```
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import IPython
from sklearn import metrics
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense



import pandas as pd

!pip install h3
import h3
import folium
import branca.colormap as cm

import torch.utils.data
from torch import optim, nn


import pytz

import scipy.optimize
# %matplotlib widget


# df = pd.read_csv('all_waybill_info_meituan_0322.csv')
df = pd.read_csv('/content/drive/MyDrive/Meituan/all_waybill_info_meituan_0322.csv')


df.dropna(inplace=True)
# df.drop(['Zodiac'], axis=1, inplace=True)

df.reset_index(drop=True, inplace=True)
# print(df)
```

```
Collecting h3
  Downloading h3-4.1.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Downloading h3-4.1.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (993 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 993.5/993.5 kB 11.7 MB/s eta 0:00:00
Installing collected packages: h3
Successfully installed h3-4.1.2
```

+ 代码 + 文本

```
type(df['platform_order_time'][0])
```

```
numpy.int64
```

```
#用来可视化的df2, 不是df
df2 = df[['is_courier_grabbed','is_prebook','platform_order_time','order_push_time','estimate_meal_prepare_time','recipient_lng','recipient_lat','sender_l

#去重,只看接受了的订单,只看非预约订单
df2 = df2[df2['is_prebook'] == 0]
df2 = df2[df2['is_courier_grabbed'] == 1]
df2= df2.sort_values(by='platform_order_time')
df2.reset_index(drop=True, inplace=True)
df2 = df2.drop(columns=['is_courier_grabbed'])

#转换time系列下单时间为date time
#转换完后数据格式是pandas._libs.tslibs.timestamps.Timestamp
df2['platform_order_time_date'] = pd.to_datetime(df2['platform_order_time'], unit='s')
df2['order_push_time_date'] = pd.to_datetime(df2['order_push_time'], unit='s')
df2['estimate_meal_prepare_time_date'] = pd.to_datetime(df2['estimate_meal_prepare_time'], unit='s')

#时区换成UTC+8hour,不要多次按！每次按都会在原基础上+8!
df2['platform_order_time_date'] = df2['platform_order_time_date'].dt.tz_localize('UTC').dt.tz_convert('Asia/Singapore')
df2['platform_order_time_date'] = df2['platform_order_time_date'].dt.tz_localize(None)
df2['order_push_time_date'] = df2['order_push_time_date'].dt.tz_localize('UTC').dt.tz_convert('Asia/Singapore')
df2['order_push_time_date'] = df2['order_push_time_date'].dt.tz_localize(None)
df2['estimate_meal_prepare_time_date'] = df2['estimate_meal_prepare_time_date'].dt.tz_localize('UTC').dt.tz_convert('Asia/Singapore')
df2['estimate_meal_prepare_time_date'] = df2['estimate_meal_prepare_time_date'].dt.tz_localize(None)

#帮助df2的时间戳数据添加一天之内的特征辅助
day = 24*60*60
```

```
27
28 df2['Day sin'] = np.sin(df2['platform_order_time'] * (2 * np.pi / day))
29 df2['Day cos'] = np.cos(df2['platform_order_time'] * (2 * np.pi / day))
30
31
```

```
1 def compute_h3_and_boundaries(row, resolution=9):
2         lng = row['recipient_lng']/1000000
3         lat = row['recipient_lat']/1000000
4         h3_index = h3.latlng_to_cell(lat, lng, resolution)
5         # print(h3_index)
6         # boundaries = h3.cell_to_boundary(h3_index)
7         # print(lng,lat,h3_index,boundaries)
8         return pd.Series([h3_index])
9
10 df2['sender_lng'] = pd.to_numeric(df2['sender_lng'], errors='coerce')
11 df2['sender_lat'] = pd.to_numeric(df2['sender_lat'], errors='coerce')
12 df2['recipient_lng'] = pd.to_numeric(df2['recipient_lng'], errors='coerce')
13 df2['recipient_lat'] = pd.to_numeric(df2['recipient_lat'], errors='coerce')
14
15 df2[['H3_Index']] = df2.apply(compute_h3_and_boundaries, axis=1)
```

```
1 df2['h3_long'] = df2['H3_Index'].apply(lambda x: int(x, 16) if isinstance(x, str) and all(c in '0123456789ABCDEFabcdef' for c in x) else
2 # resolution=9 共1711个cell
3
```

```
1 df2
```

