dan ground truth. Proses pelatihan model melibatkan iteratif memperbarui parameter-parameter model untuk meminimalkan nilai loss ini. Sebagai penerapan loss model, maka dibuat seperti berikut, dimana loss model diambil dari Fit RNN diatas.

```
[3759] # Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train Loss', 'Validation'], loc='upper right')
plt.show()
```

Gambar 107. Loss Model

Berdasarkan plot pada gambar diatas, dapat diartikan sebagai pemberian label, legenda berdasarkan data history yang menghasilkan plot train loss dan val loss seperti berikut



Gambar 108. Hasil Loss Model

e. Prediction

Selanjutnya yaitu, melakukan prediksi terhadap model untuk X_{test} , dimana prediction sendiri dalam RNN melibatkan mengalirkan input melalui model dan memperoleh output prediksi. Hasil ini kemudian dapat dibandingkan dengan nilai sebenarnya untuk mengevaluasi seberapa baik model dapat melakukan prediksi. Pada umumnya, hasil prediksi RNN

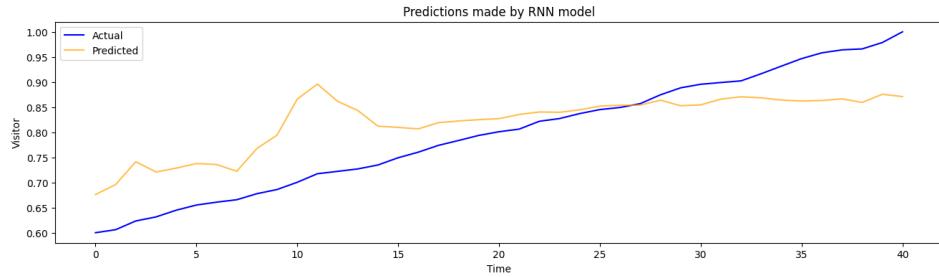
dapat digunakan untuk berbagai aplikasi, termasuk prediksi time series, pengenalan pola, dan tugas-tugas lain yang melibatkan urutan data.

```
[207] # Membuat prediksi visitor menggunakan X_train
y_train_pred = model.predict(X_train)

2/2 [=====] - 0s 8ms/step
```

Gambar 109. Fungsi Predict RNN

Berdasarkan gambar diatas, maka prediksi berhasil dijalankan dengan menggunakan fungsi predict dan menghasilkan plot seperti dibawah ini.



Gambar 110. Hasil Prediksi RNN

f. Evaluate Model (Train)

Selanjutnya yaitu tahap prediksi pada model data train, dimana data train yang akan diprediksi yaitu X_train, dilanjut dengan penerapan fungsi scaler Inverse untuk mengembalikan nilai-nilai total_visitor dari y_train hingga memplot hasil Train Set.

```
[207] # Membuat prediksi visitor menggunakan X_train
y_train_pred = model.predict(X_train)

2/2 [=====] - 0s 8ms/step

[208] # Mengembalikan skala nilai visitor dari y_train
y_train_copies = np.repeat(y_train.reshape(-1, 1), X_train.shape[-1], axis=-1)
y_train_real = scaler.inverse_transform(y_train_copies)[:,1]
y_train_real

array([ 9348.,  9378.,  9400.,  9495.,  9523.,  9566.,  9653.,  9730.,
       9791.,  9827.,  9865.,  9902.,  9946.,  9963.,  9988.,  10033.,
      10072.,  10099.,  10110.,  10168.,  10254.,  10267.,  10366.,  10464.,
      10548.,  10564.,  10647.,  10695.,  10717.,  10757.,  10837.,  10909.,
      10974.,  11025.,  11095.,  11146.,  11211.,  11256.,  11349.,  11426.,
      11442.,  11505.,  11515.,  11568.,  11622.,  11678.,  11700.,  11731.,
      11761.,  11856.,  11957.,  11975.,  12010.,  12028.,  12052.,  12071.,
     12168.,  12219.,  12300.,  12333.,  12422.])

❸ # Mengembalikan skala nilai visitor dari y_train_pred
y_train_pred_copies = np.repeat(y_train_pred, X_train.shape[-1], axis=-1)
y_train_pred_real = scaler.inverse_transform(y_train_pred_copies)[:,1]
y_train_pred_real

❹ array([10730.396 ,  9922.791 ,  9110.917 ,  10225.167 ,  12627.834 ,
       5357.427 , 10159.002 , 10424.137 , 10818.065 , 10794.123 ,
      10847.445 , 12802.395 , 10543.235 , 9771.161 , 9914.885 ,
      10283.713 , 10270.66 , 10251.479 , 10254.875 , 10550.881 ,
      10851.148 , 10793.681 , 10890.725 , 12657.516 , 10764.076 ,
      10070.019 , 10261.489 , 10502.929 , 10476.301 , 10473.978 ,
      10472.929 , 10721.199 , 10943.858 , 10859.238 , 11066.006 ,
      12656.287 , 11187.164 , 10616.571 , 11154.336 , 10914.3955,
      11463.553 , 10944.159 , 11071.06 , 11930.797 , 11943.315 ,
     12106.802 , 12591.424 , 13371.937 , 11647.671 , 11743.152 ,
     11868.539 , 11800. , 11959.321 , 11602.122 , 11743.609 ,
     12261.744 , 13674.763 , 13084.845 , 13849.134 , 14800.833 ,
     13195.602 ], dtype=float32)
```

Gambar 111. Inverse RNN

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
[210] # Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_60_train = mean_squared_error(y_train_real, y_train_pred_real)
mae_60_train = mean_absolute_error(y_train_real, y_train_pred_real)
mape_60_train = mean_absolute_percentage_error(y_train_real, y_train_pred_real)

print(f"Mean Absolute Error (MAE): {round(mae_60_train, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_60_train, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_60_train, 2)}%")

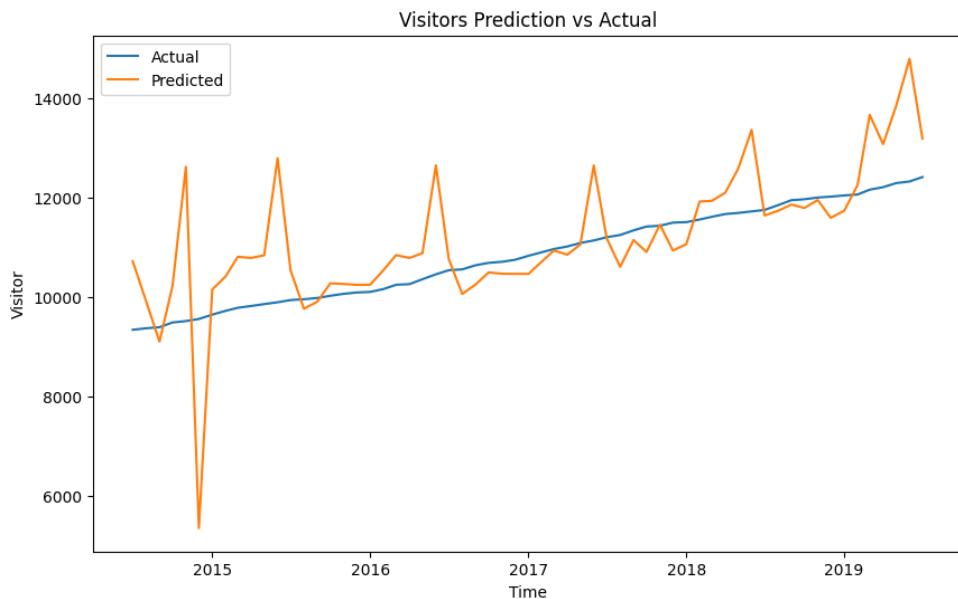
Mean Absolute Error (MAE): 687.52
Mean Squared Error (MSE): 1137744.26
Mean Absolute Percentage Error (MAPE): 0.07%
```

Gambar 112. Hasil MSE, MAE, dan MAPE RNN

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model RNN mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model RNN dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.

```
[67] # Melakukan plotting hasil prediksi visitor terhadap data visitor asli dari train set
plt.figure(figsize=(10, 6))
plt.plot(ntt.index[sequence_length:y_train.shape[0]+sequence_length], y_train_real, label='Actual')
plt.plot(ntt.index[sequence_length:y_train.shape[0]+sequence_length], y_train_pred_real, label='Predicted')
plt.title('Visitors Prediction vs Actual')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()
```



Gambar 113.

g. Evaluate Model (Test)

Selanjutnya yaitu tahap prediksi pada model data test, dimana data test yang akan diprediksi yaitu X_{test} , dilanjut dengan penerapan fungsi scaler

Inverse untuk mengembalikan nilai-nilai total_visitor dari y_test hingga memplot hasil Test Set.

```
[217] # Membuat prediksi visitor menggunakan X_test
y_test_pred = model.predict(X_test)

2/2 [=====] - 0s 8ms/step

[218] # Mengembalikan skala nilai visitor dari y_test
y_test_copies = np.repeat(y_test.reshape(-1, 1), X_test.shape[-1], axis=-1)
y_test_real = scaler.inverse_transform(y_test_copies)[:,1]
y_test_real

array([12480., 12515., 12615., 12661., 12738., 12797., 12830., 12858.,
       12927., 12975., 13058., 13156., 13183., 13211., 13256., 13339.,
       13463., 13481., 13537., 13595., 13637., 13667., 13757., 13788.,
       13845., 13891., 13915., 13959., 14060., 14140., 14181., 14201.,
       14220., 14302., 14389., 14475., 14541., 14575., 14586., 14659.,
       14782.])

[219] # Mengembalikan skala nilai visitor dari y_train_pred
y_test_pred_copies = np.repeat(y_test_pred, X_test.shape[-1], axis=-1)
y_test_pred_real = scaler.inverse_transform(y_test_pred_copies)[:,1]
y_test_pred_real

array([12917.18 , 13032.085, 13293.866, 13175.146, 13220.515, 13272.462,
       13261.897, 13183.789, 13445.145, 13598.144, 14011.33 , 14183.304,
       13985.191, 13880.741, 13701.238, 13686.82 , 13669.955, 13741.559,
       13760.292, 13775.405, 13787.012, 13834.863, 13863.626, 13859.704,
       13888.193, 13930.363, 13939.95 , 13943.113, 13998.251, 13935. ,
       13945.258, 14010.969, 14037.832, 14025.232, 14000.449, 13988.773,
       13994.691, 14013.755, 13972.651, 14066.173, 14040.045],
      dtype=float32)
```

Gambar 114.

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
[220] # Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_60_test = mean_squared_error(y_test_real, y_test_pred_real)
mae_60_test = mean_absolute_error(y_test_real, y_test_pred_real)
mape_60_test = mean_absolute_percentage_error(y_test_real, y_test_pred_real)

print(f"Mean Absolute Error (MAE): {round(mae_60_test, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_60_test, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_60_test, 2)}%")

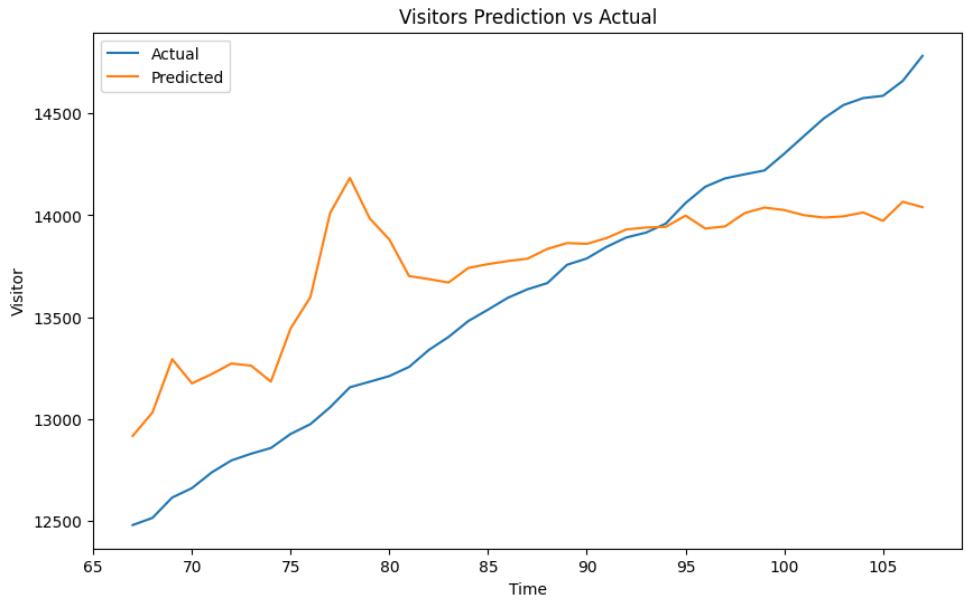
Mean Absolute Error (MAE): 387.38
Mean Squared Error (MSE): 214313.19
Mean Absolute Percentage Error (MAPE): 0.03%
```

Gambar 115.

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model RNN mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model RNN dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.

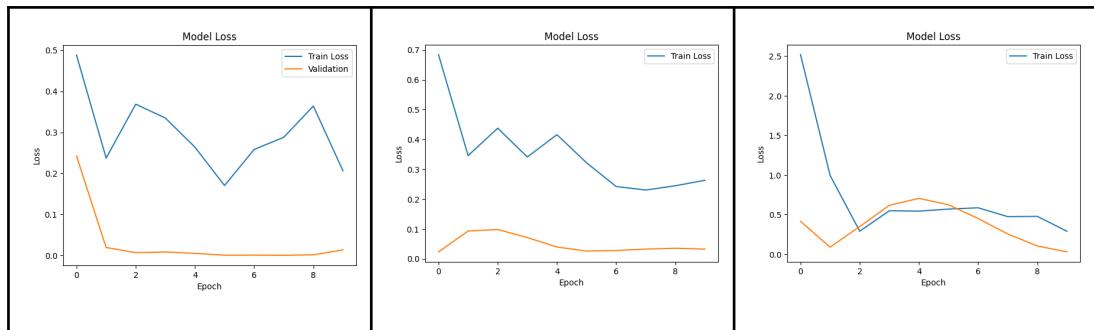
```
❸ # Melakukan plotting hasil prediksi visitor terhadap data visitor asli dari test set
plt.figure(figsize=(10, 6))
plt.plot(ntt.index[-y_test.shape[0]:], y_test_real, label='Actual')
plt.plot(ntt.index[-y_test.shape[0]:], y_test_pred_real, label='Predicted')
plt.title('Visitors Prediction vs Actual')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()
```



gambar 117.

Tabel Perbandingan Grafik Antar Skenario

Scenario 80% Train	Scenario 70% Train	Scenario 60% Train
Train (Prediction vs Actual) 	Train (Prediction vs Actual) 	Train (Prediction vs Actual)
Test (Prediction vs Actual) 	Test (Prediction vs Actual) 	Test (Prediction vs Actual)
Prediction vs Actual 	Prediction vs Actual 	Prediction vs Actual
Loss Model	Loss Model	Loss Model



Scenario	Model	MSE	MAE	MAPE (%)
Scenario 1	RNN Train 80	3.27025e+06	1738.07	0.15755
Scenario 1	RNN Test 80	590984	733.493	0.0521801
Scenario 2	RNN Train 70	567485	571.946	0.0514201
Scenario 2	RNN Test 70	145807	332.409	0.0242785
Scenario 3	RNN Train 60	1.13774e+06	687.524	0.0651035
Scenario 3	RNN Test 60	214313	387.378	0.0287745

MAPE sering kali efektif untuk menganalisis kumpulan data yang besar dan membutuhkan penggunaan nilai kumpulan data selain nol. Nilai MAPE lebih mudah diinterpretasikan dibandingkan alat ukur yang lain karena berupa nilai persen yang dapat terukur skalanya. Berdasarkan nilai MAPE, skenario terbaik pada model RNN adalah skenario 2 dengan pembagian 70% data latih (train) dan 30% data uji (test).

3. LSTM

Setelah melakukan pra proses dan normalisasi data pada model RNN, proses yang dilakukan selanjutnya adalah melakukan pembagian data *train* dan *test* berdasarkan ketiga skenario. Kemudian, proses berlanjut dengan pembuatan model LSTM yang menyesuaikan masing-masing skenario.

- Skenario 80 % Train
 - a. Splitting Train and Test Data

Hal pertama yang dilakukan sebelum membangun model LSTM adalah membagi keseluruhan data menjadi *training* dan *testing* data. Pembagian ini disesuaikan kondisi skenario yang telah ditentukan. Menurut kode yang tertera pada gambar terlampir, ditentukan bahwa data yang digunakan sebagai data *training* berjumlah 0,8 kali (80%) keseluruhan jumlah data, sedangkan sisanya merupakan data *testing*. Sumber data pada pembagian ini adalah ‘ntt_scaled’ yang merupakan data hasil normalisasi. Hasil menunjukkan bahwa jumlah baris pada data *training* dan *testing* masing-masing sebanyak 86 dan 22 baris.

```

# Split the data into training and testing sets
ntt_val = ntt_lstm.values
train_lstm80 = int(len(ntt_val) * 0.8)
test_lstm20 = len(ntt_val) - train_lstm80
train80, test20 = ntt_scaled[0:train_lstm80,:], ntt_scaled[train_lstm80:len(ntt_val),:]

print(len(train80), len(test20))

```

86 22

Gambar 118. Coding dan Hasil Splitting Data

Setelah itu, proses pada langkah ini dilanjutkan dengan pembuatan *time series* berdasarkan panjang sekuens yang telah ditentukan. Untuk bagian ini, penjelasan terdapat pada sub bab ‘Time Series (Sequence)’ dalam bab ketiga laporan ini.

```

# convert an array of values into a dataset matrix
def create_dataset(data, look_back):
    dataX, dataY = [], []
    for i in range(len(data)-look_back):
        dataX.append(data[i:(i+look_back), 0:data.shape[1]])
        dataY.append(data[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

# reshape into X=t and Y=t+1
look_back = 10
X_train80, y_train80 = create_dataset(train80, look_back)
X_test20, y_test20 = create_dataset(test20, look_back)

```

Gambar 119. Coding Pembuatan Time Series

Proses berlanjut dengan menyesuaikan kembali dimensi data pada masing-masing data *train* dan *test*. Untuk setiap data dengan variabel X harus memiliki bentuk yang sama, begitu juga dengan variabel y. Selain itu, jumlah data yang digunakan pada masing-masing data *train* dan *test* harus sama, yaitu sesuai dengan jumlah data pada saat pembagian sebelumnya.

```

# Reshape the output (y) to ensure it is a 2D array
y_train80 = y_train80.reshape(-1, 1)
y_test20 = y_test20.reshape(-1, 1)

# reshape input to be [samples, time steps, features]
X_train80 = np.reshape(X_train80, (X_train80.shape[0], 2, X_train80.shape[1]))
X_test20 = np.reshape(X_test20, (X_test20.shape[0], 2, X_test20.shape[1]))

print("Train X shape:", X_train80.shape)
print("Train Y shape:", y_train80.shape)
print("Test X shape:", X_test20.shape)
print("Test Y shape:", y_test20.shape)

Train X shape: (76, 2, 10)
Train Y shape: (76, 1)
Test X shape: (12, 2, 10)
Test Y shape: (12, 1)

```

Gambar 120. Coding dan Hasil Reshape Dataset

Gambar di atas menunjukkan hasil akhir dimensi pada masing-masing data *train* dan *test* pada masing-masing variabel.

b. Create LSTM Model

Langkah kedua yang dilakukan adalah pembangunan model LSTM untuk skenario ini. Pembuatan model dilakukan menggunakan fungsi

‘Sequential’ pada metode keras milik *library* ‘Tensorflow’ yang telah di-*import* pada bagian awal penggerjaan. Setelah itu, ditetapkan parameter dan *input* yang akan digunakan pada model LSTM pada *layer* pertama dan kedua. Setelah menetapkan *layer* keluaran, disebutkan fungsi *compile* untuk menjalankan model LSTM tersebut.

```
# Build the LSTM model
lstm80_model = Sequential()

# First LSTM layer
lstm80_model.add(LSTM(100, activation='relu', return_sequences=True, input_shape=(X_train80.shape[1], look_back)))

# Second LSTM layer
lstm80_model.add(LSTM(100, activation='relu'))

# Output layer
lstm80_model.add(Dense(1)) # Assuming you want to predict 1 feature (visitor)

lstm80_model.compile(optimizer='adam', loss='mse')
```

