به نام خدا

عنوان:

گزارش پروژه سوم تکلیف چهارم شبکه عصبی

استاد:

دکتر منصوری

دانشجو:

محمدعلی مجتهدسلیمانی-٤٠٣٣٩٠٤٥٠٤

تاریخ:

١٤٠٣/١٠/٢٠

Table of Contents

اضافه کردن کتابخانه‌ها:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow import keras
import tensorflow as tf
from tensorflow.keras import layers
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

کتابخانه های مورد نیاز را اضافه کرده ایم. از **keras** برای پیاده سازی این پروژه استفاده کرده ایم.

از کتابخانه **numpy** برای انجام عملیات هایی عددی با ماتریس ها و آرایه ها استفاده کرده ایم.

از **pandas** برای انجام عملیات های با **data frame** ها یا داده های جدولی مخصوصا برای تحلیل داده ها استفاده کرده ایم.

با استفاده از **matplotlib** نمودار هایی را که میخواهیم نمایش میدهیم.

از کتاب خانه **sklearn** نیز استفاده کرده ایم. از **MinMaxScaler** برای مقیاس دادن به داده های خودمان برای اینکه مقادیری بین ۰ و ۱ بگیرند.

کتابخانه اصلی TensorFlow برای ساخت و آموزش مدل‌های یادگیری ماشین است که با tf اضافه کرده ایم. از statsmodels.graphics.tsplots برای رسم توابع ACFو PACF استفاده کرده ایم.

بارگذاری و پیش پردازش داده‌ها

```
df = pd.read_csv("nyc_taxi.csv")


df['timestamp'] = pd.to_datetime(df['timestamp'])


df = df.set_index('timestamp')


print(df.head())
```

```
                     value
timestamp
2014-07-01 00:00:00  10844
2014-07-01 00:30:00   8127
2014-07-01 01:00:00   6210
2014-07-01 01:30:00   4656
2014-07-01 02:00:00   3820
```

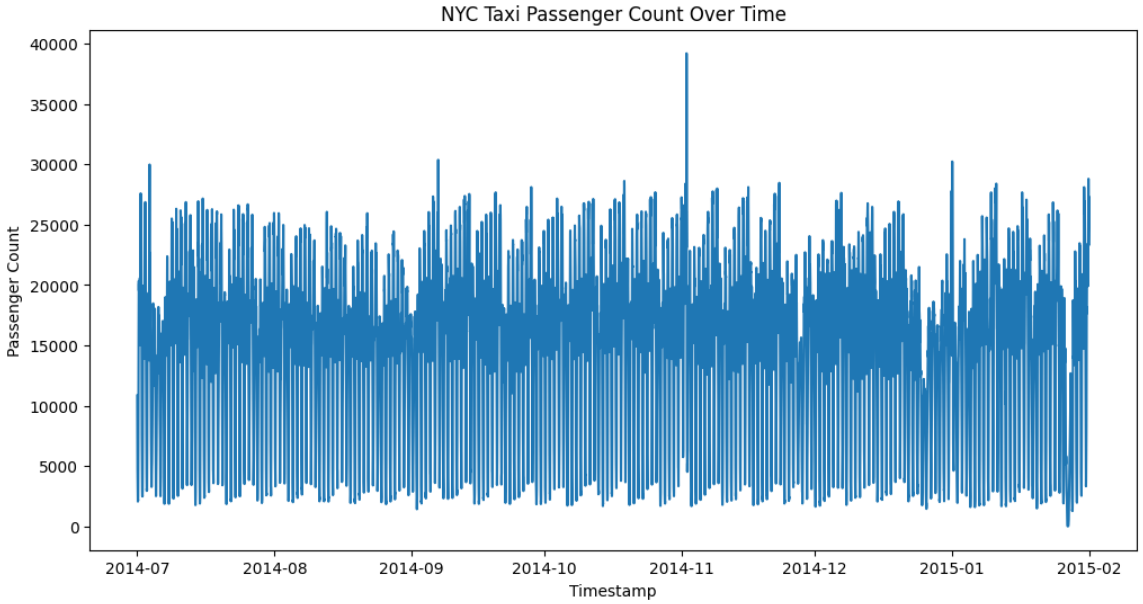
df = pd.read_csv("nyc_taxi.csv"): داده ها را از فایل اکسل وارد dataframe کتابخانه پانداز میکند.

df['timestamp'] = pd.to_datetime(df['timestamp']): ستون timestap را به object، date-time تبدیل میکند.

df = df.set_index('timestamp'): ستون timestamp را به عنوان اندیس dataframe پانداز تنظیم میکند.