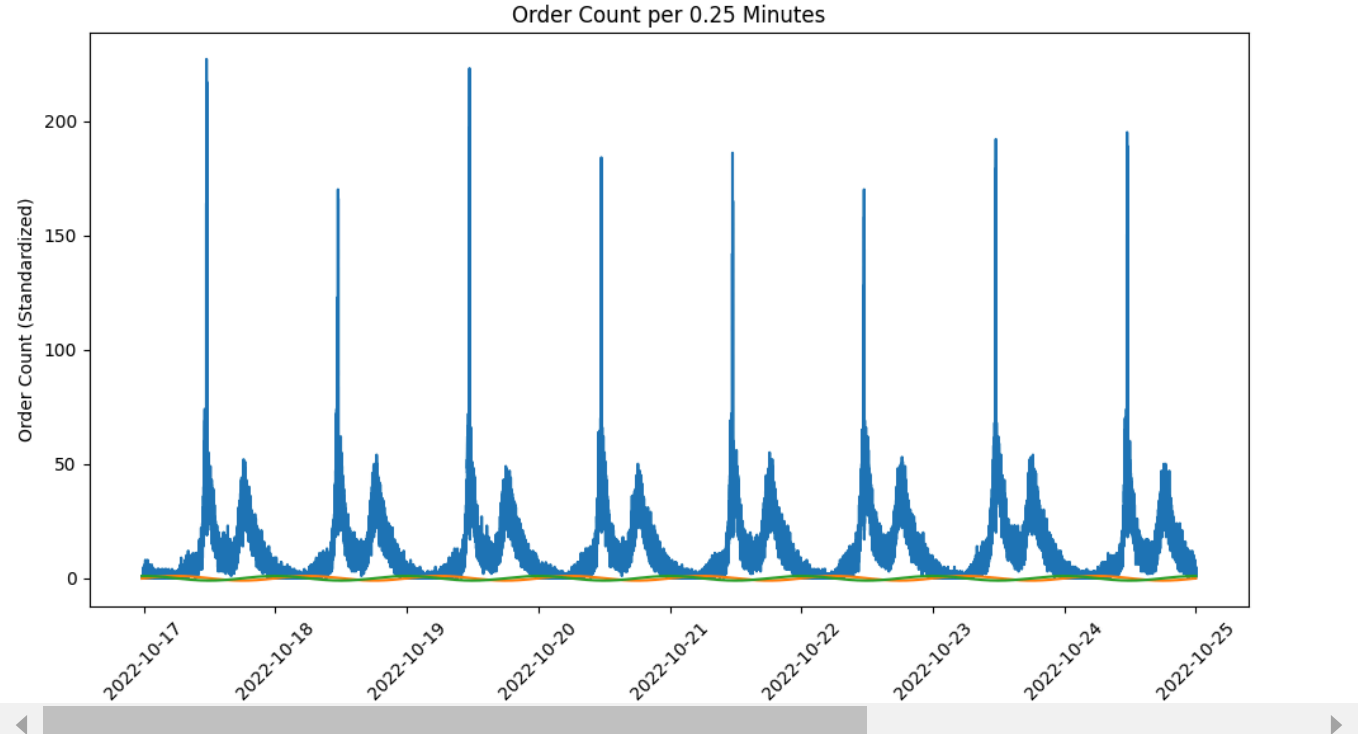
	is_prebook	platform_order_time	order_push_time	estimate_meal_prepare_time	recipient_lng	recipient_lat	sender_lng	sender
0	0	1665934777	1665936573	1665935391	174580449	45824914	174555530	4589
1	0	1665935139	1665937231	1665935753	174481572	45876511	174528619	4590
2	0	1665935365	1665936036	1665935979	174527950	45815592	174551066	4585
3	0	1665935379	1665935381	1665936161	174547139	45897170	174529930	4590
4	0	1665935707	1665935711	1665935711	174529156	45880736	174535543	4588
...	...	...	...	...	...	...	...	
546360	0	1666627171	1666627181	0	174545570	45868856	174573555	4586
546361	0	1666627173	1666627189	0	174569259	45879156	174595486	4587
546362	0	1666627181	1666627186	1666627845	174941500	46045530	174941512	4605
546363	0	1666627181	1666627194	0	174536704	45905336	174554670	4589
546364	0	1666627188	1666627199	0	174549457	45891735	174555813	4589

546365 rows × 15 columns

```
1 #切成0.25分钟，de_grouped_5min之后拿去当输入输出
2 df_grouped_025min = df2.resample('0.25min', on='platform_order_time_date').size()
3 df_grouped_025min = df_grouped_025min.to_frame(name='order_count')
4
5 df_grouped_025min = df_grouped_025min.reset_index()
6 df_grouped_025min.rename(columns={'index': 'platform_order_time_date'}, inplace=True)
7
8 df_grouped_025min['timestamp'] = df_grouped_025min['platform_order_time_date'].apply(lambda x: int(x.timestamp()))
9
10 day = 24*60*60
11
12 df_grouped_025min['Day sin'] = np.sin(df_grouped_025min['timestamp'] * (2 * np.pi / day))
13 df_grouped_025min['Day cos'] = np.cos(df_grouped_025min['timestamp'] * (2 * np.pi / day))
14
15 # scaler = MinMaxScaler()
16 # df_grouped_025min['order_count'] = scaler.fit_transform(df_grouped_025min[['order_count']])
17 # df_grouped_025min['Day sin'] = scaler.fit_transform(df_grouped_025min[['Day sin']])
18 # df_grouped_025min['Day cos'] = scaler.fit_transform(df_grouped_025min[['Day cos']])
19
20
21
22
23 print(df_grouped_025min)
24
25 #绘制图表
26 plt.figure(figsize=(10, 6))
27 plt.plot(df_grouped_025min['platform_order_time_date'].values, df_grouped_025min['order_count'].values)
28 plt.plot(df_grouped_025min['platform_order_time_date'].values, df_grouped_025min['Day sin'].values)
29 plt.plot(df_grouped_025min['platform_order_time_date'].values, df_grouped_025min['Day cos'].values)
30 plt.title('Order Count per 0.25 Minutes')
31 plt.xlabel('Time Slot')
32 plt.ylabel('Order Count (Standardized)')
33 plt.xticks(rotation=45)
```

```
plt.tight_layout()
plt.show()
```