Gambar 121. Coding Pembuatan Model LSTM Skenario 1

```
lstm80_model.summary()

Model: "sequential_18"

-----  
Layer (type)          Output Shape         Param #
-----  
lstm_36 (LSTM)        (None, 2, 100)      44400  
lstm_37 (LSTM)        (None, 100)       80400  
dense_18 (Dense)      (None, 1)           101  
-----  
Total params: 124901 (487.89 KB)  
Trainable params: 124901 (487.89 KB)  
Non-trainable params: 0 (0.00 Byte)
```

Gambar 122. Ringkasan Model LSTM

Gambar di atas merupakan *summary* dari model LSTM yang telah dibuat.

c. Training LSTM Model

Langkah berikutnya adalah melakukan *training* terhadap model LSTM yang telah dibuat. Berikut kode yang digunakan.

```

# Train the model
history80 = lstm80_model.fit(
    X_train80, y_train80,
    epochs=100,
    batch_size=32,
    verbose=2,
    validation_split=0.2, # Use part of the training data as validation
)

Epoch 1/100
2/2 - 3s - loss: 0.3854 - val_loss: 0.0058 - 3s/epoch - 2s/step
Epoch 2/100
2/2 - 0s - loss: 0.3594 - val_loss: 0.0053 - 42ms/epoch - 21ms/step
Epoch 3/100
2/2 - 0s - loss: 0.3355 - val_loss: 0.0056 - 38ms/epoch - 19ms/step
Epoch 4/100
2/2 - 0s - loss: 0.3116 - val_loss: 0.0069 - 43ms/epoch - 21ms/step
Epoch 5/100
2/2 - 0s - loss: 0.2860 - val_loss: 0.0096 - 42ms/epoch - 21ms/step
Epoch 6/100
2/2 - 0s - loss: 0.2580 - val_loss: 0.0143 - 48ms/epoch - 24ms/step
Epoch 7/100
2/2 - 0s - loss: 0.2302 - val_loss: 0.0219 - 43ms/epoch - 21ms/step
Epoch 8/100
2/2 - 0s - loss: 0.1985 - val_loss: 0.0338 - 44ms/epoch - 22ms/step
Epoch 9/100
2/2 - 0s - loss: 0.1652 - val_loss: 0.0518 - 44ms/epoch - 22ms/step
Epoch 10/100
2/2 - 0s - loss: 0.1352 - val_loss: 0.0783 - 43ms/epoch - 21ms/step
Epoch 11/100

```

Gambar 123. Coding Training Model LSTM

Fungsi yang digunakan adalah *fitting model* terhadap model yang sudah ada. Data yang digunakan pada langkah ini adalah ‘X_train80’ yang berisi fitur data yang memengaruhi prediksi dan ‘y_train80’ yang merupakan data target prediksi. Setelah itu, parameter *epochs* menentukan jumlah iterasi pemrosesan data oleh model LSTM. Dengan menggunakan parameter *batch_size*, ditentukan bahwa setiap iterasi akan mengambil sampel data sejumlah 32. Parameter *verbose* akan mengontrol sejauh mana informasi pelatihan akan ditampilkan selama proses pelatihan. Parameter terakhir adalah *validation_split* yang menentukan sebagian data pelatihan yang akan diambil untuk validasi. Dalam hal ini, 20% dari data pelatihan akan diambil sebagai data validasi untuk memantau kinerja model selama pelatihan. Besaran parameter tersebut bersesuaian dengan jumlah data *test* yang digunakan pada skenario ini.

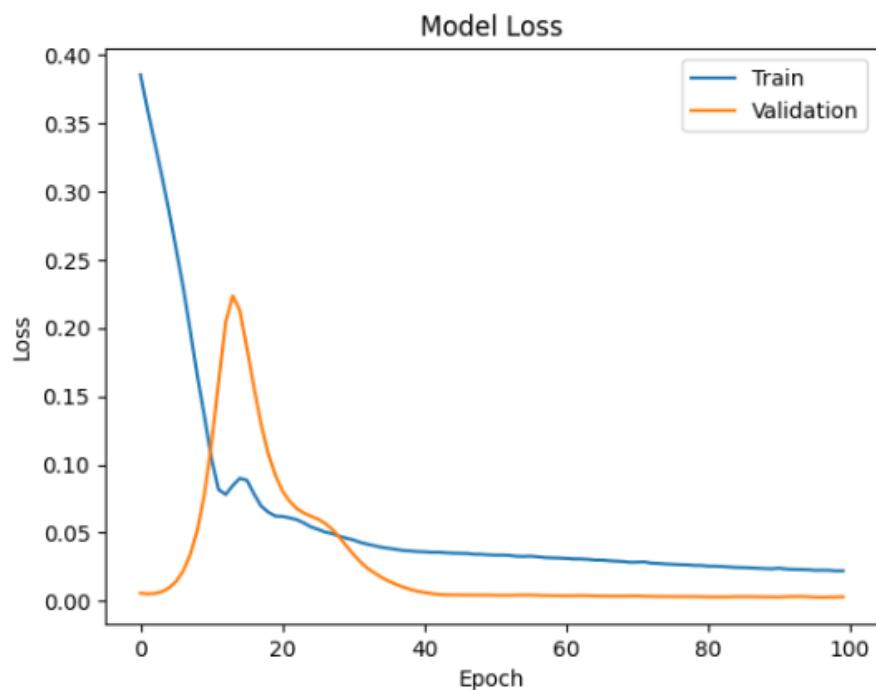
d. Plotting Training-Validation Loss

Langkah ini digunakan untuk membandingkan *data loss* pada data *training* dan *validation* hasil dari model LSTM.

```

# Plot training & validation loss values
plt.plot(history80.history['loss'])
plt.plot(history80.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

```



Gambar 124. Training-Validation Loss Plot LSTM

Berdasarkan grafik, diketahui bahwa model masih belum merupakan model terbaik karena bentuk grafik *validation* tidak begitu sesuai dengan bentuk grafik *training*.

e. Make Prediction on the Test Set

Menggunakan model LSTM yang sudah ada, dilakukan prediksi terhadap data ‘X_test20’ sebagai berikut.

```

# Make predictions on the test set
y_pred20_scaled = lstm80_model.predict(X_test20)

print("Shape Hasil Prediksi:", y_pred20_scaled.shape)
print("Hasil Prediksi:", y_pred20_scaled)

```

1/1 [=====] - 0s 427ms/step
Shape Hasil Prediksi: (12, 1)
Hasil Prediksi: [[3.6435202e-03]
[1.9461866e-03]
[-1.6503036e-05]
[-1.6399100e-03]
[-2.6085600e-03]
[-3.3321828e-03]
[-3.8944930e-03]
[-6.2974058e-03]
[-6.8075880e-03]
[-8.2224980e-03]
[-8.5746124e-03]
[-9.3411952e-03]]

Gambar 125. Hasil Prediksi Model LSTM

Berdasarkan kode tersebut, diketahui bahwa hasil prediksi berupa array dengan *value* dalam skala normalisasi dengan dimensi data berjumlah 12 baris.

f. Inverse Transform Dataset

Karena data saat ini dalam skala hasil normalisasi, maka perlu dilakukan denormalisasi data untuk mengembalikannya dalam skala aktual. Untuk itu, digunakan fungsi ‘inverse_transform’ pada data yang perlu diberlakukan denormalisasi.

```

y_pred20_sc = np.repeat(y_pred20_scaled,2, axis=-1)
y_pred20 = scaler_lstm.inverse_transform(np.reshape(y_pred20_sc,(len(y_pred20_scaled),2))[:,0])

y_test20_sc = np.repeat(y_test20,2, axis=-1)
y_test20_re = scaler_lstm.inverse_transform(np.reshape(y_test20_sc,(len(y_test20),2))[:,0])

print("Original Test Shape:", y_test20.shape)
print("Transformed Test Shape:", y_test20_re.shape)
print("Predicted Shape:", y_pred20.shape)

Original Test Shape: (12, 1)
Transformed Test Shape: (12,)
Predicted Shape: (12,)

# print("Original Test Subset:", y_test20)
print("Transformed Test Subset:", y_test20_re)
print("Predicted Subset:", y_pred20)

Transformed Test Subset: [ 234.    226.    317.    737.   1203.   1792.   1149.   3120.   2672.   2646.   2511.   4300.]
Predicted Subset: [ 235.5135     125.79956    -1.0667398   -106.002144   -168.61472
-215.38896   -251.73613    -407.058     -440.03568    -531.494
-554.2544    -603.80554   ]

```

Gambar 126. Coding dan Hasil Invers Data

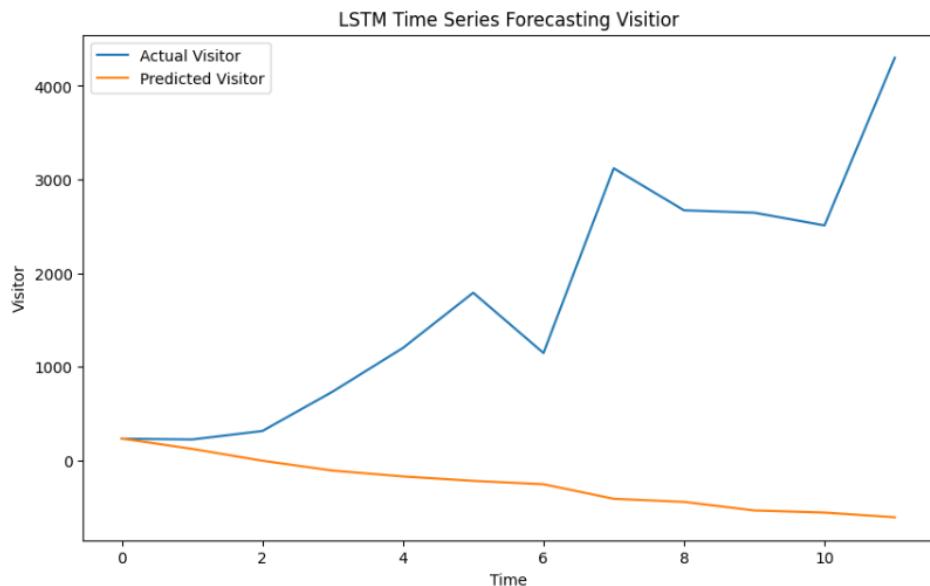
Berdasarkan kode tersebut, dilakukan juga *reshape* dimensi data menyesuaikan dimensi data aktual supaya proses invers berjalan sesuai dengan yang diinginkan. Terakhir, dimunculkan juga data tes dan hasil prediksi dalam skala aktual.

g. Plotting Actual-Prediction

Untuk mengetahui efektivitas model, diperlukan grafik untuk *plotting* perbandingan antara data aktual dan hasil *training*.

```
# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(y_test20_re, label='Actual Visitor')
plt.plot(y_pred20, label='Predicted Visitor')
plt.title('LSTM Time Series Forecasting Visitior')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()

plt.show()
```



Gambar 127. Plot Data Aktual dan Hasil Prediksi LSTM

Karena hasil prediksi bernilai negatif, maka terlihat bahwa hasil grafik sangat tidak bersesuaian. Oleh karena itu, diperlukan evaluasi model untuk mengetahui tingkat optimasi dari model LSTM.

h. Evaluate LSTM Model

Langkah terakhir pada eksperimen model LSTM adalah melakukan evaluasi terhadap hasil prediksi. Evaluasi model dilakukan menggunakan perhitungan terhadap fungsi MSE, MAE, dan MAPE.

```

# Calculate MSE, RMSE, MAPE
mse_test1 = mean_squared_error(y_test20_re, y_pred20).round(2)
mae_test1 = mean_absolute_error(y_test20_re, y_pred20).round(2)
mape_test1 = mean_absolute_percentage_error(y_test20_re, y_pred20).round(2)

print(f"Mean Squared Error (MSE): {mse_test1}")
print(f"Mean Absolute Error (MAE): {mae_test1}")
print(f"Mean Absolute Percetange Error (MAPE): {mape_test1} %")

```

Mean Squared Error (MSE): 6196610.53
 Mean Absolute Error (MAE): 1985.68
 Mean Absolute Percetange Error (MAPE): 0.99 %

Gambar 128. Evaluasi Model LSTM

Penentu nilai dari model ini adalah MAPE yang bersifat *general* karena nilainya berupa persentase. Evaluasi untuk model LSTM pada skenario ini adalah nilai MAPE sebesar 0,99 persen.

```

# Membuat prediksi visitor menggunakan X_train
y_pred80 = lstm80_model.predict(X_train80)

3/3 [=====] - 0s 5ms/step

# Calculate MSE, RMSE, MAPE
mse_train1 = mean_squared_error(y_train80, y_pred80).round(2)
mae_train1 = mean_absolute_error(y_train80, y_pred80).round(2)
mape_train1 = mean_absolute_percentage_error(y_train80, y_pred80).round(2)

print(f"Mean Squared Error (MSE): {mse_train1}")
print(f"Mean Absolute Error (MAE): {mae_train1}")
print(f"Mean Absolute Percetange Error (MAPE): {mape_train1} %")

```

Mean Squared Error (MSE): 0.02
 Mean Absolute Error (MAE): 0.1
 Mean Absolute Percetange Error (MAPE): 13026066045727.44 %

Gambar 129. Evaluasi Model Train LSTM

Hal yang sama juga dilakukan terhadap prediksi pada data *train*.

- Skenario 70 % Train

- a. Splitting Train and Test Data

Hal pertama yang dilakukan sebelum membangun model LSTM adalah membagi keseluruhan data menjadi *training* dan *testing* data. Pembagian ini disesuaikan kondisi skenario yang telah ditentukan. Menurut kode yang tertera pada gambar terlampir, ditentukan bahwa data yang digunakan sebagai data *training* berjumlah 0,7 kali (70%) keseluruhan jumlah data, sedangkan sisanya merupakan data *testing*. Sumber data pada pembagian ini adalah ‘ntt_scaled’ yang merupakan data hasil normalisasi. Hasil menunjukkan bahwa jumlah baris pada data *training* dan *testing* masing-masing sebanyak 75 dan 33 baris.

```

# Split the data into training and testing sets
ntt_val = ntt_lstm.values
train_lstm70 = int(len(ntt_val) * 0.7)
test_lstm30 = len(ntt_val) - train_lstm70
train70, test30 = ntt_scaled[0:train_lstm70,:], ntt_scaled[train_lstm70:len(ntt_val),:]

print(len(train70), len(test30))

```

75 33

Gambar 130. Coding dan Hasil Splitting Data

Setelah itu, proses pada langkah ini dilanjutkan dengan pembuatan *time series* berdasarkan panjang sekuens yang telah ditentukan. Untuk bagian ini, penjelasan terdapat pada sub bab ‘Time Series (Sequence)’ dalam bab ketiga laporan ini.

```
# convert an array of values into a dataset matrix
def create_dataset(data, look_back):
    dataX, dataY = [], []
    for i in range(len(data)-look_back-1):
        dataX.append(data[i:(i+look_back), 0:data.shape[1]])
        dataY.append(data[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

# reshape into X=t and Y=t+1
look_back = 12
X_train70, y_train70 = create_dataset(train70, look_back)
X_test30, y_test30 = create_dataset(test30, look_back)
```

Gambar 131. Coding Pembuatan Time Series

Proses berlanjut dengan menyesuaikan kembali dimensi data pada masing-masing data *train* dan *test*. Untuk setiap data dengan variabel X harus memiliki bentuk yang sama, begitu juga dengan variabel y. Selain itu, jumlah data yang digunakan pada masing-masing data *train* dan *test* harus sama, yaitu sesuai dengan jumlah data pada saat pembagian sebelumnya.

```
# Reshape the output (y) to ensure it is a 2D array
y_train70 = y_train70.reshape(-1, 1)
y_test30 = y_test30.reshape(-1, 1)

# reshape input to be [samples, time steps, features]
X_train70 = np.reshape(X_train70, (X_train70.shape[0], 2, X_train70.shape[1]))
X_test30 = np.reshape(X_test30, (X_test30.shape[0], 2, X_test30.shape[1]))

print("Train X shape:", X_train70.shape)
print("Train Y shape:", y_train70.shape)
print("Test X shape:", X_test30.shape)
print("Test Y shape:", y_test30.shape)

Train X shape: (62, 2, 12)
Train Y shape: (62, 1)
Test X shape: (20, 2, 12)
Test Y shape: (20, 1)
```

Gambar 132. Coding dan Hasil Reshape Dataset

Gambar di atas menunjukkan hasil akhir dimensi pada masing-masing data *train* dan *test* pada masing-masing variabel.

b. Create LSTM Model

Langkah kedua yang dilakukan adalah pembangunan model LSTM untuk skenario ini. Pembuatan model dilakukan menggunakan fungsi ‘Sequential’ pada metode keras milik *library* ‘Tensorflow’ yang telah di-import pada bagian awal penggerjaan. Setelah itu, ditetapkan parameter dan *input* yang akan digunakan pada model LSTM pada *layer* pertama dan kedua. Setelah menetapkan *layer* keluaran, disebutkan fungsi *compile* untuk menjalankan model LSTM tersebut.

```

# Build the LSTM model
lstm70_model = Sequential()

# First LSTM layer
lstm70_model.add(LSTM(100, activation='relu', return_sequences=True, input_shape=(X_train70.shape[1], look_back)))

# Second LSTM layer
lstm70_model.add(LSTM(100, activation='relu'))

# Output layer
lstm70_model.add(Dense(1)) # Assuming you want to predict 1 feature (visitor)

# optimizer = Adam(learning_rate=0.001)
lstm70_model.compile(optimizer='adam', loss='mse')

```

Gambar 133. Coding Pembuatan Model LSTM Skenario 2

```

lstm70_model.summary()

Model: "sequential_19"

```

Layer (type)	Output Shape	Param #
lstm_38 (LSTM)	(None, 2, 100)	45200
lstm_39 (LSTM)	(None, 100)	80400
dense_19 (Dense)	(None, 1)	101

```

Total params: 125701 (491.02 KB)
Trainable params: 125701 (491.02 KB)
Non-trainable params: 0 (0.00 Byte)

```

Gambar 134. Ringkasan Model LSTM

Gambar di atas merupakan *summary* dari model LSTM yang telah dibuat.

c. Training LSTM Model

Langkah berikutnya adalah melakukan *training* terhadap model LSTM yang telah dibuat. Berikut kode yang digunakan.

```

# Train the model
history70 = lstm70_model.fit(
    X_train70, y_train70,
    epochs=100,
    batch_size=32,
    verbose=2,
    validation_split=0.3, # Use part of the training data as validation
)

Epoch 1/100
2/2 - 3s - loss: 0.4570 - val_loss: 0.1091 - 3s/epoch - 2s/step
Epoch 2/100
2/2 - 0s - loss: 0.4327 - val_loss: 0.1008 - 41ms/epoch - 20ms/step
Epoch 3/100
2/2 - 0s - loss: 0.4099 - val_loss: 0.0928 - 39ms/epoch - 20ms/step
Epoch 4/100
2/2 - 0s - loss: 0.3866 - val_loss: 0.0846 - 40ms/epoch - 20ms/step
Epoch 5/100
2/2 - 0s - loss: 0.3601 - val_loss: 0.0759 - 39ms/epoch - 19ms/step
Epoch 6/100
2/2 - 0s - loss: 0.3307 - val_loss: 0.0668 - 41ms/epoch - 21ms/step
Epoch 7/100
2/2 - 0s - loss: 0.2978 - val_loss: 0.0576 - 39ms/epoch - 20ms/step
Epoch 8/100
2/2 - 0s - loss: 0.2597 - val_loss: 0.0492 - 43ms/epoch - 22ms/step
Epoch 9/100
2/2 - 0s - loss: 0.2185 - val_loss: 0.0429 - 40ms/epoch - 20ms/step
Epoch 10/100
2/2 - 0s - loss: 0.1728 - val_loss: 0.0411 - 41ms/epoch - 21ms/step
Epoch 11/100

```

Gambar 135. Coding Training Model LSTM

Fungsi yang digunakan adalah *fitting model* terhadap model yang sudah ada. Data yang digunakan pada langkah ini adalah ‘X_train70’ yang berisi fitur data yang memengaruhi prediksi dan ‘y_train70’ yang merupakan data target prediksi. Setelah itu, parameter *epochs* menentukan jumlah iterasi pemrosesan data oleh model LSTM. Dengan menggunakan parameter *batch_size*, ditentukan bahwa setiap iterasi akan mengambil sampel data sejumlah 32. Parameter *verbose* akan mengontrol sejauh mana informasi pelatihan akan ditampilkan selama proses pelatihan. Parameter terakhir adalah *validation_split* yang menentukan sebagian data pelatihan yang akan diambil untuk validasi. Dalam hal ini, 30% dari data pelatihan akan diambil sebagai data validasi untuk memantau kinerja model selama pelatihan. Besaran parameter tersebut bersesuaian dengan jumlah data *test* yang digunakan pada skenario ini.