در خط بعدی صرفا چند سطر اول را نمایش داده ایم تا یک نمایی از داده ها داشته باشیم.

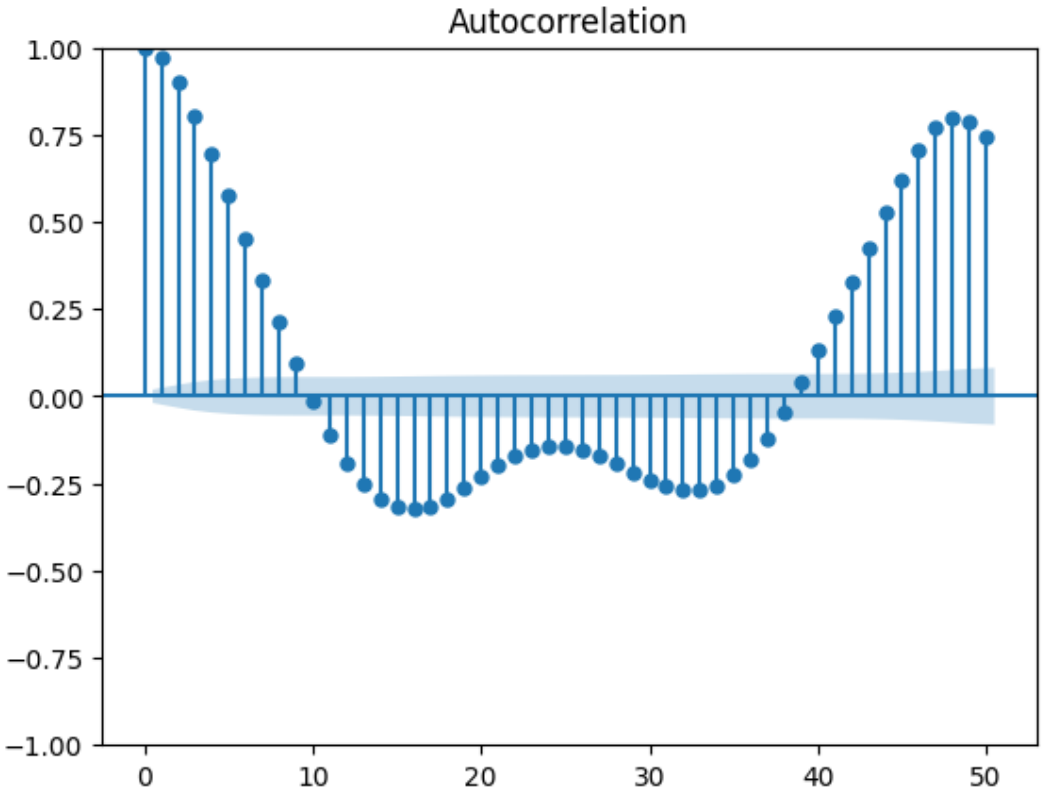
```
plt.figure(figsize=(12, 6))
plt.plot(df['value'])
plt.title('NYC Taxi Passenger Count Over Time')
plt.xlabel('Timestamp')
plt.ylabel('Passenger Count')
plt.show()
```