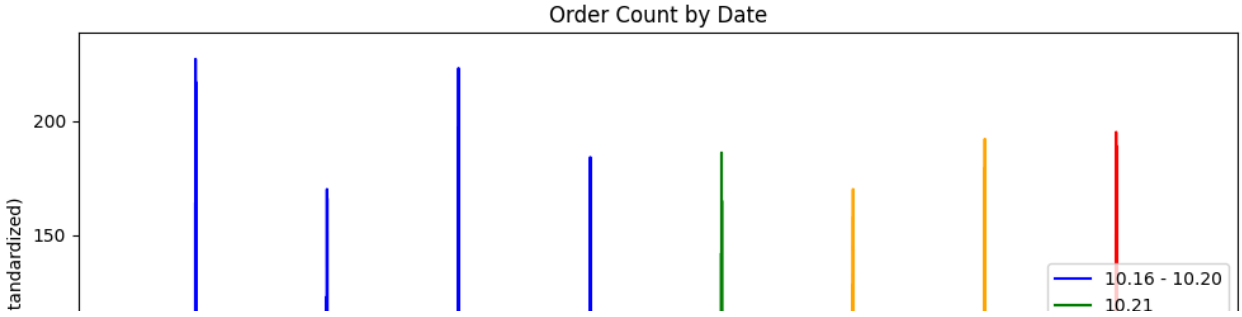
```
      platform_order_time_date  order_count   timestamp   Day sin   Day cos
0          2022-10-16 23:39:30            1  1665963570 -0.089329  0.996002
1          2022-10-16 23:39:45            0  1665963585 -0.088242  0.996099
2          2022-10-16 23:40:00            0  1665963600 -0.087156  0.996195
3          2022-10-16 23:40:15            0  1665963615 -0.086069  0.996289
4          2022-10-16 23:40:30            0  1665963630 -0.084982  0.996382
...                        ...          ...         ...       ...       ...
46157      2022-10-24 23:58:45            5  1666655925 -0.005454  0.999985
46158      2022-10-24 23:59:00            2  1666655940 -0.004363  0.999990
46159      2022-10-24 23:59:15            4  1666655955 -0.003272  0.999995
46160      2022-10-24 23:59:30            4  1666655970 -0.002182  0.999998
46161      2022-10-24 23:59:45            1  1666655985 -0.001091  0.999999

[46162 rows x 5 columns]
```




```
#10.17:  Monday  10.21:  Friday
#10.22-10.23:  Weekend
#10.24:  Monday
#区分成每天每天的

#  print(df_grouped_025min)
#df_grouped_025min已经标准化了

df_10_16_10_20  =  df_grouped_025min[
        (df_grouped_025min['platform_order_time_date']  >=  '2022-10-16')  &
        (df_grouped_025min['platform_order_time_date']  <  '2022-10-21')
]
print(df_10_16_10_20)

df_10_21  =  df_grouped_025min[
        (df_grouped_025min['platform_order_time_date']  >=  '2022-10-21')  &
        (df_grouped_025min['platform_order_time_date']  <  '2022-10-22')
]
#  print(df_10_21)

df_10_22_10_23  =  df_grouped_025min[
        (df_grouped_025min['platform_order_time_date']  >=  '2022-10-22')  &
        (df_grouped_025min['platform_order_time_date']  <  '2022-10-24')
]
#  print(df_10_22_10_23)

df_10_24  =  df_grouped_025min[
        (df_grouped_025min['platform_order_time_date']  >=  '2022-10-24')  &
        (df_grouped_025min['platform_order_time_date']  <  '2022-10-25')
]
print(df_10_24)


plt.figure(figsize=(10,  6))
plt.plot(df_10_16_10_20['platform_order_time_date'],  df_10_16_10_20['order_count'],
                  label='10.16  -  10.20',  color='blue')
plt.plot(df_10_16_10_20['platform_order_time_date'].values,  df_10_16_10_20['Day  sin'].values)
plt.plot(df_10_21['platform_order_time_date'],  df_10_21['order_count'],
                  label='10.21',  color='green')
```

```
plt.plot(df_10_22_10_23['platform_order_time_date'],  df_10_22_10_23['order_count'],
                  label='10.22  -  10.23',  color='orange')
plt.plot(df_10_24['platform_order_time_date'],  df_10_24['order_count'],
                  label='10.24',  color='red')


plt.title('Order  Count  by  Date')
plt.xlabel('Time')
plt.ylabel('Order  Count  (Standardized)')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```

```
      platform_order_time_date  order_count   timestamp   Day sin   Day cos
0          2022-10-16 23:39:30            1  1665963570 -0.089329  0.996002
1          2022-10-16 23:39:45            0  1665963585 -0.088242  0.996099
2          2022-10-16 23:40:00            0  1665963600 -0.087156  0.996195
3          2022-10-16 23:40:15            0  1665963615 -0.086069  0.996289
4          2022-10-16 23:40:30            0  1665963630 -0.084982  0.996382
...                        ...          ...         ...       ...       ...
23117      2022-10-20 23:58:45            4  1666310325 -0.005454  0.999985
23118      2022-10-20 23:59:00            3  1666310340 -0.004363  0.999990
23119      2022-10-20 23:59:15            5  1666310355 -0.003272  0.999995
23120      2022-10-20 23:59:30            4  1666310370 -0.002182  0.999998
23121      2022-10-20 23:59:45            1  1666310385 -0.001091  0.999999

[23122 rows x 5 columns]
      platform_order_time_date  order_count   timestamp       Day sin  \
40402      2022-10-24 00:00:00            5  1666569600 -2.301492e-12
40403      2022-10-24 00:00:15            2  1666569615  1.090831e-03
40404      2022-10-24 00:00:30            1  1666569630  2.181660e-03
40405      2022-10-24 00:00:45            3  1666569645  3.272486e-03
40406      2022-10-24 00:01:00            3  1666569660  4.363309e-03
...                        ...          ...         ...           ...
46157      2022-10-24 23:58:45            5  1666655925 -5.454127e-03
46158      2022-10-24 23:59:00            2  1666655940 -4.363309e-03
46159      2022-10-24 23:59:15            4  1666655955 -3.272487e-03
46160      2022-10-24 23:59:30            4  1666655970 -2.181660e-03
46161      2022-10-24 23:59:45            1  1666655985 -1.090831e-03

        Day cos
40402  1.000000
40403  0.999999
40404  0.999998
40405  0.999995
40406  0.999990
...         ...
46157  0.999985
46158  0.999990
46159  0.999995
46160  0.999998
46161  0.999999

[5760 rows x 5 columns]
```




```
df_10_16_10_20
df_10_24
```

	platform_order_time_date	order_count	timestamp	Day sin	Day cos
40402	2022-10-24 00:00:00	5	1666569600	-2.301492e-12	1.000000
40403	2022-10-24 00:00:15	2	1666569615	1.090831e-03	0.999999
40404	2022-10-24 00:00:30	1	1666569630	2.181660e-03	0.999998
40405	2022-10-24 00:00:45	3	1666569645	3.272486e-03	0.999995
40406	2022-10-24 00:01:00	3	1666569660	4.363309e-03	0.999990
...	...	...	...	...	...
46157	2022-10-24 23:58:45	5	1666655925	-5.454127e-03	0.999985
46158	2022-10-24 23:59:00	2	1666655940	-4.363309e-03	0.999990
46159	2022-10-24 23:59:15	4	1666655955	-3.272487e-03	0.999995
46160	2022-10-24 23:59:30	4	1666655970	-2.181660e-03	0.999998
46161	2022-10-24 23:59:45	1	1666655985	-1.090831e-03	0.999999

```
1
```

```
# train_data = df_10_16_10_20[['order_count']]


train_df = df_10_16_10_20[0:int(len(df_10_16_10_20)*0.9)]
val_df = df_10_16_10_20[int(len(df_10_16_10_20)*0.9):]
test_df = df_10_24


# print(train_df, val_df,test_df)

train_df = train_df.drop(columns=['platform_order_time_date'])
val_df = val_df.drop(columns=['platform_order_time_date'])
test_df = test_df.drop(columns=['platform_order_time_date'])