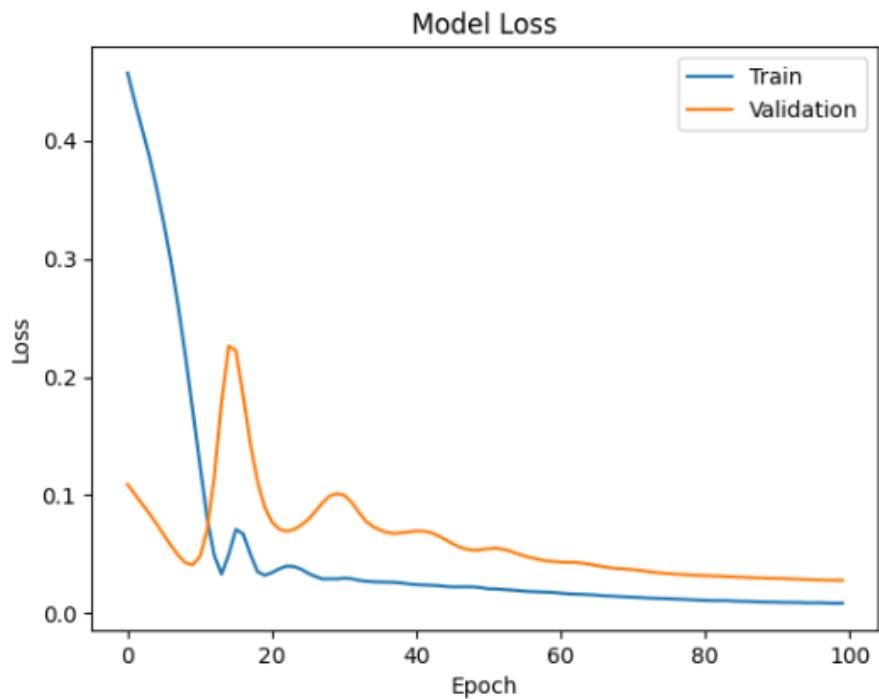
d. Plotting Training-Validation Loss

Langkah ini digunakan untuk membandingkan *data loss* pada data *training* dan *validation* hasil dari model LSTM.

```

# Plot training & validation loss values
plt.plot(history70.history['loss'])
plt.plot(history70.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

```



Gambar 136. Training-Validation Loss Plot LSTM

Berdasarkan grafik, diketahui bahwa model masih belum merupakan model terbaik karena bentuk grafik *validation* tidak begitu sesuai dengan bentuk grafik *training*.

e. Make Prediction on the Test Set

Menggunakan model LSTM yang sudah ada, dilakukan prediksi terhadap data ‘X_test30’ sebagai berikut.

```
# Make predictions on the test set
y_pred30_scaled = lstm70_model.predict(x_test30)

print("Shape Hasil Prediksi:", y_pred30_scaled.shape)
print("Hasil Prediksi:", y_pred30_scaled)

1/1 [=====] - 0s 309ms/step
Shape Hasil Prediksi: (20, 1)
Hasil Prediksi: [[0.16364329]
[0.16428101]
[0.16500986]
[0.1658352 ]
[0.16650249]
[0.16720378]
[0.16793716]
[0.16867527]
[0.16939726]
[0.17000593]
[0.17076954]
[0.17155887]
[0.17217372]
[0.17289221]
[0.17370568]
[0.17444023]
[0.17529425]
[0.17667857]
[0.1776485 ]
[0.1786266 ]]
```

Gambar 137. Hasil Prediksi Model LSTM

Berdasarkan kode tersebut, diketahui bahwa hasil prediksi berupa *array* dengan *value* dalam skala normalisasi dengan dimensi data berjumlah 20 baris.

f. Inverse Transform Dataset

Karena data saat ini dalam skala hasil normalisasi, maka perlu dilakukan denormalisasi data untuk mengembalikannya dalam skala aktual. Untuk itu, digunakan fungsi ‘inverse_transform’ pada data yang perlu diberlakukan denormalisasi.

```

y_pred30_sc = np.repeat(y_pred30_scaled,2, axis=-1)
y_pred30 = scaler_lstm.inverse_transform(np.reshape(y_pred30_sc,(len(y_pred30_scaled),2))[:,0])

y_test30_sc = np.repeat(y_test30,2, axis=-1)
y_test30_re = scaler_lstm.inverse_transform(np.reshape(y_test30_sc,(len(y_test30),2))[:,0])

print("Original Test Shape:", y_test30.shape)
print("Transformed Test Shape:", y_test30_re.shape)
print("Predicted Shape:", y_pred30.shape)

Original Test Shape: (20, 1)
Transformed Test Shape: (20,)
Predicted Shape: (20,)
Original Test Shape: (20, 1)
Transformed Test Shape: (20,)
Predicted Shape: (20,)

# print("Original Test Subset:", y_test20)
print("Transformed Test Subset:", y_test30_re)
print("Predicted Subset:", y_pred30)

Transformed Test Subset: [3.000e+00 1.300e+01 1.520e+02 1.200e+01 1.200e+01 7.300e+01 6.400e+01
1.420e+02 1.840e+02 2.340e+02 2.260e+02 3.170e+02 7.370e+02 1.203e+03
1.792e+03 1.149e+03 3.120e+03 2.672e+03 2.646e+03 2.511e+03]
Predicted Subset: [10577.738 10618.96 10666.072 10719.422 10762.555 10807.886 10855.29
10903.001 10949.67 10989.014 11038.372 11089.394 11129.138 11175.58
11228.161 11275.643 11330.845 11420.326 11483.021 11546.244]
Transformed Test Subset: [3.000e+00 1.300e+01 1.520e+02 1.200e+01 1.200e+01 7.300e+01 6.400e+01
1.420e+02 1.840e+02 2.340e+02 2.260e+02 3.170e+02 7.370e+02 1.203e+03
1.792e+03 1.149e+03 3.120e+03 2.672e+03 2.646e+03 2.511e+03]
Predicted Subset: [-585.88416 -592.99054 -597.45746 -601.78644 -611.7836 -620.809
-630.13837 -638.3332 -644.0528 -646.73883 -655.7875 -664.1465
-673.40826 -680.4418 -681.81744 -684.8648 -691.8188 -701.70807
-712.1263 -716.21655]

```

Gambar 138. Coding dan Hasil Invers Data

Berdasarkan kode tersebut, dilakukan juga *reshape* dimensi data menyesuaikan dimensi data aktual supaya proses invers berjalan sesuai dengan yang diinginkan. Terakhir, dimunculkan juga data tes dan hasil prediksi dalam skala aktual.

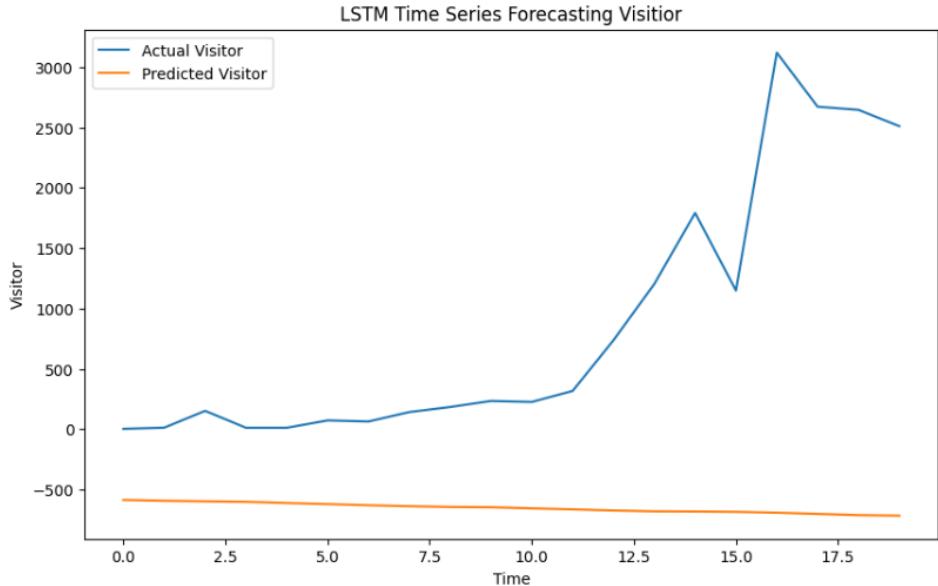
g. Plotting Actual-Prediction

Untuk mengetahui efektivitas model, diperlukan grafik untuk *plotting* perbandingan antara data aktual dan hasil *training*.

```

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(y_test30_re, label='Actual Visitor')
plt.plot(y_pred30, label='Predicted Visitor')
plt.title('LSTM Time Series Forecasting Visitor')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()

```



Gambar 139. Plot Data Aktual dan Hasil Prediksi LSTM

Karena hasil prediksi bernilai negatif, maka terlihat bahwa hasil grafik sangat tidak bersesuaian. Oleh karena itu, diperlukan evaluasi model untuk mengetahui tingkat optimasi dari model LSTM.

h. Evaluate LSTM Model

Langkah terakhir pada eksperimen model LSTM adalah melakukan evaluasi terhadap hasil prediksi. Evaluasi model dilakukan menggunakan perhitungan terhadap fungsi MSE, MAE, dan MAPE.

```

# Calculate MSE, MAE, MAPE
mse_test2 = mean_squared_error(y_test30_re, y_pred30).round(2)
mae_test2 = mean_absolute_error(y_test30_re, y_pred30).round(2)
mape_test2 = mean_absolute_percentage_error(y_test30_re, y_pred30).round(2)

print(f"Mean Squared Error (MSE): {mse_test2}")
print(f"Mean Absolute Error (MAE): {mae_test2}")
print(f"Mean Absolute Percentage Error (MAPE): {mape_test2} %")

```

Mean Squared Error (MSE): 3471923.44
 Mean Absolute Error (MAE): 1514.72
 Mean Absolute Percentage Error (MAPE): 20.18 %

Gambar 140. Evaluasi Model LSTM

Penentu nilai dari model ini adalah MAPE yang bersifat *general* karena nilainya berupa persentase. Evaluasi untuk model LSTM pada skenario ini adalah nilai MAPE sebesar 20,18 persen.

```

# Membuat prediksi visitor menggunakan X_train
y_pred70 = lstm70_model.predict(X_train70)

# Calculate MSE, RMSE, MAPE
mse_train2 = mean_squared_error(y_train70, y_pred70).round(2)
mae_train2 = mean_absolute_error(y_train70, y_pred70).round(2)
mape_train2 = mean_absolute_percentage_error(y_train70, y_pred70).round(2)

print(f"Mean Squared Error (MSE): {mse_train2}")
print(f"Mean Absolute Error (MAE): {mae_train2}")
print(f"Mean Absolute Percentage Error (MAPE): {mape_train2} %")

2/2 [=====] - 0s 8ms/step
Mean Squared Error (MSE): 0.36
Mean Absolute Error (MAE): 0.54
Mean Absolute Percentage Error (MAPE): 1.01 %

```

Gambar 141. Evaluasi Model Train LSTM

Hal yang sama juga dilakukan terhadap prediksi pada data *train*.

- Skenario 60 % Train

- a. Splitting Train and Test Data

Hal pertama yang dilakukan sebelum membangun model LSTM adalah membagi keseluruhan data menjadi *training* dan *testing* data. Pembagian ini disesuaikan kondisi skenario yang telah ditentukan. Menurut kode yang tertera pada gambar terlampir, ditentukan bahwa data yang digunakan sebagai data *training* berjumlah 0,6 kali (60%) keseluruhan jumlah data, sedangkan sisanya merupakan data *testing*. Sumber data pada pembagian ini adalah ‘*ntt_scaled*’ yang merupakan data hasil normalisasi. Hasil menunjukkan bahwa jumlah baris pada data *training* dan *testing* masing-masing sebanyak 64 dan 44 baris.

```

# Split the data into training and testing sets
ntt_val = ntt_lstm.values
train_lstm60 = int(len(ntt_val) * 0.6)
test_lstm40 = len(ntt_val) - train_lstm60
train60, test40 = ntt_scaled[0:train_lstm60,:], ntt_scaled[train_lstm60:len(ntt_val),:]

print(len(train60), len(test40))

64 44

```

Gambar 142. Coding dan Hasil Splitting Data

Setelah itu, proses pada langkah ini dilanjutkan dengan pembuatan *time series* berdasarkan panjang sekuens yang telah ditentukan. Untuk bagian ini, penjelasan terdapat pada sub bab ‘Time Series (Sequence)’ dalam bab ketiga laporan ini.

```

# convert an array of values into a dataset matrix
def create_dataset(data, look_back):
    dataX, dataY = [], []
    for i in range(len(data)-look_back):
        dataX.append(data[i:(i+look_back), 0:data.shape[1]])
        dataY.append(data[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

# reshape into X=t and Y=t+1
look_back = 12
X_train60, y_train60 = create_dataset(train60, look_back)
X_test40, y_test40 = create_dataset(test40, look_back)

```

Gambar 143. Coding Pembuatan Time Series

Proses berlanjut dengan menyesuaikan kembali dimensi data pada masing-masing data *train* dan *test*. Untuk setiap data dengan variabel X harus memiliki bentuk yang sama, begitu juga dengan variabel y. Selain itu, jumlah data yang digunakan pada masing-masing data *train* dan *test* harus sama, yaitu sesuai dengan jumlah data pada saat pembagian sebelumnya.

```
# Reshape the output (y) to ensure it is a 2D array
y_train60 = y_train60.reshape(-1, 1)
y_test40 = y_test40.reshape(-1, 1)

# reshape input to be [samples, time steps, features]
X_train60 = np.reshape(X_train60, (X_train60.shape[0], 2, X_train60.shape[1]))
X_test40 = np.reshape(X_test40, (X_test40.shape[0], 2, X_test40.shape[1]))

print("Train X shape:", X_train60.shape)
print("Train Y shape:", y_train60.shape)
print("Test X shape:", X_test40.shape)
print("Test Y shape:", y_test40.shape)

Train X shape: (52, 2, 12)
Train Y shape: (52, 1)
Test X shape: (32, 2, 12)
Test Y shape: (32, 1)
```

Gambar 144. Coding dan Hasil Reshape Dataset

Gambar di atas menunjukkan hasil akhir dimensi pada masing-masing data *train* dan *test* pada masing-masing variabel.

b. Create LSTM Model

Langkah kedua yang dilakukan adalah pembangunan model LSTM untuk skenario ini. Pembuatan model dilakukan menggunakan fungsi ‘Sequential’ pada metode keras milik *library* ‘Tensorflow’ yang telah *di-import* pada bagian awal penggeraan. Setelah itu, ditetapkan parameter dan *input* yang akan digunakan pada model LSTM pada *layer* pertama dan kedua. Setelah menetapkan *layer* keluaran, disebutkan fungsi *compile* untuk menjalankan model LSTM tersebut.

```
# Build the LSTM model
lstm60_model = Sequential()

# First LSTM layer
lstm60_model.add(LSTM(100, activation='relu', return_sequences=True, input_shape=(X_train60.shape[1], look_back)))

# Second LSTM layer
lstm60_model.add(LSTM(100, activation='relu'))

# Output layer
lstm60_model.add(Dense(1)) # Assuming you want to predict 1 feature (visitor)

# optimizer = Adam(learning_rate=0.001)
lstm60_model.compile(optimizer='adam', loss='mse')
```

Gambar 145. Coding Pembuatan Model LSTM Skenario 3

```
lstm60_model.summary()  
  
Model: "sequential_17"  
  
Layer (type)          Output Shape         Param #  
=====              ======           ======
```

lstm_34 (LSTM)	(None, 2, 100)	45200
lstm_35 (LSTM)	(None, 100)	80400
dense_17 (Dense)	(None, 1)	101


```
Total params: 125701 (491.02 KB)  
Trainable params: 125701 (491.02 KB)  
Non-trainable params: 0 (0.00 Byte)
```

Gambar 146. Ringkasan Model LSTM

Gambar di atas merupakan *summary* dari model LSTM yang telah dibuat.

c. Training LSTM Model

Langkah berikutnya adalah melakukan *training* terhadap model LSTM yang telah dibuat. Berikut kode yang digunakan.

```

# Train the model
history60 = lstm60_model.fit(
    X_train60, y_train60,
    epochs=100,
    batch_size=32,
    verbose=2,
    validation_split=0.4, # Use part of the training data as validation
)

Epoch 67/100
1/1 - 0s - loss: 0.0240 - val_loss: 0.1222 - 36ms/epoch - 36ms/step
Epoch 68/100
1/1 - 0s - loss: 0.0236 - val_loss: 0.1168 - 37ms/epoch - 37ms/step
Epoch 69/100
1/1 - 0s - loss: 0.0234 - val_loss: 0.1119 - 38ms/epoch - 38ms/step
Epoch 70/100
1/1 - 0s - loss: 0.0234 - val_loss: 0.1077 - 38ms/epoch - 38ms/step
Epoch 71/100
1/1 - 0s - loss: 0.0233 - val_loss: 0.1043 - 38ms/epoch - 38ms/step
Epoch 72/100
1/1 - 0s - loss: 0.0234 - val_loss: 0.1017 - 35ms/epoch - 35ms/step
Epoch 73/100
1/1 - 0s - loss: 0.0234 - val_loss: 0.0999 - 35ms/epoch - 35ms/step
Epoch 74/100
1/1 - 0s - loss: 0.0234 - val_loss: 0.0988 - 39ms/epoch - 39ms/step
Epoch 75/100
1/1 - 0s - loss: 0.0234 - val_loss: 0.0984 - 37ms/epoch - 37ms/step
Epoch 76/100
1/1 - 0s - loss: 0.0233 - val_loss: 0.0985 - 36ms/epoch - 36ms/step
Epoch 77/100
1/1 - 0s - loss: 0.0232 - val_loss: 0.0992 - 37ms/epoch - 37ms/step
Epoch 78/100
1/1 - 0s - loss: 0.0230 - val_loss: 0.1004 - 37ms/epoch - 37ms/step
Epoch 79/100

```

Gambar 147. Coding Training Model LSTM

Fungsi yang digunakan adalah *fitting model* terhadap model yang sudah ada. Data yang digunakan pada langkah ini adalah ‘X_train60’ yang berisi fitur data yang memengaruhi prediksi dan ‘y_train60’ yang merupakan data target prediksi. Setelah itu, parameter *epochs* menentukan jumlah iterasi pemrosesan data oleh model LSTM. Dengan menggunakan parameter *batch_size*, ditentukan bahwa setiap iterasi akan mengambil sampel data sejumlah 32. Parameter *verbose* akan mengontrol sejauh mana informasi pelatihan akan ditampilkan selama proses pelatihan. Parameter terakhir adalah *validation_split* yang menentukan sebagian data pelatihan yang akan diambil untuk validasi. Dalam hal ini, 40% dari data pelatihan akan diambil sebagai data validasi untuk memantau kinerja model selama pelatihan. Besaran parameter tersebut bersesuaian dengan jumlah data *test* yang digunakan pada skenario ini.