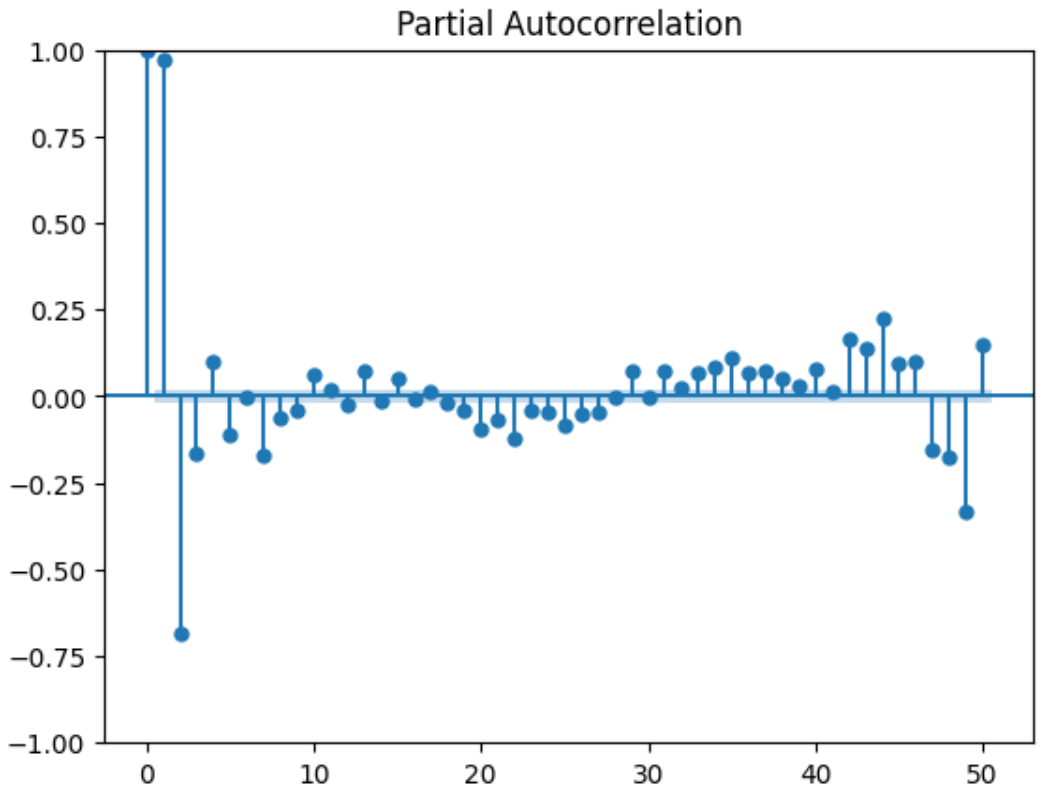


با کمک از کتابخانه matplotlib برای داده ها دیاگرام رسم کرده ایم.

```
# ACF Plot
plot_acf(df['value'], lags=50)
plt.show()

# PACF Plot
plot_pacf(df['value'], lags=50)
plt.show()
```





همچنین دیاگرام های AFC و PACF را رسم کرده ایم. این نمودار ها برای درک ساختار زیرین داده‌های سری زمانی، شناسایی الگو ها در دنباله ها، راهنمایی در انتخاب مدل های مناسب پیشبینی سری های زمانی و بررسی مانایی استفاده میشوند.

نرمال سازی داده:

```
scaler = MinMaxScaler()
df['value'] = scaler.fit_transform(df['value'].values.reshape(-1, 1))
```

scaler = MinMaxScaler(): یک object ایجاد میکند که داده ها را بین بازه ۰ و ۱ مقیاس بندی کند.

df['value'].values.reshape(-1, 1): ستون value را به یک آرایه دوبعدی تبدیل می‌کند.

scaler.fit_transform(...): مقیاس را بر داده‌ها اعمال می‌کند (حداقل و حداکثر مقادیر را پیدا می‌کند) و سپس داده‌ها را به بازه ۰ تا ۱ تبدیل می‌کند. داده‌های مقیاس‌شده به ستون value بازگردانده می‌شوند.

ایجاد دنباله برای Transformer

```
def create_sequences(data, seq_length):
    xs = []
    ys = []
    for i in range(len(data) - seq_length):
        x = data[i:(i + seq_length)]
        y = data[i + seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)


seq_length = 24


X, y = create_sequences(df['value'], seq_length)
```

create_sequences(data, seq_length): این تابع داده‌های سری زمانی و **seq_length**را به عنوان ورودی دریافت می‌کند و جفت‌های ورودی–خروجی (دنباله‌ها) را برای مدل **transformer** ایجاد می‌کند.

این تابع در داده‌ها پیمایش می‌کند و دنباله‌هایی با طول **seq_length** به عنوان ورودی **(x)**و نقطه داده بعدی را به عنوان هدف **(y)** ایجاد می‌کند.

seq_length = 24: طول دنباله را تنظیم می‌کند (تعداد گام‌های زمانی که برای پیش‌بینی استفاده می‌شود).

X, y = create_sequences(): این تابع را فراخوانی می‌کند تا دنباله‌ها را از داده‌های مقیاس‌بندی شده **'value'** ایجاد کند.