# print(train_df, val_df,test_df)

# scaler = MinMaxScaler()
scaler = StandardScaler()
train_df = pd.DataFrame(scaler.fit_transform(train_df), columns=train_df.columns, index=train_df.index)
val_df = pd.DataFrame(scaler.transform(val_df), columns=val_df.columns, index=val_df.index)
test_df = pd.DataFrame(scaler.transform(test_df), columns=test_df.columns, index=test_df.index)

print(train_df, val_df,test_df)


# X_train = df_10_16_10_20[['platform_order_time_date']].values    # 获取时间戳
# y_train = df_10_16_10_20['order_count'].values    # 获取订单计数

# # X_val. =

# X_test = df_10_24[['platform_order_time_date']].values
# y_test = df_10_24['order_count'].values

# print(type(X_train))

# scaler = StandardScaler()
# X_train = scaler.fit_transform(X_train)
# y_train = df_10_16_10_20['order_count'].values
# df_grouped_025min['order_count'] = scaler.fit_transform(df_grouped_025min[['order_count']])


# column_indices = {name: i for i, name in enumerate(df.columns)}

# n = len(df)
# train_df = df[0:int(n*0.7)]
# val_df = df[int(n*0.7):int(n*0.9)]
# test_df = df[int(n*0.9):]

# num_features = df.shape[1]



# 将时间分解为不同的特征
# df_10_16_10_20['month'] = df_10_16_10_20['platform_order_time_date'].dt.month
# df_10_16_10_20['day'] = df_10_16_10_20['platform_order_time_date'].dt.day
# df_10_16_10_20['hour'] = df_10_16_10_20['platform_order_time_date'].dt.hour
# df_10_16_10_20['minute'] = df_10_16_10_20['platform_order_time_date'].dt.minute

# print(df_10_16_10_20)


```

```
62
63
```

```
       order_count  timestamp   Day sin   Day cos
0        -0.741694  -1.731968 -0.243773  1.422220
1        -0.816142  -1.731801 -0.242207  1.422355
2        -0.816142  -1.731635 -0.240640  1.422489
3        -0.816142  -1.731468 -0.239074  1.422621
4        -0.816142  -1.731302 -0.237507  1.422752
...            ...        ...       ...       ...
20804    -0.146109   1.731302 -0.944382 -1.113010
20805    -0.220557   1.731468 -0.945667 -1.112132
20806     0.226131   1.731635 -0.946952 -1.111253
20807     0.002787   1.731801 -0.948236 -1.110373
20808    -0.146109   1.731968 -0.949518 -1.109491

[20809 rows x 4 columns]        order_count  timestamp   Day sin   Day cos
20809     0.077235   1.732134 -0.950800 -1.108607
20810     0.002787   1.732301 -0.952080 -1.107723
20811     0.077235   1.732467 -0.953360 -1.106837
20812    -0.071661   1.732633 -0.954639 -1.105949
20813     0.077235   1.732800 -0.955916 -1.105060
...            ...        ...       ...       ...
23117    -0.518350   2.116350 -0.122869  1.427787
23118    -0.592798   2.116516 -0.121297  1.427794
23119    -0.443902   2.116683 -0.119724  1.427800
23120    -0.518350   2.116849 -0.118152  1.427804
23121    -0.741694   2.117016 -0.116579  1.427807

[2313 rows x 4 columns]        order_count  timestamp   Day sin   Day cos
40402    -0.443902   4.993807 -0.115007  1.427808
40403    -0.667246   4.993973 -0.113434  1.427807
40404    -0.741694   4.994139 -0.111862  1.427804
40405    -0.592798   4.994306 -0.110290  1.427800
40406    -0.592798   4.994472 -0.108717  1.427794
...            ...        ...       ...       ...
46157    -0.443902   5.951849 -0.122869  1.427787
46158    -0.667246   5.952015 -0.121297  1.427794
46159    -0.518350   5.952182 -0.119724  1.427800
46160    -0.518350   5.952348 -0.118152  1.427804
46161    -0.741694   5.952515 -0.116579  1.427807

[5760 rows x 4 columns]
```

```
#generate window, 数据窗口化
#  windows  generator  可以处理如上图所示的索引和偏移量。
#  将特征窗口拆分为 (features, labels) 对。 feature: train label: 要预测的量
#  使用 tf.data.Dataset 从训练、评估和测试数据高效生成这些窗口的批次。


class WindowGenerator():
    def __init__(self, input_width, label_width, shift,
                             train_df=train_df, val_df=val_df, test_df=test_df,
                             label_columns=None):
        #  把傳進去的數據保存為self裡面的
        self.train_df = train_df
        self.val_df = val_df
        self.test_df = test_df

        #  Work out the label column indices.
        #  label_columns_indices：用于存储标签列名和它们的索引位置（例如，{'col_name': index}），也就是你想要预测的列
        #  column_indices：用于存储所有列的列名和它们的索引位置（方便在创建窗口时进行列选择） (index: 'col_name')
        self.label_columns = label_columns
        if label_columns is not None:
            self.label_columns_indices = {name: i for i, name in
                                                                enumerate(label_columns)}
        self.column_indices = {name: i for i, name in
                                                enumerate(train_df.columns)}

        #  Work out the window parameters 外界輸入
        self.input_width = input_width
        self.label_width = label_width
        self.shift = shift

        self.total_window_size = input_width + shift

        self.input_slice = slice(0, input_width) #输入train窗口的索引，np.arange(self.total_window_size) 会生成一个范围从 0 到 total window size
        self.input_indices = np.arange(self.total_window_size)[self.input_slice] #将 self.input_slice 应用得到输入窗口的索引 self.input_indices =

        self.label_start = self.total_window_size - self.label_width #output window的index
        self.labels_slice = slice(self.label_start, None)
        self.label_indices = np.arange(self.total_window_size)[self.labels_slice]

    def __repr__(self):
        return '\n'.join([
                f'Total window size: {self.total_window_size}',
                f'Input indices: {self.input_indices}',
                f'Label indices: {self.label_indices}',
                f'Label column name(s): {self.label_columns}'])
```

```
# 给定一个连续输入的列表，split_window 将它们转换为输入窗口和标签窗口。
# input: from input width, train feature in this window
# label: from label column, test, 希望预测的那个量


def split_window(self, features): #features:input df的所有col
    inputs = features[:, self.input_slice, :]
    labels = features[:, self.labels_slice, :]
    if self.label_columns is not None:
        labels = tf.stack(
                [labels[:, :, self.column_indices[name]] for name in self.label_columns],
                axis=-1)