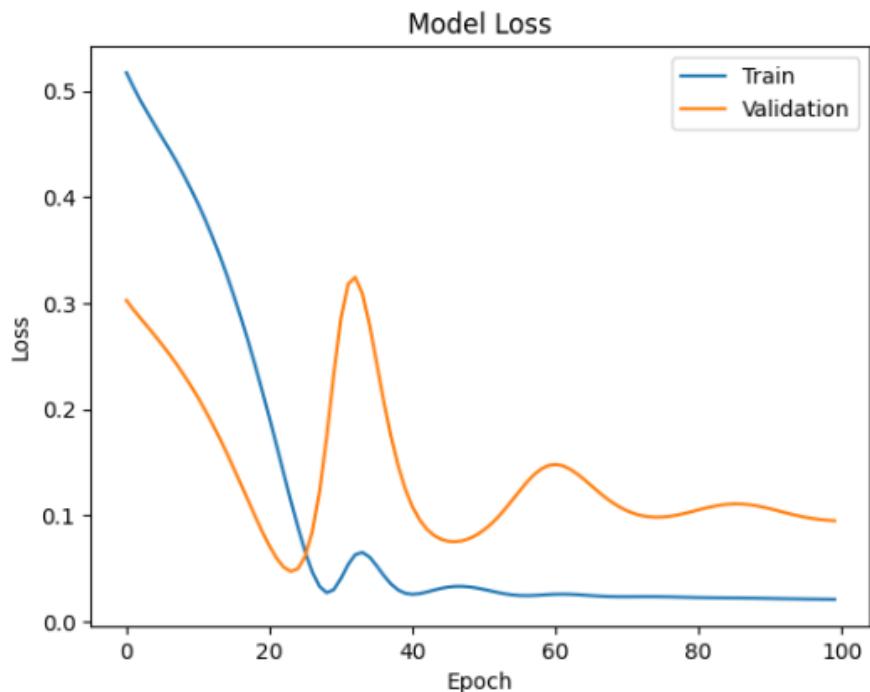
d. Plotting Training-Validation Loss

Langkah ini digunakan untuk membandingkan *data loss* pada data *training* dan *validation* hasil dari model LSTM.

```

# Plot training & validation loss values
plt.plot(history60.history['loss'])
plt.plot(history60.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

```



Gambar 148. Training-Validation Loss Plot LSTM

Berdasarkan grafik, diketahui bahwa model masih belum merupakan model terbaik karena bentuk grafik *validation* tidak begitu sesuai dengan bentuk grafik *training*.

e. Make Prediction on the Test Set

Menggunakan model LSTM yang sudah ada, dilakukan prediksi terhadap data ‘X_test40’ sebagai berikut.

```

# Make predictions on the test set
y_pred40_scaled = lstm60_model.predict(x_test40)

print("Shape Hasil Prediksi:", y_pred40_scaled.shape)
print("Hasil Prediksi:", y_pred40_scaled)

1/1 [=====] - 0s 398ms/step
Shape Hasil Prediksi: (32, 1)
Hasil Prediksi: [[0.44352207]
[0.44344878]
[0.44270408]
[0.44250622]
[0.4420861 ]
[0.4434231 ]
[0.43990666]
[0.43404412]
[0.42290536]
[0.4124171 ]
[0.408033  ]
[0.4074958 ]
[0.41121525]
[0.41491088]
[0.4187207 ]
[0.42212754]
[0.4257971 ]
[0.42966637]
[0.4333449 ]
[0.43669468]
[0.4405753 ]
[0.44428656]
[0.44781786]
[0.45145473]
[0.45491225]
[0.45888305]
[0.46336913]
[0.4680861 ]
[0.47376987]
[0.47882617]
[0.48418787]
[0.4897526 ]]

```

Gambar 149. Hasil Prediksi Model LSTM

Berdasarkan kode tersebut, diketahui bahwa hasil prediksi berupa *array* dengan *value* dalam skala normalisasi dengan dimensi data berjumlah 32z baris.

f. Inverse Transform Dataset

Karena data saat ini dalam skala hasil normalisasi, maka perlu dilakukan denormalisasi data untuk mengembalikannya dalam skala aktual. Untuk itu, digunakan fungsi ‘*inverse_transform*’ pada data yang perlu diberlakukan denormalisasi.

```

y_pred40_sc = np.repeat(y_pred40_scaled, 2, axis=-1)
y_pred40 = scaler_lstm.inverse_transform(np.reshape(y_pred40_sc,(len(y_pred40_scaled),2))[:,0])

y_test40_sc = np.repeat(y_test40, 2, axis=-1)
y_test40_re = scaler_lstm.inverse_transform(np.reshape(y_test40_sc,(len(y_test40),2))[:,0])

print("Original Test Shape:", y_test40.shape)
print("Transformed Test Shape:", y_test40_re.shape)
print("Predicted Shape:", y_pred40.shape)

Original Test Shape: (32, 1)
Transformed Test Shape: (32,)
Predicted Shape: (32,)

# print("Original Test Subset:", y_test20)
print("Transformed Test Subset:", y_test40_re)
print("Predicted Subset:", y_pred40)

Transformed Test Subset: [2.000e+00 0.000e+00 3.500e+01 3.100e+01 0.000e+00 2.800e+01 2.600e+01
1.300e+01 4.000e+00 2.600e+01 0.000e+00 3.000e+00 1.300e+01 1.520e+02
1.200e+01 1.200e+01 7.300e+01 6.400e+01 1.420e+02 1.840e+02 2.340e+02
2.260e+02 3.170e+02 7.370e+02 1.203e+03 1.792e+03 1.149e+03 3.120e+03
2.672e+03 2.646e+03 2.511e+03 4.300e+03]
Predicted Subset: [28668.822 28664.086 28615.95 28603.16 28576.004 28662.426 28435.127
28056.178 27336.18 26658.23 26374.846 26340.121 26580.543 26819.424
27065.688 27285.902 27523.1 27773.205 28010.98 28227.508 28478.348
28718.238 28946.498 29181.582 29405.072 29661.742 29951.717 30256.617
30624.012 30950.844 31297.42 31657.117]

```

Gambar 150. Coding dan Hasil Invers Data

Berdasarkan kode tersebut, dilakukan juga *reshape* dimensi data menyesuaikan dimensi data aktual supaya proses invers berjalan sesuai dengan yang diinginkan. Terakhir, dimunculkan juga data tes dan hasil prediksi dalam skala aktual.

g. Plotting Actual-Prediction

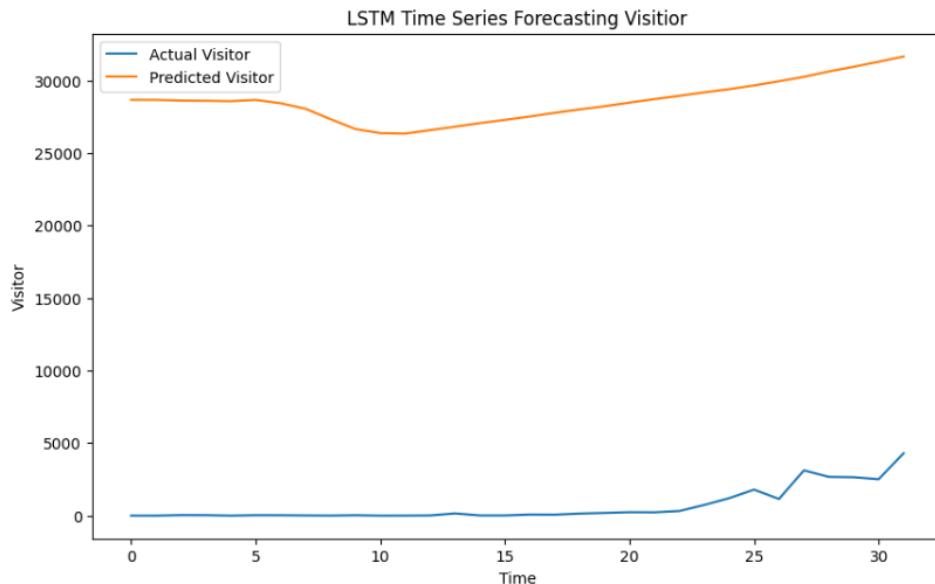
Untuk mengetahui efektivitas model, diperlukan grafik untuk *plotting* perbandingan antara data aktual dan hasil *training*.

```

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(y_test40_re, label='Actual Visitor')
plt.plot(y_pred40, label='Predicted Visitor')
plt.title('LSTM Time Series Forecasting Visitor')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()

plt.show()

```



Gambar 151. Plot Data Aktual dan Hasil Prediksi LSTM

Karena data *test* tidak berhasil diberlakukan invers menggunakan model yang ada pada skenario ini, maka hasil *plotting* data menjadi seperti pada gambar di atas, nominal tidak menggunakan skala yang sama. Oleh karena itu, diperlukan evaluasi model untuk mengetahui tingkat optimasi dari model LSTM.

h. Evaluate LSTM Model

Langkah terakhir pada eksperimen model LSTM adalah melakukan evaluasi terhadap hasil prediksi. Evaluasi model dilakukan menggunakan perhitungan terhadap fungsi MSE, MAE, dan MAPE.

```
# Calculate MSE, MAE, MAPE
mse_test3 = mean_squared_error(y_test40_re, y_pred40).round(2)
mae_test3 = mean_absolute_error(y_test40_re, y_pred40).round(2)
mape_test3 = mean_absolute_percentage_error(y_test40_re, y_pred40).round(2)

print(f"Mean Squared Error (MSE): {mse_test3}")
print(f"Mean Absolute Error (MAE): {mae_test3}")
print(f"Mean Absolute Percetange Error (MAPE): {mape_test3} %")

Mean Squared Error (MSE): 777042701.36
Mean Absolute Error (MAE): 27864.99
Mean Absolute Percetange Error (MAPE): 1.1767756017859822e+19 %
```

Gambar 152. Evaluasi Model LSTM

Penentu nilai dari model ini adalah MAPE yang bersifat *general* karena nilainya berupa persentase. Evaluasi untuk model LSTM pada skenario ini adalah nilai MAPE sangat besar. Hal ini disebabkan oleh data *test* yang nilainya masih dalam skala normalisasi, gagal diberlakukan invers.

```
# Membuat prediksi visitor menggunakan x_train
y_pred60 = lstm60_model.predict(x_train60)

# Calculate MSE, RMSE, MAPE
mse_train3 = mean_squared_error(y_train60, y_pred60).round(2)
mae_train3 = mean_absolute_error(y_train60, y_pred60).round(2)
mape_train3 = mean_absolute_percentage_error(y_train60, y_pred60).round(2)

print(f"Mean Squared Error (MSE): {mse_train3}")
print(f"Mean Absolute Error (MAE): {mae_train3}")
print(f"Mean Absolute Percetange Error (MAPE): {mape_train3} %")

2/2 [=====] - 0s 12ms/step
Mean Squared Error (MSE): 0.05
Mean Absolute Error (MAE): 0.18
Mean Absolute Percetange Error (MAPE): 0.61 %
```

Gambar 153. Evaluasi Model Train LSTM

Hal yang sama juga dilakukan terhadap prediksi pada data *train*.

Scenario	Model	MSE	MAE	MAPE (%)
Scenario 1	RNN Train 80	0.02	0.1	1.30261e+13
Scenario 1	RNN Test 80	6.19661e+06	1985.68	0.99
Scenario 2	RNN Train 70	0.36	0.54	1.01
Scenario 2	RNN Test 70	3.47192e+06	1514.72	20.18
Scenario 3	RNN Train 60	0.05	0.18	0.61
Scenario 3	RNN Test 60	7.77043e+08	27865	1.17678e+19

MAPE sering kali efektif untuk menganalisis kumpulan data yang besar dan membutuhkan penggunaan nilai kumpulan data selain nol. Nilai MAPE lebih mudah diinterpretasikan dibandingkan alat ukur yang lain karena berupa nilai persen yang dapat terukur skalanya. Berdasarkan nilai MAPE, skenario terbaik pada model LSTM adalah skenario 1 dengan pembagian 80% data latih (train) dan 20% data uji (test).

4. GRU

- Skenario 80 % Train

Selanjutnya yaitu, melakukan analisis GRU dengan Skenario 80 % Train dan 20 % test dengan melalui beberapa tahapan dibawah ini :

a. Splitting Train and Test Data

Membagi data pelatihan dan pengujian dengan rasio 80% dan 20% serta menetapkan hasilnya ke dalam variabel baru, seperti yang ditunjukkan di bawah ini:

```
[ ] # Getting trainX and trainY
trainX = []
trainY = []
n_future = 1
n_past = 24

for i in range(n_past, len(scaled_data_train) - n_future + 1):
    trainX.append(scaled_data_train[i - n_past:i, 0:scaled_data_train.shape[1]])
    trainY.append(scaled_data_train[i + n_future - 1:i + n_future, 0])

trainX = np.array(trainX)
trainY = np.array(trainY)

# Memisahkan data menjadi train dan test
X_train, X_test, y_train, y_test = train_test_split(trainX, trainY, test_size=0.2, random_state=42)

print('X_train shape == {}'.format(X_train.shape))
print('y_train shape == {}'.format(y_train.shape))
print('X_test shape == {}'.format(X_test.shape))
print('y_test shape == {}'.format(y_test.shape))

X_train shape == (67, 24, 2).
y_train shape == (67, 1).
X_test shape == (17, 24, 2).
y_test shape == (17, 1).
```

Gambar 154. Splitting train & Test GRU 80 %

Dalam penjelasan di atas, yaitu menghitung ukuran data pelatihan berdasarkan proporsi tertentu dari total urutan. Selanjutnya, menggunakan nilai tersebut untuk memisahkan urutan menjadi data pelatihan dan pengujian yang menjadi variabel baru Variabel X_train, X_test, y_train, dan y_test kemudian digunakan untuk menyimpan data dan label yang

sesuai. Lalu, melakukan print untuk menampilkan ukuran dari data pelatihan dan pengujian untuk memastikan proses pemisahan berjalan dengan baik.

b. Create Model

Tahap Selanjutnya, yaitu memanggil model GRU untuk membuat kerangka kerja model gru dengan menggunakan beberapa parameter yang telah ditentukan pada kode berikut.

```
[509] # Define the GRU model
model = Sequential()
model.add(GRU(64, activation = 'tanh', input_shape = (trainX.shape[1], trainX.shape[2])))
model.add(Dense(trainY.shape[1]))

model.summary()
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

Model: "sequential_12"
-----  

Layer (type)          Output Shape         Param #
-----  

gru_9 (GRU)           (None, 64)           13056  

dense_9 (Dense)       (None, 1)            65  

-----  

Total params: 13121 (51.25 KB)
Trainable params: 13121 (51.25 KB)
Non-trainable params: 0 (0.00 Byte)
```

Gambar 155. Model GRU 80 %

Pertama, sebuah objek model sequential dibuat menggunakan Sequential(). Model ini adalah wadah linear untuk lapisan-lapisan neural network. Kemudian, lapisan GRU ditambahkan ke dalam model menggunakan model.add(GRU(64, activation='tanh', input_shape=(trainX.shape[1], trainX.shape[2]))). Lapisan GRU ini memiliki 64 unit (neuron) dengan fungsi aktivasi tangen hiperbolik (tanh) dan mendefinisikan bentuk input dengan parameter input_shape sesuai dengan dimensi data time series yang diharapkan.

Terakhir, sebuah lapisan Dense ditambahkan dengan menggunakan model.add(Dense(trainY.shape[1])). Lapisan Dense ini bertanggung jawab untuk menghasilkan output dengan jumlah dimensi yang sesuai dengan dimensi target, yang dalam konteks ini sesuai dengan trainY.shape[1]. Dengan menambahkan lapisan ini, model memiliki kemampuan untuk menghasilkan prediksi yang sesuai dengan struktur target yang diharapkan.

c. Fit Model

```

    history = model.fit(trainX, trainY, epochs=100, batch_size=32, verbose=2, validation_split=0.2)

    Epoch 72/100
    3/3 - 0s - loss: 0.1539 - val_loss: 0.1779 - 209ms/epoch - 70ms/step
    Epoch 73/100
    3/3 - 0s - loss: 0.1533 - val_loss: 0.1689 - 148ms/epoch - 49ms/step
    Epoch 74/100
    3/3 - 0s - loss: 0.1525 - val_loss: 0.1575 - 181ms/epoch - 60ms/step
    Epoch 75/100
    3/3 - 0s - loss: 0.1519 - val_loss: 0.1482 - 200ms/epoch - 67ms/step
    Epoch 76/100
    3/3 - 0s - loss: 0.1523 - val_loss: 0.1365 - 119ms/epoch - 40ms/step
    Epoch 77/100
    3/3 - 0s - loss: 0.1520 - val_loss: 0.1331 - 234ms/epoch - 78ms/step
    Epoch 78/100
    3/3 - 0s - loss: 0.1525 - val_loss: 0.1321 - 179ms/epoch - 60ms/step
    Epoch 79/100
    3/3 - 0s - loss: 0.1511 - val_loss: 0.1400 - 199ms/epoch - 66ms/step
    Epoch 80/100
    3/3 - 0s - loss: 0.1492 - val_loss: 0.1530 - 144ms/epoch - 48ms/step
    Epoch 81/100
    3/3 - 0s - loss: 0.1479 - val_loss: 0.1758 - 119ms/epoch - 40ms/step
    Epoch 82/100
    3/3 - 0s - loss: 0.1449 - val_loss: 0.2019 - 186ms/epoch - 62ms/step
    Epoch 83/100
    3/3 - 0s - loss: 0.1450 - val_loss: 0.2243 - 192ms/epoch - 64ms/step
    Epoch 84/100
    3/3 - 0s - loss: 0.1443 - val_loss: 0.2238 - 204ms/epoch - 68ms/step
    Epoch 85/100
    3/3 - 0s - loss: 0.1438 - val_loss: 0.2044 - 171ms/epoch - 57ms/step
    Epoch 86/100
    3/3 - 0s - loss: 0.1425 - val_loss: 0.1904 - 186ms/epoch - 62ms/step
    Epoch 87/100
    3/3 - 0s - loss: 0.1419 - val_loss: 0.1859 - 139ms/epoch - 46ms/step
    Epoch 88/100

```

Gambar 156. Model GRU 80 %

Pada kode tersebut, trainX dan trainY adalah input dan output yang digunakan untuk melatih model. Parameter epochs=100 menentukan berapa kali model akan melihat seluruh dataset pada proses pelatihan. Batch size, ditentukan oleh batch_size=32, menentukan seberapa banyak sampel data yang digunakan pada setiap iterasi pembelajaran. Semakin kecil nilai batch size, semakin sering pembelajaran akan diperbarui, tetapi akan membutuhkan lebih banyak waktu komputasi ini bertujuan untuk menghasilkan model GRU yang dapat memahami dan memprediksi pola dalam data time series dengan lebih baik.

Selama proses pelatihan, informasi pelatihan disimpan dalam objek history. Hasil pelatihan tersebut bisa digunakan untuk mengevaluasi performa model pada setiap epoch, melihat perubahan nilai loss function, dan menganalisis bagaimana model beradaptasi terhadap data pelatihan. Parameter verbose=2 menampilkan progres pelatihan secara terperinci, sedangkan nilai 0 akan membuatnya tampil secara diam-diam.

d. Loss Model

Tahap selanjutnya yaitu, mengenai Loss Model. Loss Model sendiri adalah fungsi matematis yang menyederhanakan perbedaan antara prediksi model dengan nilai sebenarnya. Sebagai contoh, dalam regresi, fungsi kerugian umumnya adalah Mean Squared Error (MSE), yang menghitung rata-rata dari kuadrat selisih antara prediksi dan ground truth. Proses pelatihan model melibatkan iteratif memperbarui parameter-parameter model untuk meminimalkan nilai loss ini. Sebagai penerapan loss model, maka dibuat seperti berikut, dimana loss model diambil dari Fit GRU diatas.