تفکیک داده های آموزشی و تست

```
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

train_size = int(len(X) * 0.8): تعداد نمونه‌ها را برای مجموعه آموزشی محاسبه می‌کند (۸۰٪ از داده‌ها).

X_train, X_test = X[:train_size], X[train_size:]: دنباله‌های ورودی (X) را به مجموعه‌های آموزشی و تستی تقسیم می‌کند.

y_train, y_test = y[:train_size], y[train_size:]: مقادیر هدف (y) را به مجموعه‌های آموزشی و تستی تقسیم می‌کند.

برای اینکه ورودی Transformer مناسب باشد یکبار دیگر داده ها را reshape کردیم تا مناسب ورودی شوند.

تعریف هایپر پارامتر های Transformer

```
d_model = 32  # Embedding dimension
num_heads = 2  # Number of attention heads
ff_dim = 32  # Hidden layer size in feedforward network
dropout_rate = 0.1
```

پیاده سازی positional encoding (رمز گذاری موقعیتی)

ادامه در صفحه بعد*

```
def get_angles(pos, i, d_model):
  angle_rates = 1 / np.power(10000, (2 * (i//2)) / np.float32(d_model))
  return pos * angle_rates

def positional_encoding(position, d_model):
  angle_rads = get_angles(np.arange(position)[:, np.newaxis],
                          np.arange(d_model)[np.newaxis, :],
                          d_model)


  angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])


  angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])

  pos_encoding = angle_rads[np.newaxis, ...]

  return tf.cast(pos_encoding, dtype=tf.float32)
```

get_angles(pos, i, d_model): زوایا را برای رمزگذاری موقعیتی بر اساس موقعیت**(pos)** ، شاخص بُعد **(i)** و **d_model** محاسبه می‌کند.

positional_encoding(position, d_model): ماتریس را تولید می‌کند.

این تابع یک ماتریس ایجاد می‌کند که در آن هر سطر نشان‌دهنده یک موقعیت در دنباله است. مقادیر موجود در ماتریس با استفاده از توابع سینوس و کسینوس با فرکانس‌های مختلف برای **positional encoding** محاسبه می‌شوند.

ساخت مدل Transformer

ادامه در صفحه بعد*

```
# Transformer Encoder
x = layers.MultiHeadAttention(num_heads=num_heads, key_dim=d_model)(x, x)
x = layers.Dropout(dropout_rate)(x)
x = layers.LayerNormalization(epsilon=1e-6)(x)

x = layers.Conv1D(filters=ff_dim, kernel_size=1, activation="relu")(x)
x = layers.Dropout(dropout_rate)(x)
x = layers.Conv1D(filters=inputs.shape[-1], kernel_size=1)(x)
x = layers.LayerNormalization(epsilon=1e-6)(x)

# Output layer
x = layers.GlobalAveragePooling1D()(x)
outputs = layers.Dense(1)(x)


model = keras.Model(inputs=inputs, outputs=outputs)
```

inputs = layers.Input(shape=(seq_length, 1)): لایه ورودی را با شکل **(seq_length, 1)** تعریف می‌کند.

pos_encoding = positional_encoding(): این خط یک رمزگذاری مکانی را برای یک دنباله بر اساس طول داده شده و بعد محاسبه می‌کند.

x = inputs + pos_encoding[]: این خط رمزگذاری مکانی را به دنباله ورودی **x** اضافه می‌کند. این اطمینان حاصل می‌کند که هر عنصر دنباله ورودی دارای اطلاعات مکانی تعبیه شده همراه با ویژگی‌های اصلی خود است.

لایه بعد **multi-head attention** را اضافه کرده ایم.

`x = layers.Dropout(dropout_rate)(x)`: یک لایه Dropout اضافه می‌کند.

`x = layers.LayerNormalization(epsilon=1e-6)(x)`: لایه نرمال‌سازی را اضافه می‌کند که به تثبیت آموزش کمک می‌کند و می‌تواند عملکرد را بهبود بخشد.

یک لایه کانولوشن ۱ بعدی اعمال کرده ایم در خط بعدی.

`x = layers.Dropout(dropout_rate)(x)`: یک لایه Dropout دیگر.

یک لایه کانولوشن یک بعدی دیگر اعمال کرده ایم در خط بعد.

`x = layers.LayerNormalization(epsilon=1e-6)(x)`: یک لایه نرمال‌سازی دیگر.

`x = layers.GlobalAveragePooling1D()(x)`: میانگین‌گیری را اعمال می‌کند تا دنباله را به یک بردار واحد کاهش دهد.

`outputs = layers.Dense(1)(x)`: یک لایه Dense (کاملاً متصل) با یک واحد خروجی برای انجام پیش‌بینی نهایی اضافه می‌کند.

مدل :model = keras.Model(inputs=inputs, outputs=outputs)

Kerasرا با مشخص کردن ورودی‌ها و خروجی‌ها ایجاد می‌کند.

کامپایل کردن مدل