    # Slicing doesn't preserve static shape information, so set the shapes
    # manually. This way the `tf.data.Datasets` are easier to inspect.
    inputs.set_shape([None, self.input_width, None])
    labels.set_shape([None, self.label_width, None])

    return inputs, labels

WindowGenerator.split_window = split_window
#windowgenerator：生成windowgenerator類
#split——window:輸入需要被split的feature input，label
```

```
# w2 = WindowGenerator(input_width=6, label_width=1, shift=1, label_columns=['order_count'])
# w2

# w1 = WindowGenerator(input_width=24, label_width=1, shift=24,
#                                       label_columns=['T (degC)'])


# 我创建了一个窗口
w = WindowGenerator(input_width=5760, label_width=240, shift=240, label_columns=['order_count'])
#这个是给定train - 预测最后一天test，offset = 5760,只想预测order_count
#这个只是单纯给定的window而已
```

```
print(w.split_window)
```

```
<bound method split_window of Total window size: 6000
Input indices: [   0    1    2 ... 5757 5758 5759]
Label indices: [5760 5761 5762 5763 5764 5765 5766 5767 5768 5769 5770 5771 5772 5773
 5774 5775 5776 5777 5778 5779 5780 5781 5782 5783 5784 5785 5786 5787
 5788 5789 5790 5791 5792 5793 5794 5795 5796 5797 5798 5799 5800 5801
 5802 5803 5804 5805 5806 5807 5808 5809 5810 5811 5812 5813 5814 5815
 5816 5817 5818 5819 5820 5821 5822 5823 5824 5825 5826 5827 5828 5829
 5830 5831 5832 5833 5834 5835 5836 5837 5838 5839 5840 5841 5842 5843
 5844 5845 5846 5847 5848 5849 5850 5851 5852 5853 5854 5855 5856 5857
 5858 5859 5860 5861 5862 5863 5864 5865 5866 5867 5868 5869 5870 5871
 5872 5873 5874 5875 5876 5877 5878 5879 5880 5881 5882 5883 5884 5885
 5886 5887 5888 5889 5890 5891 5892 5893 5894 5895 5896 5897 5898 5899
 5900 5901 5902 5903 5904 5905 5906 5907 5908 5909 5910 5911 5912 5913
 5914 5915 5916 5917 5918 5919 5920 5921 5922 5923 5924 5925 5926 5927
 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5938 5939 5940 5941
 5942 5943 5944 5945 5946 5947 5948 5949 5950 5951 5952 5953 5954 5955
 5956 5957 5958 5959 5960 5961 5962 5963 5964 5965 5966 5967 5968 5969
 5970 5971 5972 5973 5974 5975 5976 5977 5978 5979 5980 5981 5982 5983
 5984 5985 5986 5987 5988 5989 5990 5991 5992 5993 5994 5995 5996 5997
 5998 5999]
Label column name(s): ['order_count']>
```

```
# Stack three slices, the length of the total window.
example_window = tf.stack([np.array(train_df[:w.total_window_size]),
                                                np.array(train_df[100:100+w.total_window_size]),
                                                np.array(train_df[10000:10000+w.total_window_size])])
#从train df里面随便抓三个窗口出来

example_inputs, example_labels = w.split_window(example_window)

print('All shapes are: (batch, time, features)')
print(f'Window shape: {example_window.shape}')
print(f'Inputs shape: {example_inputs.shape}')
print(f'Labels shape: {example_labels.shape}')

# w.example = example_inputs, example_labels
# 把这个example set成为w这个window的特征
```

```
All shapes are: (batch, time, features)
Window shape: (3, 6000, 4)
Inputs shape: (3, 5760, 4)
Labels shape: (3, 240, 1)
```

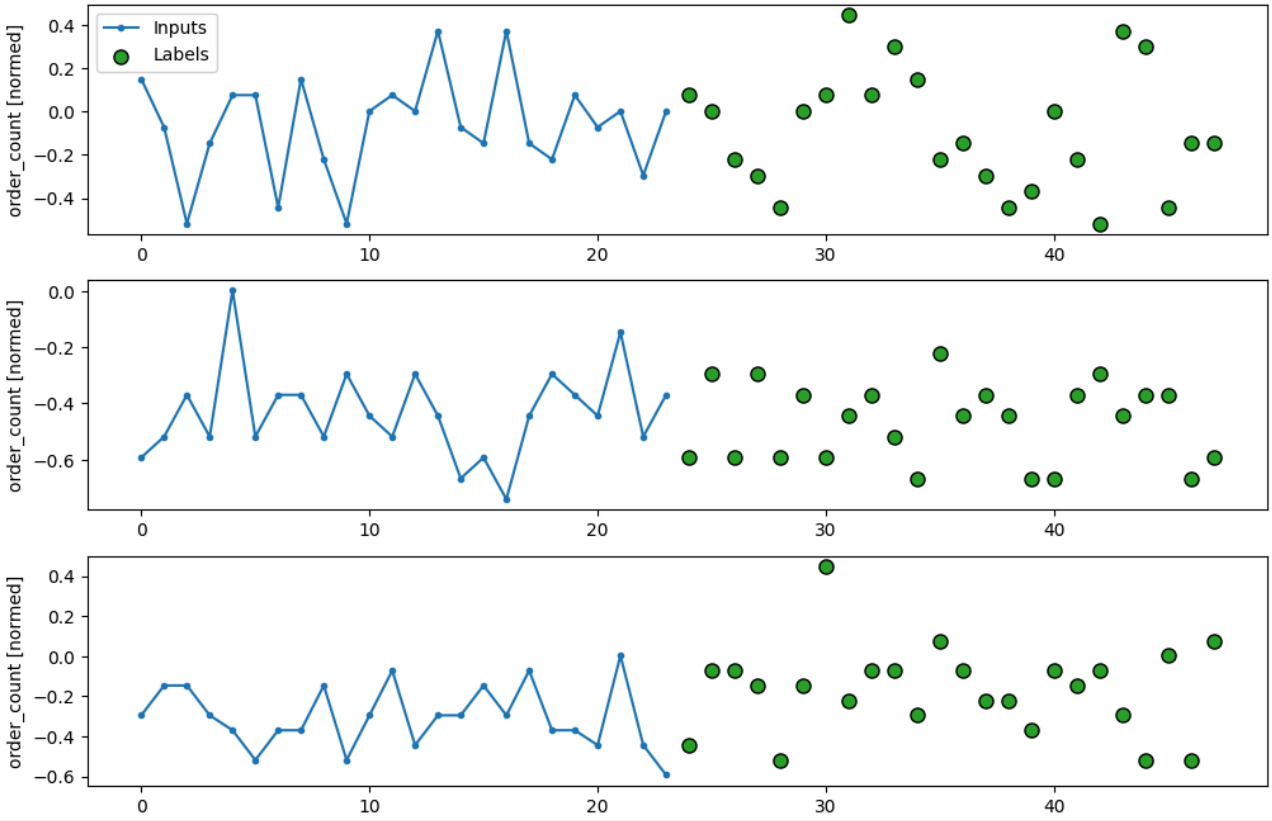
```
1
```