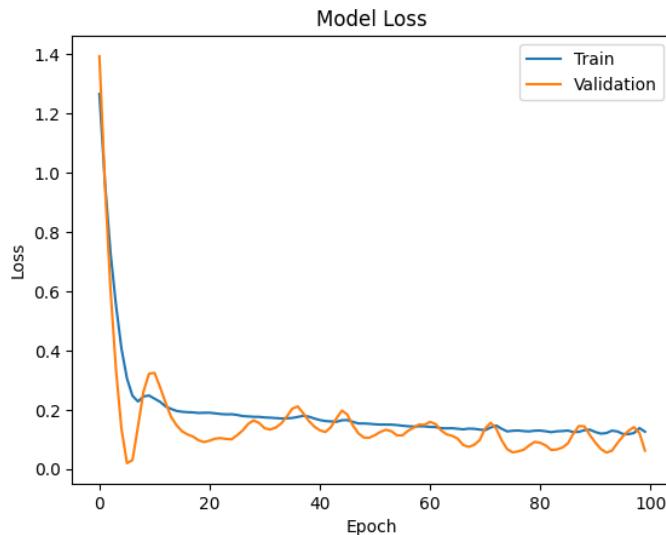
```

[ ] # Plot the training loss
plt.plot(history.history['loss'], label='Training loss')
plt.legend()

```

Gambar 157. Loss Model GRU 80 %

Berdasarkan plot pada gambar diatas, dapat diartikan sebagai pemberian label, legenda berdasarkan data history yang menghasilkan plot train loss dan val loss seperti berikut



Gambar 158. Hasil Loss Model GRU 80 %

e. Prediction

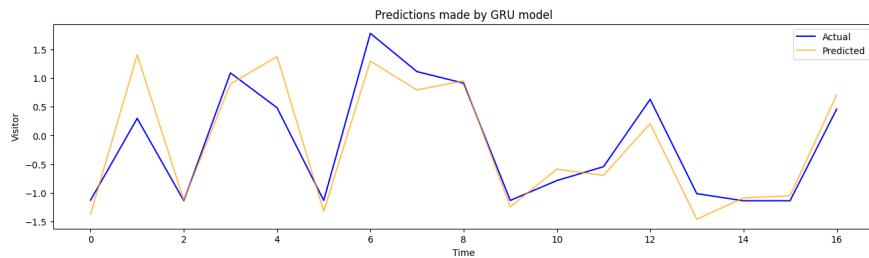
Selanjutnya yaitu, melakukan prediksi terhadap model untuk X_{test} , dimana prediction sendiri dalam GRU melibatkan mengalirkan input melalui model dan memperoleh output prediksi. Hasil ini kemudian dapat dibandingkan dengan nilai sebenarnya untuk mengevaluasi seberapa baik model dapat melakukan prediksi. Pada umumnya, hasil prediksi GRU dapat digunakan untuk berbagai aplikasi, termasuk prediksi time series, pengenalan pola, dan tugas-tugas lain yang melibatkan urutan data.

```
[25] # Melakukan prediksi pada test set
      predictions = model.predict(X_test)
```

```
1/1 [=====] - 0s 445ms/step
```

Gambar 11. Prediksi Model GRU 80 %

Berdasarkan gambar diatas, maka prediksi berhasil dijalankan dengan menggunakan fungsi predict dan menghasilkan plot seperti dibawah ini.



Gambar 159. Prediksi Model GRU 80 %

f. Evaluate Model (Train)

Selanjutnya yaitu tahap prediksi pada model data train, dimana data train yang akan diprediksi yaitu X_{train} , dilanjut dengan penerapan fungsi scaler Inverse untuk mengembalikan nilai-nilai total_visitor dari y_{train} hingga memplot hasil Train Set.

```
[28] # Membuat prediksi visitor menggunakan X_train
y_train_pred = model.predict(X_train)

[29] # Mengembalikan skala nilai visitor dari y_train
y_train_copies = np.repeat(y_train.reshape(-1, 1), X_train.shape[-1], axis=-1)
y_train_real = scaler.inverse_transform(y_train_copies)[:,1]
y_train_real

array([10000.48114892, 10614.65660256, 10457.74815333, 15022.50152488,
       12744.16234209, 10666.66914997, 12529.54135023, 14539.42616808,
       9911.02274951, 13573.27545448, 10617.26910449, 12942.39582184,
       12585.35389146, 10517.28153063, 10245.66049668, 13731.37140458,
       9906.27274601, 14856.35081128, 9906.19357928, 9905.24357858,
       14371.67187751, 9910.31024899, 11717.68658393, 9905.24357858,
       12400.73708843, 9905.24357858, 13113.95011521, 13134.21679685,
       10438.96647275, 14297.88848968, 9906.19357928, 13814.81313288,
       10525.51487005, 9963.58945501, 9907.30191343, 10215.73547458,
       10682.97748635, 10104.03122541, 13228.02936614, 12438.34128287,
       10595.41908835, 12257.76198283, 10893.08597487, 11319.31962302,
       10656.69413361, 10116.77706815, 10047.11035003, 9907.46024688,
       14202.73008606, 10152.24376102, 9906.27274601, 9905.48107875,
       9923.76859226, 10378.10642784, 11940.8575821 , 12858.974336 ,
       9905.40191203, 13883.45068357, 12555.5872028 , 13441.22535694,
       9996.20614576, 9930.33943045, 13869.75484012, 9905.56024548,
       9919.81025601, 13482.86785437, 9905.24357858])

[30] # Mengembalikan skala nilai visitor dari y_train_pred
y_train_pred_copies = np.repeat(y_train_pred, X_train.shape[-1], axis=-1)
y_train_pred_real = scaler.inverse_transform(y_train_pred_copies)[:,1]
y_train_pred_real

array([ 9457.409, 10809.791, 10676.231, 14037.328, 12191.876, 10813.002,
       12407.425, 13904.48 , 9645.871, 13578.544, 10727.877, 12353.183,
       12420.479, 10768.797, 9342.913, 13214.257, 9749.151, 13762.373,
       9673.165, 9961.967, 13288.724, 9620.29 , 11889.2 , 10147.605,
       12789.428, 9804.314, 13008.586, 13421.311, 10919.226, 13828.699,
       9694.951, 13686.027, 12195.252, 9471.073, 9834.782, 10621.586,
       10672.09 , 9364.172, 13146.256, 12210.707, 10823.325, 11899.755,
       10651.776, 12986.509, 10790.185, 9357.852, 9436.509, 9944.208,
       12499.726, 9396.079, 9897.244, 9775.032, 9547.79 , 11265.527,
       12467.924, 12894.694, 10302.602, 13426.249, 12667.103, 13130.015,
       9413.668, 9488.593, 13644.899, 9868.118, 9581.464, 12542.338,
       10477.578], dtype=float32)
```

Gambar 160. Inverse GRU

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
[31] # Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_80_train = mean_squared_error(y_train_real, y_train_pred_real)
mae_80_train = mean_absolute_error(y_train_real, y_train_pred_real)
mape_80_train = mean_absolute_percentage_error(y_train_real, y_train_pred_real)

print(f"Mean Absolute Error (MAE): {round(mae_80_train, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_80_train, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_80_train, 2)}%")

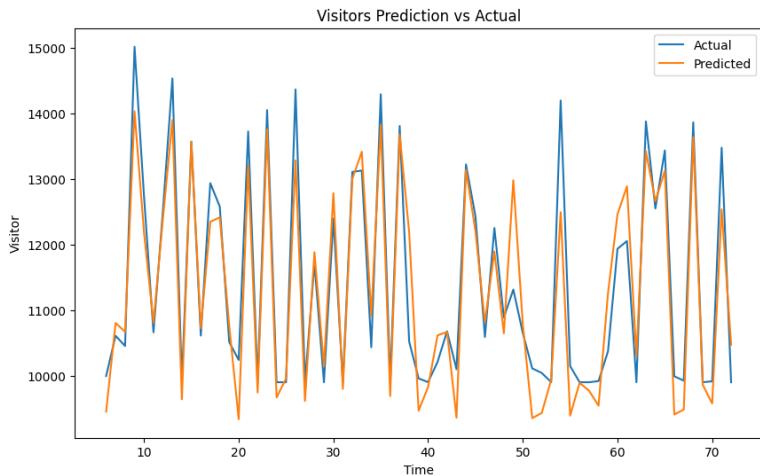
Mean Absolute Error (MAE): 422.06
Mean Squared Error (MSE): 321711.66
Mean Absolute Percentage Error (MAPE): 0.04%
```

Gambar 161. Hasil MSE, MAE, dan MAPE GRU

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model GRU mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model GRU dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.

```
# Melakukan plotting hasil prediksi visitor terhadap data visitor asli dari train set
plt.figure(figsize=(10, 6))
plt.plot(ntt.index[sequence_length:y_train.shape[0]+sequence_length], y_train_real, label='Actual')
plt.plot(ntt.index[sequence_length:y_train.shape[0]+sequence_length], y_train_pred_real, label='Predicted')
plt.title('Visitors Prediction vs Actual')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()
```



Gambar 162.

g. Evaluate Model (Test)

Selanjutnya yaitu tahap prediksi pada model data test, dimana data test yang akan diprediksi yaitu X_test, dilanjut dengan penerapan fungsi scaler Inverse untuk mengembalikan nilai-nilai total_visitor dari y_test hingga memplot hasil Test Set.

```
[38] # Membuat prediksi visitor menggunakan X_test
y_test_pred = model.predict(X_test)

1/1 [=====] - 0s 34ms/step

[39] # Mengembalikan skala nilai visitor dari y_test
y_test_copies = np.repeat(y_test.reshape(-1, 1), X_test.shape[-1], axis=-1)
y_test_real = scaler.inverse_transform(y_test_copies)[:,1]
y_test_real

array([ 9923.13525846, 12306.05368516, 9907.30191343, 13629.087957 ,
       12615.12058011, 9916.48525355, 14780.96384648, 13667.24635722,
      13331.73777607, 9917.2769208 , 10495.58984795, 10899.65681306,
     12860.06242769, 10114.7187333 , 9907.69774706, 9908.01441396,
     12576.17055134])
```

```
[40] # Mengembalikan skala nilai visitor dari y_train_pred
y_test_pred_copies = np.repeat(y_test_pred, X_test.shape[-1], axis=-1)
y_test_pred_real = scaler.inverse_transform(y_test_pred_copies)[:,1]
y_test_pred_real

array([ 9513.523, 14156.002, 9925.42 , 13308.736, 14104.254, 9602.122,
       13973.355, 13133.935, 13398.384, 9727.926, 10827.589, 10645.803,
      12154.16 , 9367.268, 9990.969, 10051.353, 12988.34 ],
      dtype=float32)
```

Gambar 11. Inverse GRU

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
[41] # Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_80_test = mean_squared_error(y_test_real, y_test_pred_real)
mae_80_test = mean_absolute_error(y_test_real, y_test_pred_real)
mape_80_test = mean_absolute_percentage_error(y_test_real, y_test_pred_real)

print(f"Mean Absolute Error (MAE): {round(mae_80_test, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_80_test, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_80_test, 2)}%")

Mean Absolute Error (MAE): 510.38
Mean Squared Error (MSE): 495020.26
Mean Absolute Percentage Error (MAPE): 0.04%
```

Gambar 11. MAE,MSE,MAPE GRU 80 %

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model GRU mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model GRU dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.

```
[42] # Melakukan plotting hasil prediksi visitor terhadap data visitor asli dari train set
plt.figure(figsize=(10, 6))
plt.plot(ntt.index[-y_test.shape[0]:], y_test_real, label='Actual')
plt.plot(ntt.index[-y_test.shape[0]:], y_test_pred_real, label='Predicted')
plt.title('Visitors Prediction vs Actual')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()
```



- Skenario 70 % Train

Selanjutnya yaitu, melakukan analisis GRU dengan Skenario 70 % Train dan 30 % test dengan melalui beberapa tahapan dibawah ini :

- a. Splitting Train and Test Data

Membagi data pelatihan dan pengujian dengan rasio 70% dan 30% serta menetapkan hasilnya ke dalam variabel baru, seperti yang ditunjukkan di bawah ini:

```
[51] # Getting trainX and trainY
trainX = []
trainY = []
n_future = 1
n_past = 24

for i in range(n_past, len(scaled_data_train) - n_future + 1):
    trainX.append(scaled_data_train[i - n_past:i, 0:scaled_data_train.shape[1]])
    trainY.append(scaled_data_train[i + n_future - 1:i + n_future, 0])

trainX = np.array(trainX)
trainY = np.array(trainY)

# Memisahkan data menjadi train dan test
X_train, X_test, y_train, y_test = train_test_split(trainX, trainY, test_size=0.3, random_state=42)

print('X_train shape == {}'.format(X_train.shape))
print('y_train shape == {}'.format(y_train.shape))
print('X_test shape == {}'.format(X_test.shape))
print('y_test shape == {}'.format(y_test.shape))

X_train shape == (58, 24, 2).
y_train shape == (58, 1).
X_test shape == (26, 24, 2).
y_test shape == (26, 1).
```

Gambar 11. Splitting train & Test GRU 70 %

Dalam penjelasan di atas, yaitu menghitung ukuran data pelatihan berdasarkan proporsi tertentu dari total urutan. Selanjutnya, menggunakan nilai tersebut untuk memisahkan urutan menjadi data pelatihan dan pengujian yang menjadi variabel baru Variabel X_train, X_test, y_train, dan y_test kemudian digunakan untuk menyimpan data dan label yang sesuai. Lalu, melakukan print untuk menampilkan ukuran dari data pelatihan dan pengujian untuk memastikan proses pemisahan berjalan dengan baik.

b. Create Model

Tahap Selanjutnya, yaitu memanggil model GRU untuk membuat kerangka kerja model gru dengan menggunakan beberapa parameter yang telah ditentukan pada kode berikut.

```
[52] # Define the GRU model
model = Sequential()
model.add(GRU(64, activation = 'tanh', input_shape = (trainX.shape[1], trainX.shape[2])))
model.add(Dense(trainY.shape[1]))

model.summary()
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

Model: "sequential_1"
-----  

Layer (type)           Output Shape        Param #
-----  

gru_1 (GRU)            (None, 64)          13056  

dense_1 (Dense)        (None, 1)           65  

-----  

Total params: 13121 (51.25 KB)
Trainable params: 13121 (51.25 KB)
Non-trainable params: 0 (0.00 Byte)
```

Gambar 11. Model GRU 70 %

Pertama, sebuah objek model sequential dibuat menggunakan Sequential(). Model ini adalah wadah linear untuk lapisan-lapisan neural network. Kemudian, lapisan GRU ditambahkan ke dalam model menggunakan model.add(GRU(64, activation='tanh', input_shape=(trainX.shape[1], trainX.shape[2]))). Lapisan GRU ini memiliki 64 unit (neuron) dengan fungsi aktivasi tangen hiperbolik (tanh) dan mendefinisikan bentuk input dengan parameter input_shape sesuai dengan dimensi data time series yang diharapkan.

Terakhir, sebuah lapisan Dense ditambahkan dengan menggunakan model.add(Dense(trainY.shape[1])). Lapisan Dense ini bertanggung jawab untuk menghasilkan output dengan jumlah dimensi yang sesuai dengan dimensi target, yang dalam konteks ini sesuai dengan trainY.shape[1]. Dengan menambahkan lapisan ini, model memiliki kemampuan untuk menghasilkan prediksi yang sesuai dengan struktur target yang diharapkan.

c. Fit Model

```
[53] history = model.fit(trainX, trainY, epochs=100, batch_size=32, verbose=1, validation_split=0.2)
Epoch 72/100
3/3 [=====] - 0s 46ms/step - loss: 0.1250 - val_loss: 0.1165
Epoch 73/100
3/3 [=====] - 0s 34ms/step - loss: 0.1254 - val_loss: 0.1252
Epoch 74/100
3/3 [=====] - 0s 31ms/step - loss: 0.1255 - val_loss: 0.1239
Epoch 75/100
3/3 [=====] - 0s 31ms/step - loss: 0.1255 - val_loss: 0.1169
Epoch 76/100
3/3 [=====] - 0s 32ms/step - loss: 0.1244 - val_loss: 0.1005
Epoch 77/100
3/3 [=====] - 0s 34ms/step - loss: 0.1218 - val_loss: 0.1012
Epoch 78/100
3/3 [=====] - 0s 59ms/step - loss: 0.1210 - val_loss: 0.1151
Epoch 79/100
3/3 [=====] - 0s 57ms/step - loss: 0.1217 - val_loss: 0.1070
Epoch 80/100
3/3 [=====] - 0s 47ms/step - loss: 0.1200 - val_loss: 0.0911
Epoch 81/100
3/3 [=====] - 0s 50ms/step - loss: 0.1186 - val_loss: 0.0767
Epoch 82/100
3/3 [=====] - 0s 41ms/step - loss: 0.1168 - val_loss: 0.0719
Epoch 83/100
3/3 [=====] - 0s 43ms/step - loss: 0.1165 - val_loss: 0.0687
Epoch 84/100
3/3 [=====] - 0s 51ms/step - loss: 0.1157 - val_loss: 0.0584
Epoch 85/100
3/3 [=====] - 0s 42ms/step - loss: 0.1128 - val_loss: 0.0474
Epoch 86/100
3/3 [=====] - 0s 44ms/step - loss: 0.1155 - val_loss: 0.0392
Epoch 87/100
3/3 [=====] - 0s 39ms/step - loss: 0.1221 - val_loss: 0.0575
Epoch 88/100
3/3 [=====] - 0s 42ms/step - loss: 0.1163 - val_loss: 0.1103
```

Gambar 11. Model GRU 70 %

Pada kode tersebut, trainX dan trainY adalah input dan output yang digunakan untuk melatih model. Parameter epochs=100 menentukan berapa kali model akan melihat seluruh dataset pada proses pelatihan. Batch size, ditentukan oleh batch_size=32, menentukan seberapa banyak sampel data yang digunakan pada setiap iterasi pembelajaran. Semakin kecil nilai batch size, semakin sering pembelajaran akan diperbarui, tetapi akan membutuhkan lebih banyak waktu komputasi. ini bertujuan untuk menghasilkan model GRU yang dapat memahami dan memprediksi pola dalam data time series dengan lebih baik.

Selama proses pelatihan, informasi pelatihan disimpan dalam objek history. Hasil pelatihan tersebut bisa digunakan untuk mengevaluasi performa model pada setiap epoch, melihat perubahan nilai loss function, dan menganalisis bagaimana model beradaptasi terhadap data pelatihan. Parameter verbose=2 menampilkan progres pelatihan secara terperinci, sedangkan nilai 0 akan membuatnya tampil secara diam-diam.