```
model.compile(optimizer="adam", loss="mse")

model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 24, 1)	0	-
add (Add)	(None, 24, 32)	0	input_layer_2[0][0]
multi_head_attention (MultiHeadAttention)	(None, 24, 32)	8,416	add[0][0], add[0][0]
dropout_1 (Dropout)	(None, 24, 32)	0	multi_head_attention[…
layer_normalization (LayerNormalization)	(None, 24, 32)	64	dropout_1[0][0]
conv1d (Conv1D)	(None, 24, 32)	1,056	layer_normalization[0…
dropout_2 (Dropout)	(None, 24, 32)	0	conv1d[0][0]
conv1d_1 (Conv1D)	(None, 24, 1)	33	dropout_2[0][0]
layer_normalization_1 (LayerNormalization)	(None, 24, 1)	2	conv1d_1[0][0]
global_average_pooling1d (GlobalAveragePooling1D)	(None, 1)	0	layer_normalization_1…
dense (Dense)	(None, 1)	2	global_average_poolin…

Total params: 9,573 (37.39 KB)
Trainable params: 9,573 (37.39 KB)
Non-trainable params: 0 (0.00 B)

صرفا تابع بهینه سازی را به مدل داده و آن را کامل کرده ایم خلاصه مدل را میتوانید در

عکس بالا مشاهده کنید مطابق کد قسمت توضیح داده شده است.

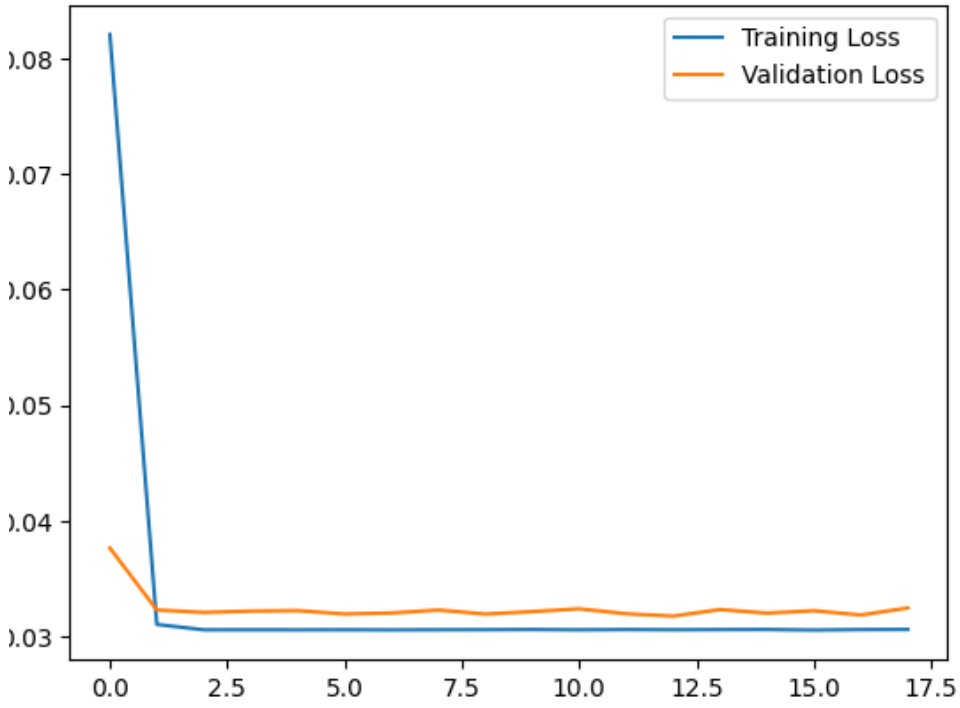
آموزش مدل

```
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=32,
    validation_split=0.1,
    callbacks=[
        keras.callbacks.EarlyStopping(monitor="val_loss", patience=5, mode="min")
    ]
)
```

```
Epoch 1/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 11s 16ms/step - loss: 0.1245 - val_loss: 0.0376
Epoch 2/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - loss: 0.0316 - val_loss: 0.0323
Epoch 3/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0308 - val_loss: 0.0321
Epoch 4/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - loss: 0.0314 - val_loss: 0.0322
Epoch 5/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - loss: 0.0305 - val_loss: 0.0322
Epoch 6/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0307 - val_loss: 0.0319
Epoch 7/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0308 - val_loss: 0.0320
Epoch 8/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 7s 20ms/step - loss: 0.0303 - val_loss: 0.0323
Epoch 9/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0301 - val_loss: 0.0319
Epoch 10/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0305 - val_loss: 0.0321
Epoch 11/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - loss: 0.0313 - val_loss: 0.0324
Epoch 12/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0304 - val_loss: 0.0320
Epoch 13/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0301 - val_loss: 0.0318
Epoch 14/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0307 - val_loss: 0.0323
Epoch 15/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 0.0312 - val_loss: 0.0320
Epoch 16/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0306 - val_loss: 0.0322
Epoch 17/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0301 - val_loss: 0.0318
Epoch 18/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0308 - val_loss: 0.0325
```