```
def plot(self, model=None, plot_col='order_count', max_subplots=3):
    inputs, labels = self.example
```

```
    plt.figure(figsize=(12, 8))
    plot_col_index = self.column_indices[plot_col]
    max_n = min(max_subplots, len(inputs))

    print("Inputs shape from plot func:", inputs.shape)
    print("Labels shape from plot func:", labels.shape)


    for n in range(max_n):
        plt.subplot(max_n, 1, n+1)
        plt.ylabel(f'{plot_col} [normed]')
        plt.plot(self.input_indices, inputs[n, :, plot_col_index],
                         label='Inputs', marker='.', zorder=-10)

        if self.label_columns:
            label_col_index = self.label_columns_indices.get(plot_col, None)
        else:
            label_col_index = plot_col_index

        if label_col_index is None:
            continue

        plt.scatter(self.label_indices, labels[n, :, label_col_index],
                                edgecolors='k', label='Labels', c='#2ca02c', s=64)
        if model is not None:
            predictions = model(inputs)
            plt.scatter(self.label_indices, predictions[n, :, label_col_index],
                                    marker='X', edgecolors='k', label='Predictions',
                                    c='#ff7f0e', s=64)

        if n == 0:
            plt.legend()

    plt.xlabel('Time [h]')

WindowGenerator.plot = plot
```

```
OUT_STEPS = 24
multi_window = WindowGenerator(input_width=24,
                                                       label_width=OUT_STEPS,
                                                       shift=OUT_STEPS)

multi_window.plot()
multi_window
```

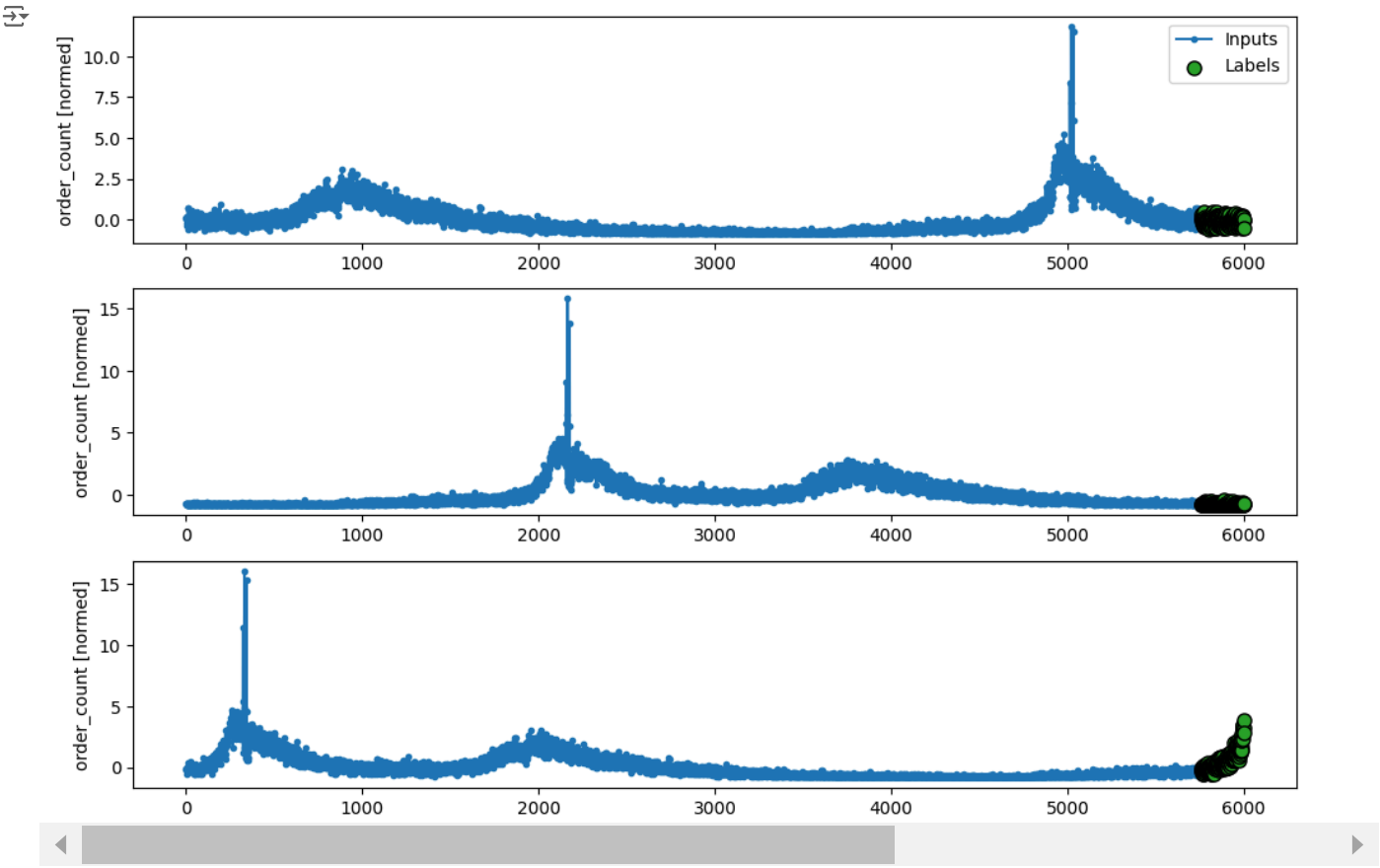
```
Total window size: 48
Input indices: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
Label indices: [24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47]
Label column name(s): None
```