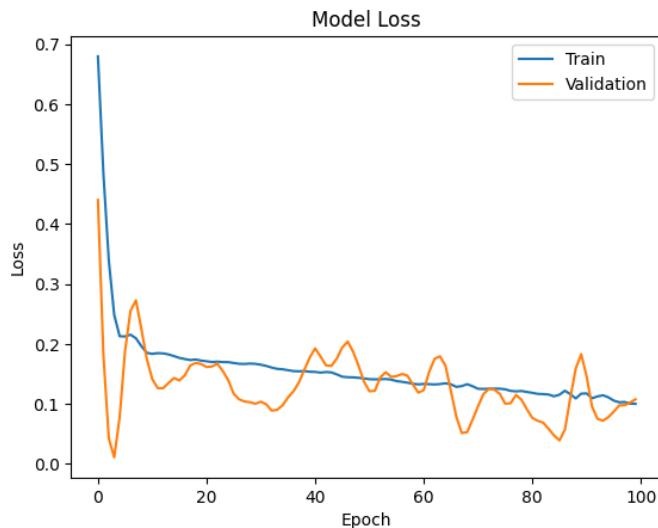
d. Loss Model

Tahap selanjutnya yaitu, mengenai Loss Model. Loss Model sendiri adalah fungsi matematis yang menyederhanakan perbedaan antara prediksi model dengan nilai sebenarnya. Sebagai contoh, dalam regresi, fungsi kerugian umumnya adalah Mean Squared Error (MSE), yang menghitung rata-rata dari kuadrat selisih antara prediksi dan ground truth. Proses pelatihan model melibatkan iteratif memperbarui parameter-parameter model untuk meminimalkan nilai loss ini. Sebagai penerapan loss model, maka dibuat seperti berikut, dimana loss model diambil dari Fit GRU diatas.

```
[ ] # Plot the training loss  
plt.plot(history.history['loss'], label='Training loss')  
plt.legend()
```

Gambar 11. Loss Model GRU 70 %

Berdasarkan plot pada gambar diatas, dapat diartikan sebagai pemberian label, legenda berdasarkan data history yang menghasilkan plot train loss dan val loss seperti berikut



Gambar 11. Hasil Loss Model GRU 70 %

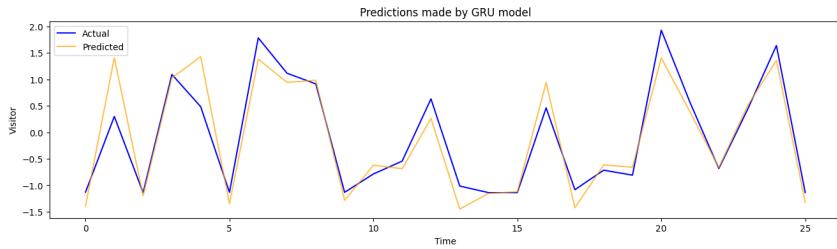
e. Prediction

Selanjutnya yaitu, melakukan prediksi terhadap model untuk X_test, dimana prediction sendiri dalam GRU melibatkan mengalirkan input melalui model dan memperoleh output prediksi. Hasil ini kemudian dapat dibandingkan dengan nilai sebenarnya untuk mengevaluasi seberapa baik model dapat melakukan prediksi. Pada umumnya, hasil prediksi GRU dapat digunakan untuk berbagai aplikasi, termasuk prediksi time series, pengenalan pola, dan tugas-tugas lain yang melibatkan urutan data.

```
[25] # Melakukan prediksi pada test set  
predictions = model.predict(X_test)  
  
1/1 [=====] - 0s 445ms/step
```

Gambar 11. Prediksi Model GRU 70 %

Berdasarkan gambar diatas, maka prediksi berhasil dijalankan dengan menggunakan fungsi predict dan menghasilkan plot seperti dibawah ini.



Gambar 11. Prediksi Model GRU 70 %

f. Evaluate Model (Train)

Selanjutnya yaitu tahap prediksi pada model data train, dimana data train yang akan diprediksi yaitu X_train, dilanjut dengan penerapan fungsi scaler Inverse untuk mengembalikan nilai-nilai total_visitor dari y_train hingga memplot hasil Train Set.

```
[160] # Membuat prediksi visitor menggunakan X_train
y_train_pred = model.predict(X_train)

2/2 [=====] - 0s @ms/step

[161] # Mengembalikan skala nilai visitor dari y_train
y_train_copies = np.repeat(y_train.reshape(-1, 1), X_train.shape[-1], axis=-1)
y_train_real = scaler.inverse_transform(y_train_copies)[:,1]
y_train_real

array([13573.27545448, 10617.26910449, 12942.39582184, 12585.35389146,
       10517.28153063, 10245.66049668, 13731.37140458, 9906.27274601,
       14056.35081128, 9980.19357928, 9905.24357858, 14371.67187751,
       9910.31824899, 11717.68658393, 9905.24357858, 12400.73788843,
       9905.24357858, 13113.95011521, 13134.21679685, 10438.9647275,
       14297.88848968, 9980.19357928, 13814.81313288, 10525.51487005,
       9963.58945501, 9987.30191343, 16215.73547458, 10682.97748635,
       10104.83122541, 13228.02396614, 12438.34126287, 10595.41988835,
       12257.76198283, 18893.08597487, 11319.31962382, 10656.69413361,
       10116.77706815, 10047.11035003, 9907.46024688, 14202.73008606,
       10152.24376102, 9986.27274601, 9905.48107875, 9923.76859226,
       10378.10642784, 11940.8575821 , 12058.974336 , 9905.40191203,
       13883.45068357, 12555.5872028 , 13441.22535694, 9996.20614576,
       9930.33943045, 13869.75484012, 9905.56024548, 9919.81025601,
       13482.86705437, 9905.24357858])
```

```
[162] # Mengembalikan skala nilai visitor dari y_train_pred
y_train_pred_copies = np.repeat(y_train_pred, X_train.shape[-1], axis=-1)
y_train_pred_real = scaler.inverse_transform(y_train_pred_copies)[:,1]
y_train_pred_real

array([13690.836 , 10711.437 , 12676.642 , 12572.095 , 10778.78 ,
       9354.266 , 13368.047 , 9689.323 , 13969.2295, 9621.537 ,
       9851.476 , 13573.864 , 9574.38 , 11940.264 , 10015.695 ,
       12739.594 , 9735.951 , 12995.626 , 13675.318 , 10769.598 ,
       14019.67 , 9645.033 , 13876.9375, 12266.971 , 9443.491 ,
       9757.54 , 10854.008 , 10678.647 , 9377.515 , 13257.765 ,
       12398.408 , 10762.628 , 12815.262 , 10700.349 , 13131.417 ,
       10784.53 , 9398.647 , 9422.696 , 9829.791 , 12809.0625,
       9394.873 , 9796.644 , 9712.528 , 9511.186 , 11136.28 ,
       12758.401 , 12801.184 , 10150.689 , 13658.186 , 12618.342 ,
       13291.264 , 9417.594 , 9463.596 , 13949.532 , 9778.377 ,
       9534.225 , 12554.413 , 10364.494 ], dtype=float32)
```

Gambar 11. Inverse GRU

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
[163] # Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_70_train = mean_squared_error(y_train_real, y_train_pred_real)
mae_70_train = mean_absolute_error(y_train_real, y_train_pred_real)
mape_70_train = mean_absolute_percentage_error(y_train_real, y_train_pred_real)

print(f"Mean Absolute Error (MAE): {round(mae_70_train, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_70_train, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_70_train, 2)}%")
```

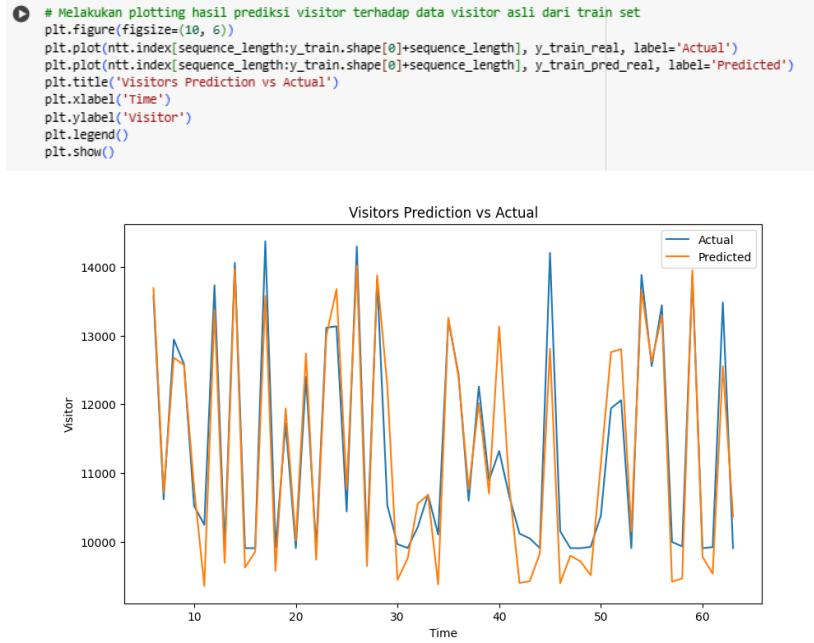
```
Mean Absolute Error (MAE): 390.04
Mean Squared Error (MSE): 302833.99
Mean Absolute Percentage Error (MAPE): 0.03%
```

Gambar 11. Hasil MSE,MAE, dan MAPE GRU

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model

GRU mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model GRU dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.



g. Evaluate Model (Test)

Selanjutnya yaitu tahap prediksi pada model data test, dimana data test yang akan diprediksi yaitu X_test, dilanjut dengan penerapan fungsi scaler Inverse untuk mengembalikan nilai-nilai total_visitor dari y_test hingga memplot hasil Test Set.

```
[170] # Membuat prediksi visitor menggunakan X_test
y_test_pred = model.predict(X_test)
1/1 [=====] - 0s 60ms/step

[171] # Mengembalikan skala nilai visitor dari y_test
y_test_copies = np.repeat(y_test.reshape(-1, 1), X_test.shape[-1], axis=-1)
y_test_real = scaler.inverse_transform(y_test_copies)[:,1]
y_test_real
array([ 9923.13525846, 12306.05368516, 9907.30191343, 13629.0879957 ,
       12615.12858011, 9916.48525355, 14780.96384648, 13667.24635722,
      13331.73777607, 9917.2769288 , 10495.58984795, 10899.65681306,
      12860.66242769, 10114.7187333 , 9907.69774706, 9908.01441396,
      12576.17055134, 10000.48114892, 10614.65660256, 10457.74815333,
      15022.50152488, 12744.16234209, 10666.66914097, 12529.54135023,
     14539.42616808, 9911.02274951])
```

```
[172] # Mengembalikan skala nilai visitor dari y_train_pred
y_test_pred_copies = np.repeat(y_test_pred, X_test.shape[-1], axis=-1)
y_test_pred_real = scaler.inverse_transform(y_test_pred_copies)[:,1]
y_test_pred_real
array([ 9487.088 , 14157.745 , 9813.652 , 13517.749 , 14188.463 ,
       9553.626 , 14112.556 , 13378.602 , 13444.372 , 9667.585 ,
      10770.218 , 10661.5205, 12244.852 , 9391.329 , 9882.991 ,
      9934.02 , 13378.644 , 9431.614 , 10781.71 , 10707.828 ,
     14146.798 , 12446.752 , 10696.269 , 12652.289 , 14072.632 ,
     9597.033 ], dtype=float32)
```

Gambar 11. Inverse GRU

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
[173] # Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_70_test = mean_squared_error(y_test_real, y_test_pred_real)
mae_70_test = mean_absolute_error(y_test_real, y_test_pred_real)
mape_70_test = mean_absolute_percentage_error(y_test_real, y_test_pred_real)

print(f"Mean Absolute Error (MAE): {round(mae_70_test, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_70_test, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_70_test, 2)}%")

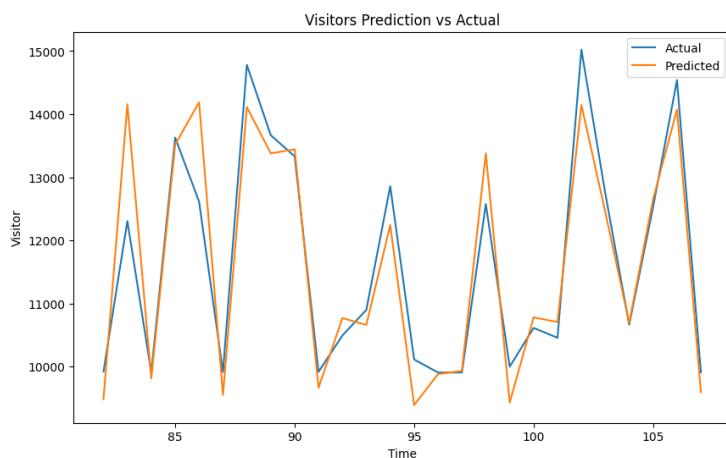
Mean Absolute Error (MAE): 444.04
Mean Squared Error (MSE): 389738.75
Mean Absolute Percentage Error (MAPE): 0.04%
```

Gambar 11. MAE,MSE,MAPE GRU 70 %

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model GRU mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model GRU dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.

```
[42] # Melakukan plotting hasil prediksi visitor terhadap data visitor asli dari train set
plt.figure(figsize=(10, 6))
plt.plot(ntt.index[-y_test.shape[0]:], y_test_real, label='Actual')
plt.plot(ntt.index[-y_test.shape[0]:], y_test_pred_real, label='Predicted')
plt.title('Visitors Prediction vs Actual')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()
```



- Skenario 60 % Train

Selanjutnya yaitu, melakukan analisis GRU dengan Skenario 80 % Train dan 20 % test dengan melalui beberapa tahapan dibawah ini :

a. Splitting Train and Test Data

Membagi data pelatihan dan pengujian dengan rasio 80% dan 20% serta menetapkan hasilnya ke dalam variabel baru, seperti yang ditunjukkan di bawah ini:

```
# Getting trainX and trainY
trainX = []
trainY = []
n_future = 1
n_past = 24

for i in range(n_past, len(scaled_data_train) - n_future + 1):
    trainX.append(scaled_data_train[i - n_past:i, 0:scaled_data_train.shape[1]])
    trainY.append(scaled_data_train[i + n_future - 1:i + n_future, 0])

trainX = np.array(trainX)
trainY = np.array(trainY)

# Memisahkan data menjadi train dan test
X_train, X_test, y_train, y_test = train_test_split(trainX, trainY, test_size=0.4, random_state=42)

print('X_train shape == {}.'.format(X_train.shape))
print('y_train shape == {}.'.format(y_train.shape))
print('X_test shape == {}.'.format(X_test.shape))
print('y_test shape == {}.'.format(y_test.shape))

X_train shape == (50, 24, 2).
y_train shape == (50, 1).
X_test shape == (34, 24, 2).
y_test shape == (34, 1).
```

Gambar 11. Splitting train & Test GRU 60 %

Dalam penjelasan di atas, yaitu menghitung ukuran data pelatihan berdasarkan proporsi tertentu dari total urutan. Selanjutnya, menggunakan nilai tersebut untuk memisahkan urutan menjadi data pelatihan dan pengujian yang menjadi variabel baru Variabel X_train, X_test, y_train, dan y_test kemudian digunakan untuk menyimpan data dan label yang sesuai. Lalu, melakukan print untuk menampilkan ukuran dari data pelatihan dan pengujian untuk memastikan proses pemisahan berjalan dengan baik.

b. Create Model

Tahap Selanjutnya, yaitu memanggil model GRU untuk membuat kerangka kerja model gru dengan menggunakan beberapa parameter yang telah ditentukan pada kode berikut.

```
# Define the GRU model
model = Sequential()
model.add(GRU(64, activation = 'tanh', input_shape = (trainX.shape[1], trainX.shape[2])))
model.add(Dense(trainY.shape[1]))

model.summary()
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

Model: "sequential_5"
=====
Layer (type)          Output Shape         Param #
=====
gru_5 (GRU)           (None, 64)          13056
dense_5 (Dense)       (None, 1)           65
=====
Total params: 13121 (51.25 KB)
Trainable params: 13121 (51.25 KB)
Non-trainable params: 0 (0.00 Byte)
```

Gambar 11. Model GRU 60 %

Pertama, sebuah objek model sequential dibuat menggunakan Sequential(). Model ini adalah wadah linear untuk lapisan-lapisan neural network. Kemudian, lapisan GRU ditambahkan ke dalam model menggunakan model.add(GRU(64, activation='tanh', input_shape=(trainX.shape[1], trainX.shape[2]))). Lapisan GRU ini memiliki 64 unit (neuron) dengan fungsi aktivasi tangen hiperbolik (tanh) dan mendefinisikan bentuk input

dengan parameter input_shape sesuai dengan dimensi data time series yang diharapkan.

Terakhir, sebuah lapisan Dense ditambahkan dengan menggunakan model.add(Dense(trainY.shape[1])). Lapisan Dense ini bertanggung jawab untuk menghasilkan output dengan jumlah dimensi yang sesuai dengan dimensi target, yang dalam konteks ini sesuai dengan trainY.shape[1]. Dengan menambahkan lapisan ini, model memiliki kemampuan untuk menghasilkan prediksi yang sesuai dengan struktur target yang diharapkan.

c. Fit Model

```
# Define the GRU model
model = Sequential()
model.add(GRU(64, activation = 'tanh', input_shape = (trainX.shape[1], trainX.shape[2])))
model.add(Dense(trainY.shape[1]))

model.summary()
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

Model: "sequential_5"
-----  
Layer (type)      Output Shape     Param #  
=====  
gru_5 (GRU)      (None, 64)       13056  
dense_5 (dense)  (None, 1)        65  
=====  
Total params: 13121 (51.25 KB)  
Trainable params: 13121 (51.25 KB)  
Non-trainable params: 0 (0.00 Byte)
```

Gambar 11. Model GRU 60 %

Pada kode tersebut, trainX dan trainY adalah input dan output yang digunakan untuk melatih model. Parameter epochs=100 menentukan berapa kali model akan melihat seluruh dataset pada proses pelatihan. Batch size, ditentukan oleh batch_size=32, menentukan seberapa banyak sampel data yang digunakan pada setiap iterasi pembelajaran. Semakin kecil nilai batch size, semakin sering pembelajaran akan diperbarui, tetapi akan membutuhkan lebih banyak waktu komputasi. ini bertujuan untuk menghasilkan model GRU yang dapat memahami dan memprediksi pola dalam data time series dengan lebih baik.

Selama proses pelatihan, informasi pelatihan disimpan dalam objek history. Hasil pelatihan tersebut bisa digunakan untuk mengevaluasi performa model pada setiap epoch, melihat perubahan nilai loss function, dan menganalisis bagaimana model beradaptasi terhadap data pelatihan. Parameter verbose=2 menampilkan progres pelatihan secara terperinci, sedangkan nilai 0 akan membuatnya tampil secara diam-diam.

d. Loss Model

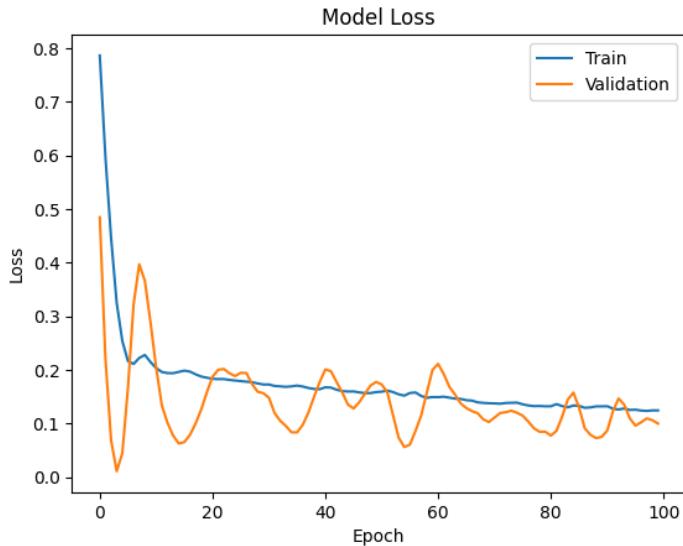
Tahap selanjutnya yaitu, mengenai Loss Model. Loss Model sendiri adalah fungsi matematis yang menyederhanakan perbedaan antara prediksi model dengan nilai sebenarnya. Sebagai contoh, dalam regresi, fungsi kerugian umumnya adalah Mean Squared Error (MSE), yang menghitung rata-rata dari kuadrat selisih antara prediksi dan ground truth. Proses pelatihan model melibatkan iteratif memperbarui parameter-parameter model untuk

meminimalkan nilai loss ini. Sebagai penerapan loss model, maka dibuat seperti berikut, dimana loss model diambil dari Fit GRU diatas.

```
[ ] # Plot the training loss  
plt.plot(history.history['loss'], label='Training loss')  
plt.legend()
```

Gambar 11. Loss Model GRU 60 %

Berdasarkan plot pada gambar diatas, dapat diartikan sebagai pemberian label, legenda berdasarkan data history yang menghasilkan plot train loss dan val loss seperti berikut



Gambar 11. Hasil Loss Model GRU 60 %

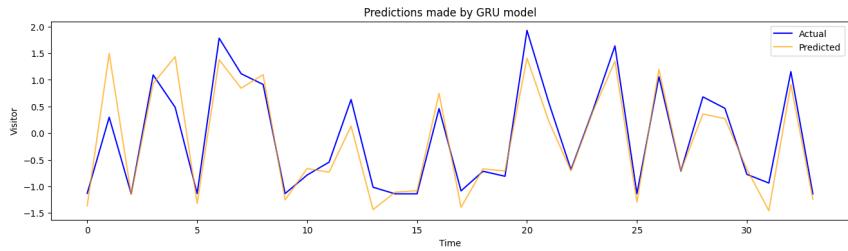
h. Prediction

Selanjutnya yaitu, melakukan prediksi terhadap model untuk X_test, dimana prediction sendiri dalam GRU melibatkan mengalirkan input melalui model dan memperoleh output prediksi. Hasil ini kemudian dapat dibandingkan dengan nilai sebenarnya untuk mengevaluasi seberapa baik model dapat melakukan prediksi. Pada umumnya, hasil prediksi GRU dapat digunakan untuk berbagai aplikasi, termasuk prediksi time series, pengenalan pola, dan tugas-tugas lain yang melibatkan urutan data.

```
[25] # Melakukan prediksi pada test set  
predictions = model.predict(x_test)  
  
1/1 [=====] - 0s 445ms/step
```

Gambar 11. Prediksi Model GRU 60 %

Berdasarkan gambar diatas, maka prediksi berhasil dijalankan dengan menggunakan fungsi predict dan menghasilkan plot seperti dibawah ini.