تعداد **epoch** و سایز **batch** ها و اندازه **validation** را مشخص کرده ایم و گفتیم بر اساس داده آموزشی، آموزش ببین.

```
t.plot(history.history['val_loss'], label='Validation Loss')
t.legend()
t.show()
```



نتیجه را در یک دیاگرام نشان داده ایم.

انجام پیشبینی

```
y_pred = model.predict(X_test)

y_pred_rescaled = scaler.inverse_transform(y_pred)
y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1, 1))
```

y_pred = model.predict(): با استفاده از مدل آموزش‌دیده، روی داده‌های تستی

پیش‌بینی انجام می‌دهد.

y_pred_rescaled = scaler.inverse_transform(): پیش‌بینی‌ها (که روی داده‌های مقیاس‌بندی‌شده انجام شده‌اند) را با استفاده از **object**، **scaler** به مقیاس اصلی تبدیل معکوس می‌کند.

y_test_rescaled=scaler.inverse_transform: مقادیر واقعی تست را برای مقایسه به مقیاس اصلی تبدیل معکوس می‌کند.

محاسبه MAE

```
mae = np.mean(np.abs(y_pred_rescaled - y_test_rescaled), axis=1)
```

میانگین خطای مطلق را بین مقدار پیشبینی شده و مقدار واقعی محاسبه میکند. به عنوان یک معیار برای خطا استفاده میشود و میگوید که چه قدر مدل میتواند به خوبی ورودی را تولید کند.

تنظیم anomaly threshold

```
threshold = np.max(mae) * 0.9
```

آستانه ناهنجاری را تنظیم می کند.

پیدا کردن ناهنجاری ها

```
anomalies = mae > threshold

anomaly_indices = np.where(anomalies)[0]
anomaly_timestamps = df.index[seq_length:][train_size:][anomaly_indices]
print("Anomalies detected at timestamps:")
print(anomaly_timestamps)
```

```
65/65 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
Anomalies detected at timestamps:
DatetimeIndex(['2014-12-26 05:00:00', '2015-01-01 00:30:00',
               '2015-01-01 01:00:00', '2015-01-20 03:30:00',
               '2015-01-26 22:30:00', '2015-01-26 23:00:00',
               '2015-01-26 23:30:00', '2015-01-27 00:00:00',
               '2015-01-27 00:30:00', '2015-01-27 01:00:00',
               '2015-01-27 01:30:00', '2015-01-27 02:00:00',
               '2015-01-27 02:30:00', '2015-01-27 03:00:00',
               '2015-01-27 03:30:00', '2015-01-27 04:00:00',
               '2015-01-27 04:30:00', '2015-01-27 05:00:00',
               '2015-01-27 05:30:00', '2015-01-27 06:00:00',
               '2015-01-27 06:30:00', '2015-01-27 07:00:00',
               '2015-01-27 07:30:00', '2015-01-27 08:00:00',
               '2015-01-27 08:30:00', '2015-01-28 03:00:00',
               '2015-01-28 03:30:00', '2015-01-28 04:00:00',
               '2015-01-28 04:30:00', '2015-01-31 19:00:00'],
              dtype='datetime64[ns]', name='timestamp', freq=None)
```

anomalies = mae > threshold: یک آرایه **boolean** ایجاد می‌کند که در آن **True** بیانگر یک ناهنجاری به این معنی که **MAE** بزرگتر از آستانه است و **False** بیان گر یک نقطه معمولی است.