```
#window generator：只定义了窗户
#make dataset：把train set切成很多很多个窗户

#  timeseries_dataset_from_array 根据 total_window_size = input_width + label_width + shift，在 train_df 上生成重叠的窗口对。
#  然后batch成32的

def make_dataset(self, data):
    data = np.array(data, dtype=np.float64)
    ds = tf.keras.utils.timeseries_dataset_from_array(
          data=data,
          targets=None,
          sequence_length=self.total_window_size,
          sequence_stride=1,
          shuffle=True,
          batch_size=32,)

    ds = ds.map(self.split_window)

    return ds

#  从训练数据集中获取并缓存一个样本批次 (inputs, labels)，以便可以轻松访问示例数据（通常用于调试或绘图）
#  把这个函数定义到window generator里面
WindowGenerator.make_dataset = make_dataset
```

```
#  内部调用:  train：用train_df扔进去make_dataset(把traindf切成窗口们)

@property
def train(self):
    return self.make_dataset(self.train_df)

@property
def val(self):
    return self.make_dataset(self.val_df)

@property
def test(self):
    return self.make_dataset(self.test_df)

@property
def example(self):
    """Get and cache an example batch of `inputs, labels` for plotting."""
    result = getattr(self, '_example', None)
    if result is None:
        # No example batch was found, so get one from the `.train` dataset
        result = next(iter(self.train))
        # And cache it for next time
        self._example = result
    return result

WindowGenerator.train = train
WindowGenerator.val = val
WindowGenerator.test = test
WindowGenerator.example = example
```

```
w.plot()  #就
```




```
# Each element is an (inputs, label) pair.
#w.train 是一个由 WindowGenerator 创建的数据集，element_spec 则展示了每个元素的输入和标签的结构，包括张量的形状（shape）和数据类型（dtype）
w.train.element_spec
```

```
(TensorSpec(shape=(None, 5760, 4), dtype=tf.float64, name=None),
 TensorSpec(shape=(None, 240, 1), dtype=tf.float64, name=None))
```

```
for example_inputs, example_labels in w.train.take(1):
    print(f'Inputs shape (batch, time, features): {example_inputs.shape}')
    print(f'Labels shape (batch, time, features): {example_labels.shape}')
```

```
Inputs shape (batch, time, features): (32, 5760, 4)
Labels shape (batch, time, features): (32, 240, 1)
```

```
# 单步window

single_step_window = WindowGenerator(input_width=1, label_width=1, shift=1, label_columns=['order_count'])
single_step_window
```

```
Total window size: 2
Input indices: [0]
Label indices: [1]
Label column name(s): ['order_count']
```

```
for example_inputs, example_labels in single_step_window.train.take(1):
    print(f'Inputs shape (batch, time, features): {example_inputs.shape}')
    print(f'Labels shape (batch, time, features): {example_labels.shape}')

```

```
Inputs shape (batch, time, features): (32, 1, 4)
Labels shape (batch, time, features): (32, 1, 1)
```

```
# class Baseline(tf.keras.Model):
#     def __init__(self, label_index=None):
#         super().__init__()
#         self.label_index = label_index
#
#     def call(self, inputs):
#         if self.label_index is None:
#             return inputs
#         result = inputs[:, :, self.label_index]
#         return result[:, :, tf.newaxis]
```

```
linear = tf.keras.Sequential([tf.keras.layers.Dense(units=1)])
print('Input shape:', single_step_window.example[0].shape)
print('Output shape:', linear(single_step_window.example[0]).shape)
```

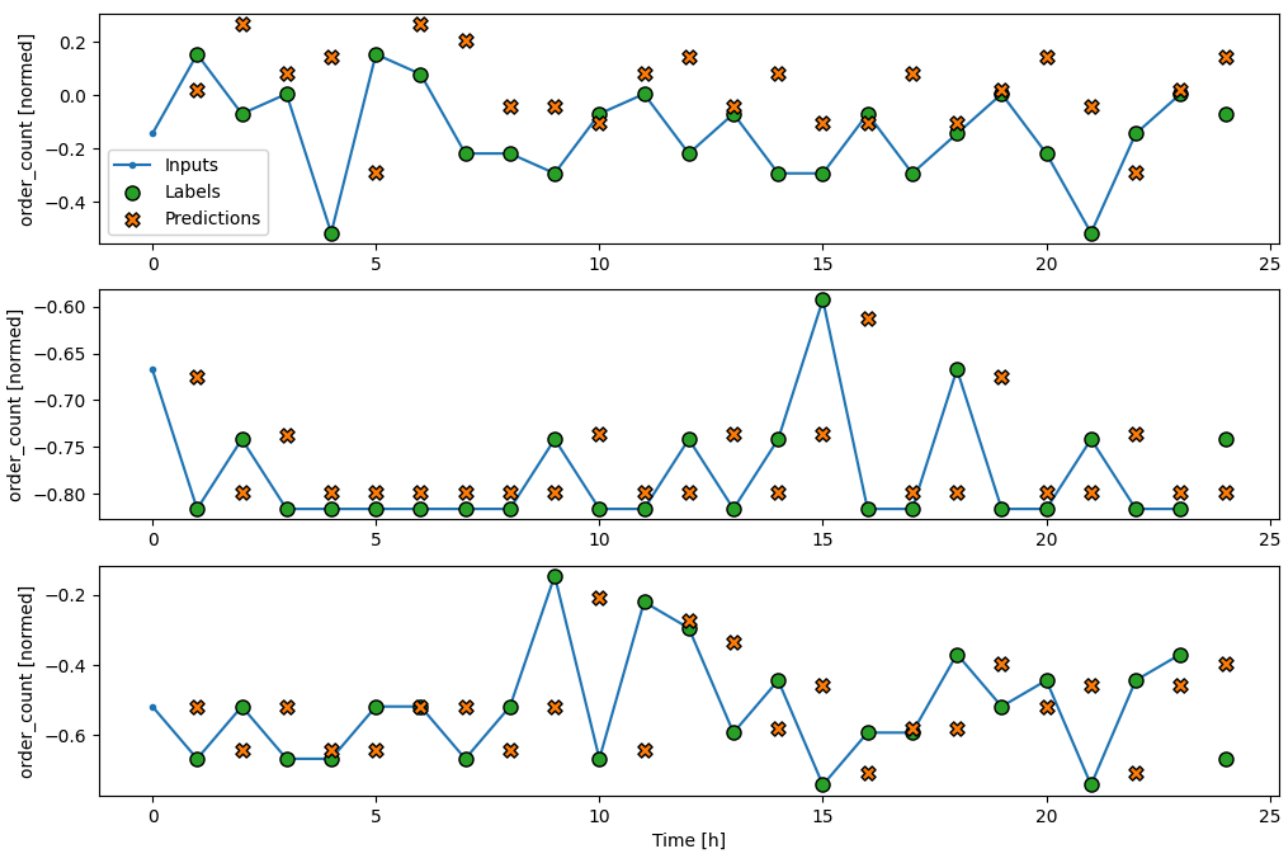
```
Input shape: (32, 1, 4)
Output shape: (32, 1, 1)
```