Gambar 11. Prediksi Model GRU 60 %

i. Evaluate Model (Train)

Selanjutnya yaitu tahap prediksi pada model data train, dimana data train yang akan diprediksi yaitu X_train, dilanjut dengan penerapan fungsi scaler Inverse untuk mengembalikan nilai-nilai total_visitor dari y_train hingga memplot hasil Train Set.

```
' [190] # Membuat prediksi visitor menggunakan X_train
y_train_pred = model.predict(X_train)
2/2 [=====] - 0s 7ms/step

' [191] # Mengembalikan skala nilai visitor dari y_train
y_train_copies = np.repeat(y_train.reshape(-1, 1), X_train.shape[-1], axis=-1)
y_train_real = scaler.inverse_transform(y_train_copies)[:,1]
y_train_real

array([14056.35081128, 9906.19357928, 9905.24357858, 14371.67187751,
       9910.31024899, 11717.68658393, 9905.24357858, 12400.73708843,
       9905.24357858, 13113.95011521, 13124.21679685, 10438.90647725,
       14297.88848968, 9906.19357928, 13814.81313288, 10525.51487005,
       9963.58945501, 9907.30191343, 10215.73547458, 10682.97748635,
       10104.03122541, 13228.02936614, 12438.34128287, 10595.41908835,
       12257.76198283, 10893.08597487, 11319.31962302, 10656.69413361,
       10116.77706815, 10047.11035003, 9907.46024688, 14202.73008606,
       10152.24376102, 9906.27274601, 9905.48107875, 9923.76859226,
       10378.10642784, 11940.8575821, 12058.974336, 9905.40191203,
       13883.45068357, 12555.5872028, 13441.22535694, 9996.20614576,
       9930.33943045, 13869.75484012, 9905.56024548, 9919.81025601,
       13482.86705437, 9905.24357858])

' [192] # Mengembalikan skala nilai visitor dari y_train_pred
y_train_pred_copies = np.repeat(y_train_pred, X_train.shape[-1], axis=-1)
y_train_pred_real = scaler.inverse_transform(y_train_pred_copies)[:,1]
y_train_pred_real

array([13959.178, 9671.847, 9930.026, 13430.484, 9625.431,
       11709.831, 10070.89, 12684.3125, 9798.898, 13196.476,
       13579.377, 10654.68, 14007.476, 9695.276, 13981.963,
       12103.259, 9492.569, 9817.295, 10498.686, 10599.507,
       9399.624, 13319.527, 12294.366, 10683.184, 11886.794,
       10615.018, 13001.806, 10685.661, 9428.11, 9469.059,
       9908.516, 12525.087, 9431.081, 9868.381, 9764.673,
       9563.875, 10925.424, 12524.859, 12861.479, 10176.959,
       13477.185, 12750.385, 13155.369, 9455.875, 9511.621,
       13803.619, 9844.191, 9588.302, 12556.128, 10346.597],
       dtype=float32)
```

Gambar 11. Inverse GRU

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
# Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_60_train = mean_squared_error(y_train_real, y_train_pred_real)
mae_60_train = mean_absolute_error(y_train_real, y_train_pred_real)
mape_60_train = mean_absolute_percentage_error(y_train_real, y_train_pred_real)

print(f"Mean Absolute Error (MAE): {round(mae_60_train, 2)}")
print(f"Mean Squared Error (MSE): {round(mse_60_train, 2)}")
print(f"Mean Absolute Percentage Error (MAPE): {round(mape_60_train, 2)}%")

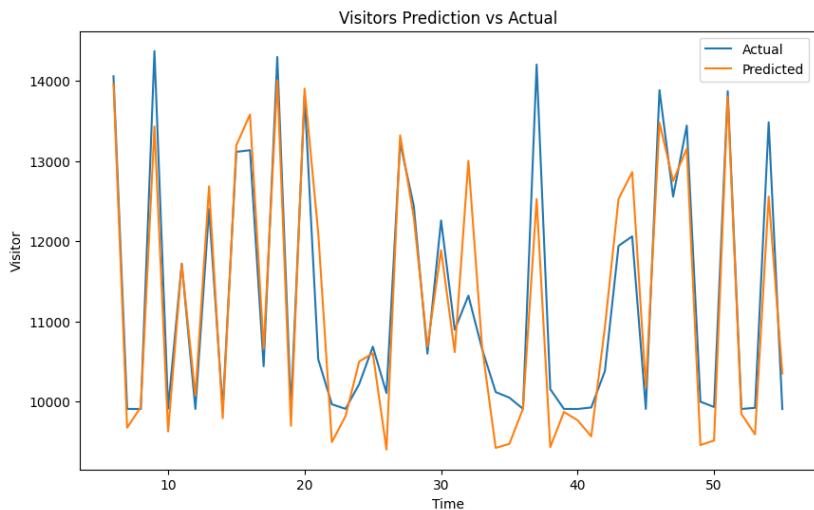
Mean Absolute Error (MAE): 389.42
Mean Squared Error (MSE): 310273.04
Mean Absolute Percentage Error (MAPE): 0.03%
```

Gambar 11. Hasil MSE, MAE, dan MAPE GRU

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model GRU mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model GRU dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.

```
# Melakukan plotting hasil prediksi visitor terhadap data visitor asli dari train set
plt.figure(figsize=(10, 6))
plt.plot(ntt.index[sequence_length:y_train.shape[0]+sequence_length], y_train_real, label='Actual')
plt.plot(ntt.index[sequence_length:y_train.shape[0]+sequence_length], y_train_pred_real, label='Predicted')
plt.title('Visitors Prediction vs Actual')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()
```



j. Evaluate Model (Test)

Selanjutnya yaitu tahap prediksi pada model data test, dimana data test yang akan diprediksi yaitu X_test, dilanjut dengan penerapan fungsi scaler Inverse untuk mengembalikan nilai-nilai total_visitor dari y_test hingga memplot hasil Test Set.

```
[200] # Membuat prediksi visitor menggunakan x_test
y_test_pred = model.predict(X_test)
2/2 [=====] - 0s 10ms/step

[201] # Mengembalikan skala nilai visitor dari y_test
y_test_copies = np.repeat(y_test.reshape(-1, 1), X_test.shape[-1], axis=-1)
y_test_real = scaler.inverse_transform(y_test_copies)[:,1]
y_test_real
array([ 9923.13525846, 12306.05368516, 9907.30191343, 13629.0879957,
       12615.12058011, 9916.48525355, 14780.96384648, 13667.24635722,
       13331.73777607, 9917.2769208, 10495.58984795, 10899.65681306,
       12860.06242769, 10114.7187333, 9907.69774706, 9908.01441396,
       12576.17055134, 10000.48114892, 10614.65660256, 10457.74815333,
       15022.50152488, 12744.16234209, 10666.66914097, 12529.54135023,
       14539.42616808, 9911.02274951, 13573.27545448, 10617.26910449,
       12942.39582184, 12585.35389146, 10517.28153063, 10245.66049668,
       13731.37140458, 9906.27274601])

[202] # Mengembalikan skala nilai visitor dari y_train_pred
y_test_pred_copies = np.repeat(y_test_pred, X_test.shape[-1], axis=-1)
y_test_pred_real = scaler.inverse_transform(y_test_pred_copies)[:,1]
y_test_pred_real
array([ 9536.407, 14302.66 , 9890.332, 13360.537, 14200.701, 9606.347,
       14105.089, 13213.908, 13638.219, 9719.348, 18700.039, 10586.139,
       12028.161, 9411.153, 9960.454, 10006.026, 13057.449, 9481.131,
       10689.562, 10620.915, 14151.293, 12162.576, 10629.148, 12486.643,
       14059.131, 9647.885, 13805.113, 10627.775, 12407.655, 12261.936,
       10679.031, 9380.467, 13328.329, 9740.393], dtype=float32)
```

Gambar 11. Inverse GRU

Setelah hasil nilai atau value visitor diubah menjadi nilai awal menggunakan transform inverse berhasil dieksekusi, maka dapat diprediksi MSE, MAE, dan MAPE sesuai dengan format import library sklearn untuk hitung 3 indikator tersebut.

```
[203] # Menghitung nilai ERROR dari hasil prediksi visitor terhadap data visitor asli dari train set
mse_60_test = mean_squared_error(y_test_real, y_test_pred_real)
mae_60_test = mean_absolute_error(y_test_real, y_test_pred_real)
mape_60_test = mean_absolute_percentage_error(y_test_real, y_test_pred_real)

print("Mean Absolute Error (MAE): {round(mae_60_test, 2)}")
print("Mean Squared Error (MSE): {round(mse_60_test, 2)}")
print("Mean Absolute Percentage Error (MAPE): {round(mape_60_test, 2)}%")

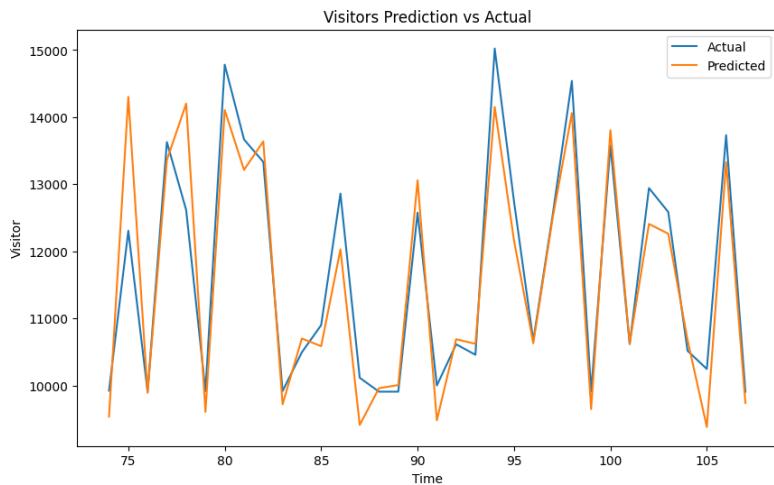
Mean Absolute Error (MAE): 429.83
Mean Squared Error (MSE): 361668.46
Mean Absolute Percentage Error (MAPE): 0.04%
```

Gambar 11. MAE,MSE,MAPE GRU 60 %

Grafik aktual vs prediksi tersebut menunjukkan hasil prediksi jumlah pengunjung ke provinsi NTT. Grafik tersebut menunjukkan bahwa model GRU mampu memprediksi tren data aktual dengan cukup akurat. Namun, ada beberapa titik data aktual yang tidak dapat diprediksi dengan sempurna oleh model. Hal ini dapat dilihat dari adanya selisih antara nilai aktual dan nilai prediksi pada beberapa titik data.

Secara umum, grafik tersebut menunjukkan bahwa model GRU dapat digunakan untuk memprediksi jumlah pengunjung ke situs web dengan cukup akurat. Namun, perlu dilakukan evaluasi lebih lanjut untuk memastikan akurasi prediksi model.