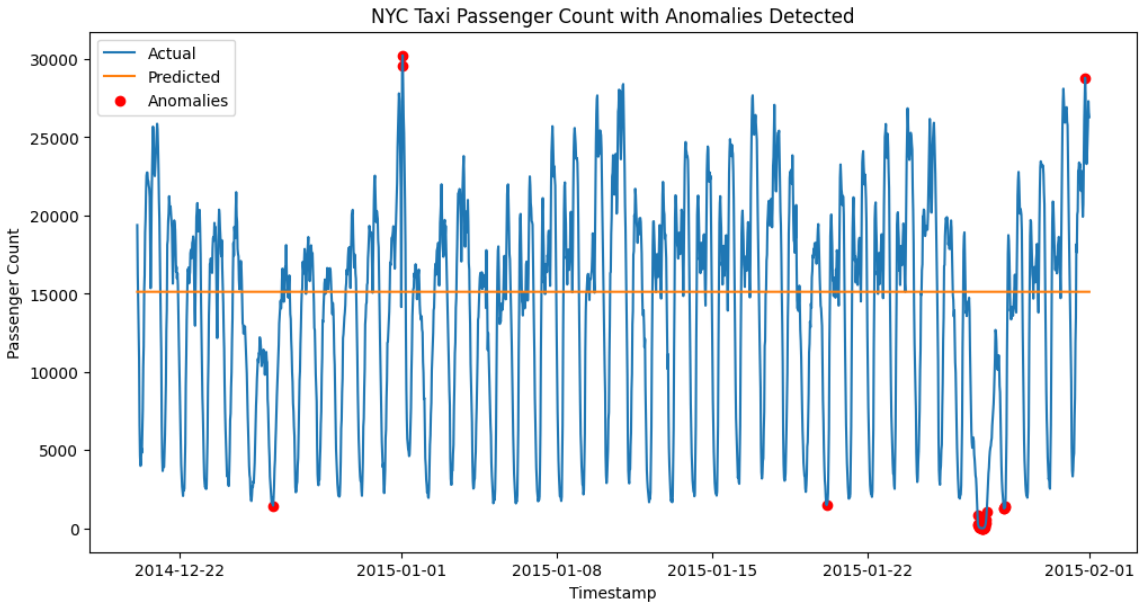
anomaly_indices = np.where(anomalies)[0]: اندیس‌های ناهنجاری‌ها را در آرایه **anomalies** قرار میدهد.

anomaly_timestamps = df.index[seq_length:][][]: بازه های زمانی

مربوط به اندیس های ناهنجاری را از DataFrame استخراج میکند.

نشان دادن ناهنجاری‌ها

```
plt.figure(figsize=(12, 6))
plt.plot(df.index[seq_length:][train_size:], y_test_rescaled, label='Actual')
plt.plot(df.index[seq_length:][train_size:], y_pred_rescaled, label='Predicted')
plt.scatter(anomaly_timestamps, y_test_rescaled[anomaly_indices], color='red', label='Anomalies')
plt.title('NYC Taxi Passenger Count with Anomalies Detected')
plt.xlabel('Timestamp')
plt.ylabel('Passenger Count')
plt.legend()
plt.show()
```



صرفا داده های واقعی و داده های پیشبینی شده توسط ما و ناهنجاری که با قرمز مشخص

شده اند در سری زمانی اصلی نشان داده ایم.

precision, recall and f-score گزارش

```python
from sklearn.metrics import precision_score, recall_score, f1_score


y_pred_binary = (mae > threshold).astype(int)

precision = precision_score(y_true, y_pred_binary)
recall = recall_score(y_true, y_pred_binary)
f1 = f1_score(y_true, y_pred_binary)

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```