```
[42] # Melakukan plotting hasil prediksi visitor terhadap data visitor asli dari train set
plt.figure(figsize=(10, 6))
plt.plot(ntt.index[-y_test.shape[0]:], y_test_real, label='Actual')
plt.plot(ntt.index[-y_test.shape[0]:], y_test_pred_real, label='Predicted')
plt.title('Visitors Prediction vs Actual')
plt.xlabel('Time')
plt.ylabel('Visitor')
plt.legend()
plt.show()
```



Scenario	Model	MSE	MAE	MAPE
Scenario 1	GRU Train 80	303187	408.392	0.0360969
Scenario 1	GRU Test 80	512361	541.215	0.0454427
Scenario 2	GRU Train 70	302034	390.043	0.0349754
Scenario 2	GRU Test 70	389739	444.042	0.037082
Scenario 3	GRU Train 60	310273	389.415	0.0343761
Scenario 3	GRU Test 60	361668	429.827	0.0356815

MAPE sering kali efektif untuk menganalisis kumpulan data yang besar dan membutuhkan penggunaan nilai kumpulan data selain nol. Nilai MAPE lebih mudah diinterpretasikan dibandingkan alat ukur yang lain karena berupa nilai persen yang dapat terukur skalanya. Berdasarkan nilai MAPE, skenario terbaik pada model SVR adalah skenario 3 dengan pembagian 60% data latih (train) dan 40% data uji (test).

V. Hasil Forecast Untuk 12 Periode

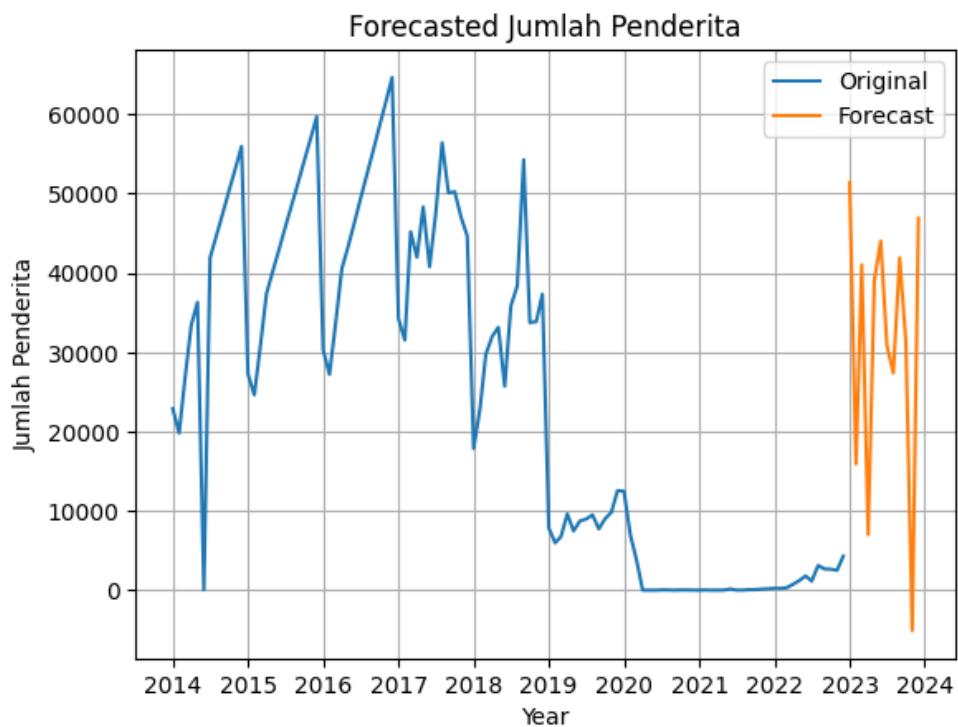
Langkah berikutnya adalah menganalisis hasil prediksi untuk 12 periode mendatang berdasarkan evaluasi model menggunakan tiga metode deep learning. Dengan merinci hasil analisis, fokus akan diberikan pada kemampuan masing-masing metode, yaitu RNN, LSTM, dan GRU, dalam meramalkan data untuk periode yang akan datang. Analisis ini bertujuan untuk mendapatkan wawasan yang lebih mendalam tentang sejauh mana keakuratan dan kemampuan prediktif dari setiap metode dalam menghadapi tantangan peramalan untuk 12 periode mendatang.

1. SVR

- Skenario 80 % Train

Date	Prediction
2023-01-01 00:00:00	51384
2023-02-01 00:00:00	15918
2023-03-01 00:00:00	41030
2023-04-01 00:00:00	7006
2023-05-01 00:00:00	39085
2023-06-01 00:00:00	44032
2023-07-01 00:00:00	30846
2023-08-01 00:00:00	27384
2023-09-01 00:00:00	41927
2023-10-01 00:00:00	31391
2023-11-01 00:00:00	-5101
2023-12-01 00:00:00	46896

Gambar Tabel Prediksi 12 Bulan Kedepan SVR 80%

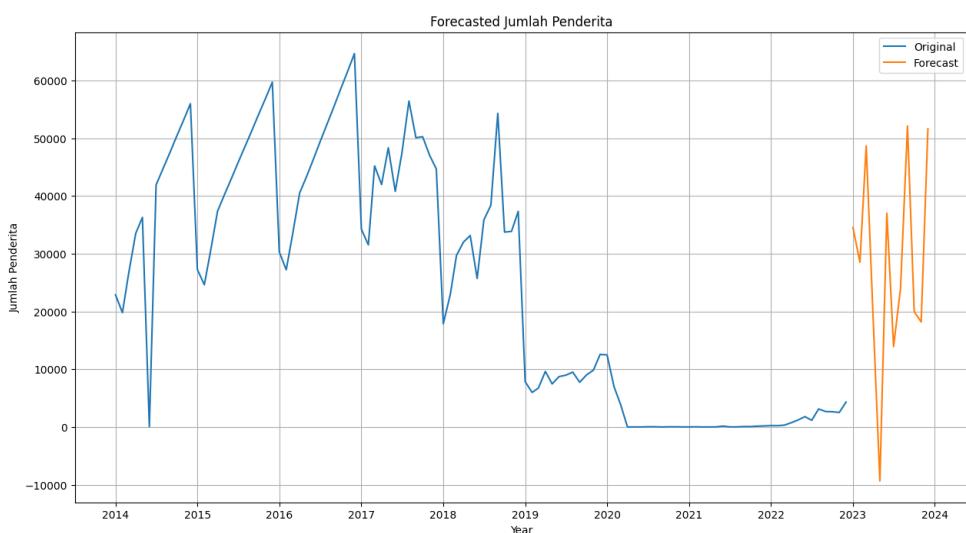


Gambar Grafik Prediksi 12 Bulan Kedepan SVR 80%

- Skenario 70 % Train

Date	Prediction
2023-01-01 00:00:00	34533
2023-02-01 00:00:00	28510
2023-03-01 00:00:00	48673
2023-04-01 00:00:00	18718
2023-05-01 00:00:00	-9348
2023-06-01 00:00:00	37013
2023-07-01 00:00:00	13919
2023-08-01 00:00:00	23979
2023-09-01 00:00:00	52109
2023-10-01 00:00:00	19963
2023-11-01 00:00:00	18170
2023-12-01 00:00:00	51626

Gambar Tabel Prediksi 12 Bulan Kedepan SVR 70%

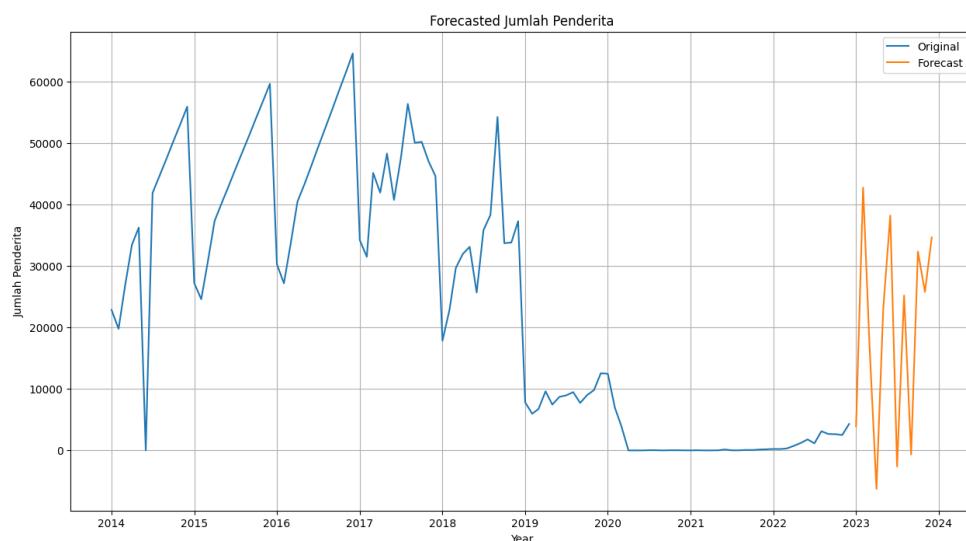


Gambar Grafik Prediksi 12 Bulan Kedepan SVR 70%

- Skenario 60 % Train

Date	Prediction
2023-01-01 00:00:00	3907
2023-02-01 00:00:00	42799
2023-03-01 00:00:00	17305
2023-04-01 00:00:00	-6252
2023-05-01 00:00:00	23142
2023-06-01 00:00:00	38250
2023-07-01 00:00:00	-2623
2023-08-01 00:00:00	25232
2023-09-01 00:00:00	-663
2023-10-01 00:00:00	32370
2023-11-01 00:00:00	25806
2023-12-01 00:00:00	34687

Gambar Tabel Prediksi 12 Bulan Kedepan SVR 60%



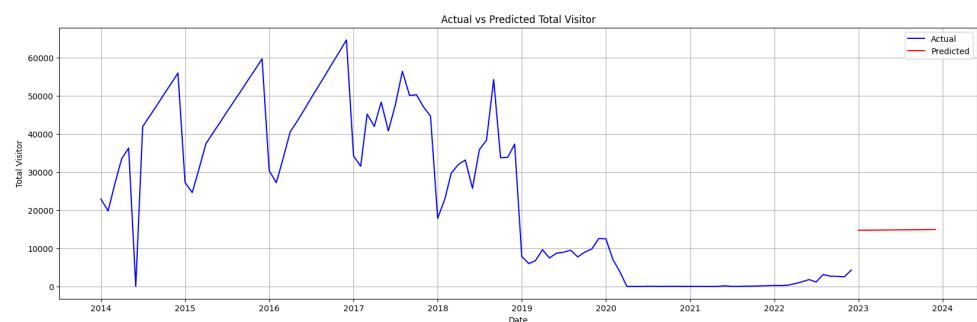
Gambar Grafik Prediksi 12 Bulan Kedepan SVR 60%

2. RNN

- Skenario 80 % Train

Date	Prediction
2023-01-01 00:00:00	14718
2023-02-01 00:00:00	14743
2023-03-01 00:00:00	14759
2023-04-01 00:00:00	14778
2023-05-01 00:00:00	14796
2023-06-01 00:00:00	14815
2023-07-01 00:00:00	14843
2023-08-01 00:00:00	14840
2023-09-01 00:00:00	14863
2023-10-01 00:00:00	14881
2023-11-01 00:00:00	14902
2023-12-01 00:00:00	14928

Gambar Tabel Prediksi 12 Bulan Kedepan RNN 80%

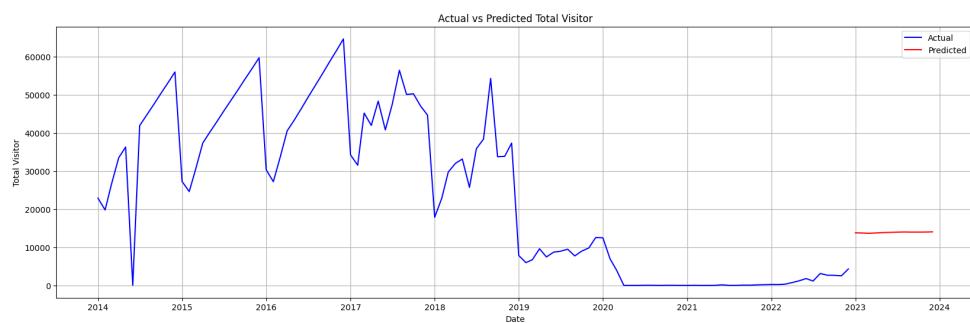


Gambar Grafik Prediksi 12 Bulan Kedepan RNN 80%

- Skenario 70 % Train

Date	Prediction
2023-01-01 00:00:00	13793
2023-02-01 00:00:00	13741
2023-03-01 00:00:00	13665
2023-04-01 00:00:00	13761
2023-05-01 00:00:00	13843
2023-06-01 00:00:00	13912
2023-07-01 00:00:00	13970
2023-08-01 00:00:00	14007
2023-09-01 00:00:00	13979
2023-10-01 00:00:00	13983
2023-11-01 00:00:00	13992
2023-12-01 00:00:00	14043

Gambar Tabel Prediksi 12 Bulan Kedepan RNN 70%

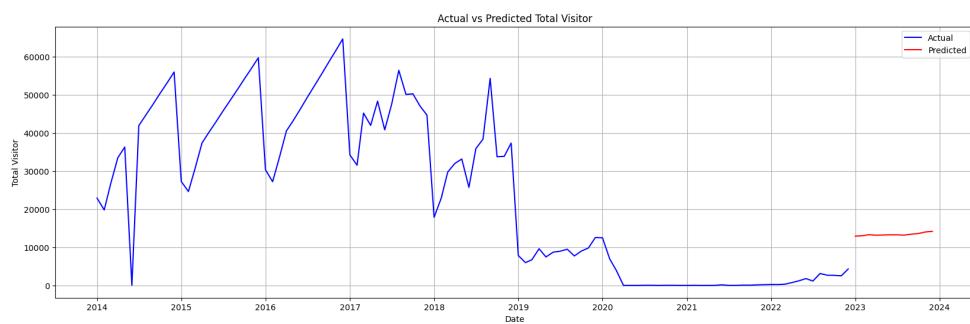


Gambar Grafik Prediksi 12 Bulan Kedepan RNN 70%

- Skenario 60 % Train

Date	Prediction
2023-01-01 00:00:00	12917
2023-02-01 00:00:00	13032
2023-03-01 00:00:00	13293
2023-04-01 00:00:00	13175
2023-05-01 00:00:00	13220
2023-06-01 00:00:00	13272
2023-07-01 00:00:00	13261
2023-08-01 00:00:00	13183
2023-09-01 00:00:00	13445
2023-10-01 00:00:00	13598
2023-11-01 00:00:00	14011
2023-12-01 00:00:00	14183

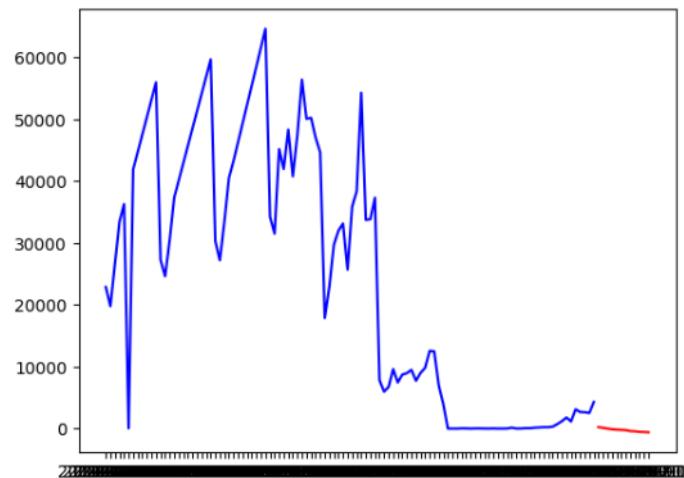
Gambar Tabel Prediksi 12 Bulan Kedepan RNN 60%



Gambar Grafik Prediksi 12 Bulan Kedepan RNN 60%

3. LSTM

- Skenario 80 % Train



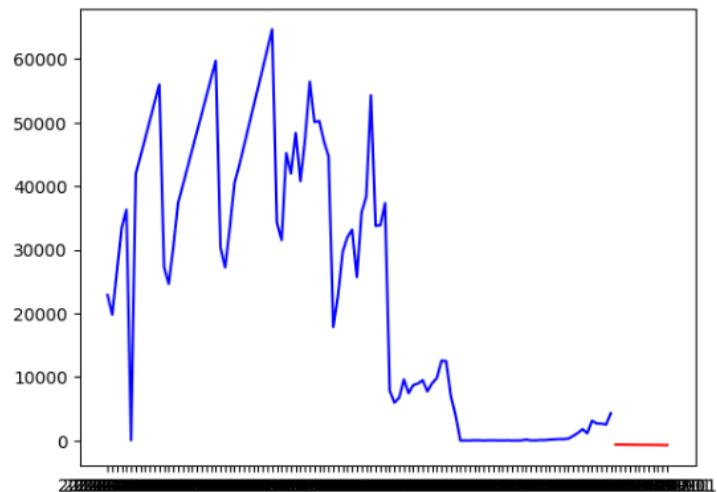
Gambar xx. Hasil Plot Prediksi 12 Bulan

```
# Mengonversi indeks result_df menjadi kolom 'Periode'  
res20_df['Datetime'] = res20_df.index  
  
# Menggabungkan DataFrame result_df dan resdate_df menggunakan pd.concat  
result20 = pd.concat([resdate_df, res20_df[['Prediksi Jumlah Pengunjung']]], axis=1)  
  
result20
```

	Datetime	Prediksi Jumlah Pengunjung	Actions
0	2023-01-01	236	grid
1	2023-02-01	126	grid
2	2023-03-01	-1	grid
3	2023-04-01	-106	grid
4	2023-05-01	-169	grid
5	2023-06-01	-215	grid
6	2023-07-01	-252	grid
7	2023-08-01	-407	grid
8	2023-09-01	-440	grid
9	2023-10-01	-531	grid
10	2023-11-01	-554	grid
11	2023-12-01	-604	grid

Gambar xx. Gabungan Data Frame Hasil Prediksi dan Periode

- Skenario 70 % Train



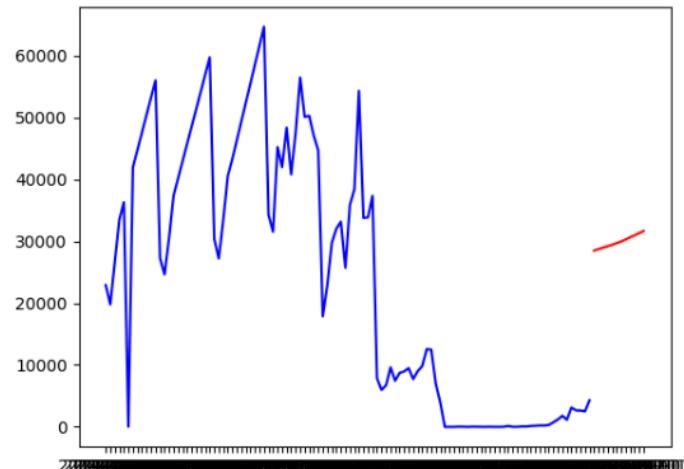
Gambar xx. Hasil Plot Prediksi 12 Bulan

```
# Mengonversi indeks result_df menjadi kolom 'Periode'  
res30_df['Datetime'] = res30_df.index  
  
# Menggabungkan DataFrame result_df dan resdate_df menggunakan pd.concat  
result30 = pd.concat([resdate_df, res30_df[['Prediksi Jumlah Pengunjung']]], axis=1)  
  
result30
```

	Datetime	Prediksi Jumlah Pengunjung
0	2023-01-01	-644
1	2023-02-01	-647
2	2023-03-01	-656
3	2023-04-01	-664
4	2023-05-01	-673
5	2023-06-01	-680
6	2023-07-01	-682
7	2023-08-01	-685
8	2023-09-01	-692
9	2023-10-01	-702
10	2023-11-01	-712
11	2023-12-01	-716

Gambar xx. Gabungan Data Frame Hasil Prediksi dan Periode

- Skenario 60 % Train



Gambar Grafik Prediksi 12 Bulan Kedepan GRU 60%

```
# Mengonversi indeks result_df menjadi kolom 'Periode'
res40_df['Datetime'] = res40_df.index

# Menggabungkan DataFrame result_df dan resdate_df menggunakan pd.concat
result40 = pd.concat([resdate_df, res40_df[['Prediksi Jumlah Pengunjung']]], axis=1)

result40
```

	Datetime	Prediksi Jumlah Pengunjung
0	2023-01-01	28478
1	2023-02-01	28718
2	2023-03-01	28946
3	2023-04-01	29182
4	2023-05-01	29405
5	2023-06-01	29662
6	2023-07-01	29952
7	2023-08-01	30257
8	2023-09-01	30624
9	2023-10-01	30951
10	2023-11-01	31297
11	2023-12-01	31657

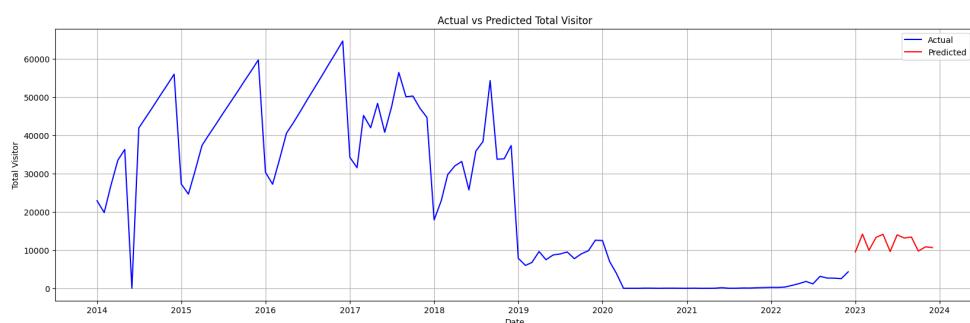
Gambar Tabel Prediksi 12 Bulan Kedepan LSTM 60%

4. GRU

- Skenario 80 % Train

Date	Prediction
2023-01-01 00:00:00	9513
2023-02-01 00:00:00	14156
2023-03-01 00:00:00	9925
2023-04-01 00:00:00	13308
2023-05-01 00:00:00	14104
2023-06-01 00:00:00	9602
2023-07-01 00:00:00	13973
2023-08-01 00:00:00	13133
2023-09-01 00:00:00	13398
2023-10-01 00:00:00	9727
2023-11-01 00:00:00	10827
2023-12-01 00:00:00	10645

Gambar Tabel Prediksi 12 Bulan Kedepan GRU 80%

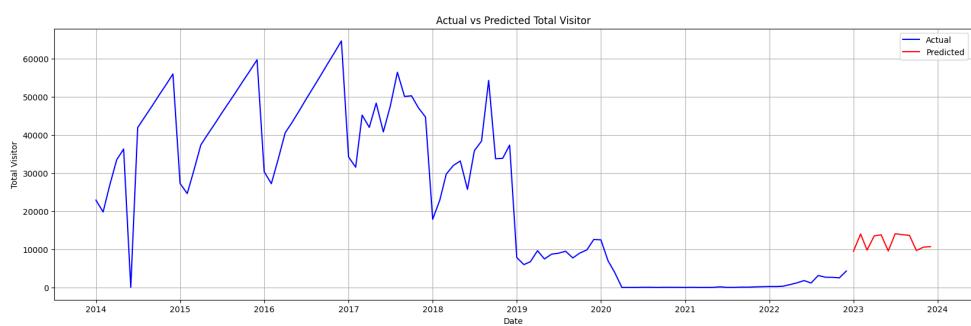


Gambar Grafik Prediksi 12 Bulan Kedepan GRU 80%

- Skenario 70 % Train

Date	Prediction
2023-01-01 00:00:00	9499
2023-02-01 00:00:00	14011
2023-03-01 00:00:00	9790
2023-04-01 00:00:00	13509
2023-05-01 00:00:00	13797
2023-06-01 00:00:00	9563
2023-07-01 00:00:00	14079
2023-08-01 00:00:00	13833
2023-09-01 00:00:00	13657
2023-10-01 00:00:00	9671
2023-11-01 00:00:00	10565
2023-12-01 00:00:00	10688

Gambar Tabel Prediksi 12 Bulan Kedepan GRU 70%

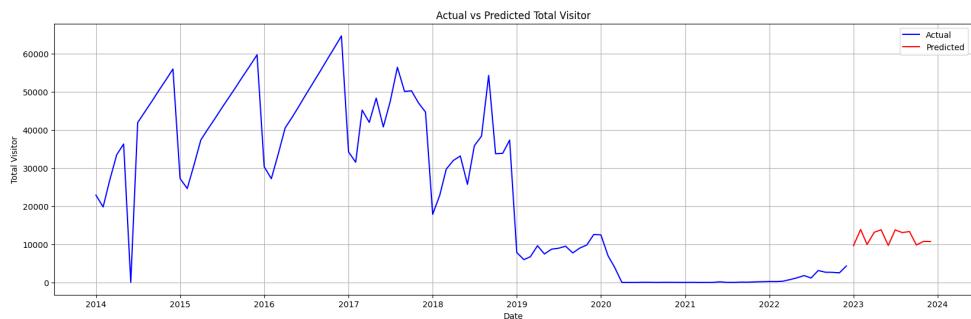


Gambar Grafik Prediksi 12 Bulan Kedepan GRU 70%

- Skenario 60 % Train

Date	Prediction
2023-01-01 00:00:00	9636
2023-02-01 00:00:00	13878
2023-03-01 00:00:00	9962
2023-04-01 00:00:00	13137
2023-05-01 00:00:00	13799
2023-06-01 00:00:00	9702
2023-07-01 00:00:00	13764
2023-08-01 00:00:00	13060
2023-09-01 00:00:00	13345
2023-10-01 00:00:00	9810
2023-11-01 00:00:00	10751
2023-12-01 00:00:00	10727

Gambar Tabel Prediksi 12 Bulan Kedepan GRU 60%



Gambar Grafik Prediksi 12 Bulan Kedepan GRU 60%

VI. Hasil Analisis dan Kesimpulan

Pada percobaan model di atas, terdapat 2 jenis model yang dicoba, yaitu SVR dan deep learning (RNN, LSTM, dan GRU). Dataset yang digunakan berupa data time series dari jumlah pengunjung dan akomodasi dari provinsi NTT. Dari keseluruhan model yang dijalankan, skenario terbaik ada pada model RNN dengan skenario 2 menggunakan pembagian 70% data latih (train) dan 30% data uji (test). Nilai MAPE untuk model tersebut adalah 0.02%

Pada model SVR, nilai parameter seperti MSE, MAE, dan MAPE cenderung sangat tinggi dari nilai parameter-parameter tersebut pada umumnya. Selain itu, beberapa hasil dari prediksi dua belas bulan kedepan menghasilkan nilai negatif padahal konteks datanya berupa jumlah pengunjung. Hal tersebut menunjukkan bahwa model tersebut mungkin tidak secara efektif menangkap pola dalam data atau mungkin perlu disesuaikan lebih lanjut.

Selanjutnya, terdapat model deep learning. Model dengan nilai MAPE yang cenderung rendah adalah RNN dan GRU. Mayoritas model deep learning

menghasilkan nilai MAPE yang rendah dengan nilai di bawah 1. Namun nilai MSE dan MAE terlihat cenderung terlalu besar. MAPE yang rendah di bawah 1% menunjukkan bahwa model deep learning mampu memberikan prediksi yang sangat akurat relatif terhadap nilai sebenarnya. MAPE memperhitungkan persentase kesalahan rata-rata dari prediksi, sehingga nilai rendah menunjukkan bahwa model cenderung memberikan prediksi yang sangat mendekati nilai sebenarnya. Akan tetapi, meskipun MAPE rendah, nilai MSE dan MAE yang tinggi menunjukkan bahwa ada variasi atau deviasi yang signifikan antara prediksi model dan nilai sebenarnya. Hal ini dapat mengindikasikan adanya outlier atau pola yang kompleks dalam data yang mungkin sulit diakomodasi oleh model.

Berdasarkan perbandingan antara SVR dan pendekatan deep learning, dapat disimpulkan bahwa model deep learning lebih efektif dalam mengelola data time series dari dataset provinsi Nusa Tenggara Timur (NTT). Meskipun deep learning memiliki keunggulan dalam menangani data time series, penting untuk diingat bahwa pemilihan model dan parameter, serta pengolahan data yang tepat, tetap krusial untuk mencapai kinerja yang optimal. Secara keseluruhan, deep learning memberikan pendekatan yang kuat dan fleksibel untuk tugas analisis data time series.