```
MAX_EPOCHS = 1

def compile_and_fit(model, window, patience=2):
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                      patience=patience,
                                                      mode='min')

    model.compile(loss=tf.keras.losses.MeanSquaredError(),
                  optimizer=tf.keras.optimizers.Adam(),
                  metrics=[tf.keras.metrics.MeanAbsoluteError()])

    history = model.fit(window.train, epochs=MAX_EPOCHS,
                        validation_data=window.val,
                        callbacks=[early_stopping])
    return history
```

```
val_performance = {}
performance = {}

history = compile_and_fit(linear, single_step_window)

val_performance['Linear'] = linear.evaluate(single_step_window.val)
performance['Linear'] = linear.evaluate(single_step_window.test, verbose=0)
```

```
Epoch 1/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.3932 - mean_absolute_error: 0.3785 - val_loss: 0.2036 - val_mean_absolute_error:
Epoch 2/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2221 - mean_absolute_error: 0.2723 - val_loss: 0.1528 - val_mean_absolute_error:
Epoch 3/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.1898 - mean_absolute_error: 0.2426 - val_loss: 0.1431 - val_mean_absolute_error:
Epoch 4/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.1835 - mean_absolute_error: 0.2365 - val_loss: 0.1407 - val_mean_absolute_error:
Epoch 5/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - loss: 0.1827 - mean_absolute_error: 0.2355 - val_loss: 0.1399 - val_mean_absolute_error
Epoch 6/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - loss: 0.1823 - mean_absolute_error: 0.2351 - val_loss: 0.1398 - val_mean_absolute_error
Epoch 7/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9s 14ms/step - loss: 0.1821 - mean_absolute_error: 0.2345 - val_loss: 0.1399 - val_mean_absolute_error
Epoch 8/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 0.1828 - mean_absolute_error: 0.2345 - val_loss: 0.1397 - val_mean_absolute_error:
Epoch 9/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11s 10ms/step - loss: 0.1845 - mean_absolute_error: 0.2348 - val_loss: 0.1399 - val_mean_absolute_erro
Epoch 10/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.1844 - mean_absolute_error: 0.2351 - val_loss: 0.1394 - val_mean_absolute_error:
Epoch 11/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.1837 - mean_absolute_error: 0.2343 - val_loss: 0.1397 - val_mean_absolute_error:
Epoch 12/20
651/651 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.1839 - mean_absolute_error: 0.2347 - val_loss: 0.1399 - val_mean_absolute_error:
73/73 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1337 - mean_absolute_error: 0.2817
```

```
wide_window = WindowGenerator(input_width=24, label_width=24, shift=1,label_columns=['order_count'])

# print('Input shape:', wide_window.example[0].shape)
# print('Output shape:', baseline(wide_window.example[0]).shape)


wide_window.plot(linear)
```




```
# plt.bar(x = range(len(train_df.columns)),
#                 height=linear.layers[0].kernel[:,0].numpy())
# axis = plt.gca()
# axis.set_xticks(range(len(train_df.columns)))
# _ = axis.set_xticklabels(train_df.columns, rotation=90)
```

```
dense = tf.keras.Sequential([
        tf.keras.layers.Dense(units=64, activation='tanh'),
        tf.keras.layers.Dense(units=64, activation='tanh'),
        tf.keras.layers.Dense(units=1)
])

history = compile_and_fit(dense, single_step_window)

val_performance['Dense'] = dense.evaluate(single_step_window.val)
performance['Dense'] = dense.evaluate(single_step_window.test, verbose=0)
```

```
Epoch 1/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - loss: 0.2057 - mean_absolute_error: 0.2743 - val_loss: 0.1346 - val_mean_absolute_error:
Epoch 2/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.1656 - mean_absolute_error: 0.2439 - val_loss: 0.1343 - val_mean_absolute_error:
Epoch 3/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 0.1596 - mean_absolute_error: 0.2363 - val_loss: 0.1357 - val_mean_absolute_error:
Epoch 4/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.1558 - mean_absolute_error: 0.2328 - val_loss: 0.1117 - val_mean_absolute_error:
Epoch 5/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - loss: 0.1511 - mean_absolute_error: 0.2257 - val_loss: 0.1080 - val_mean_absolute_error:
Epoch 6/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 0.1418 - mean_absolute_error: 0.2156 - val_loss: 0.0954 - val_mean_absolute_error:
Epoch 7/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.1341 - mean_absolute_error: 0.2073 - val_loss: 0.1020 - val_mean_absolute_error:
Epoch 8/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - loss: 0.1291 - mean_absolute_error: 0.2028 - val_loss: 0.0946 - val_mean_absolute_error:
Epoch 9/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.1241 - mean_absolute_error: 0.1964 - val_loss: 0.0919 - val_mean_absolute_error:
Epoch 10/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.1229 - mean_absolute_error: 0.1952 - val_loss: 0.0903 - val_mean_absolute_error:
Epoch 11/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.1195 - mean_absolute_error: 0.1916 - val_loss: 0.0904 - val_mean_absolute_error:
Epoch 12/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 8s 8ms/step - loss: 0.1193 - mean_absolute_error: 0.1924 - val_loss: 0.0834 - val_mean_absolute_error:
Epoch 13/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - loss: 0.1184 - mean_absolute_error: 0.1915 - val_loss: 0.0961 - val_mean_absolute_error:
Epoch 14/20
651/651 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - loss: 0.1173 - mean_absolute_error: 0.1906 - val_loss: 0.0855 - val_mean_absolute_error:
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0837 - mean_absolute_error: 0.2234
```

```
#上面的是单步预测
#下面用多步预测
```

```
3*60*4
```

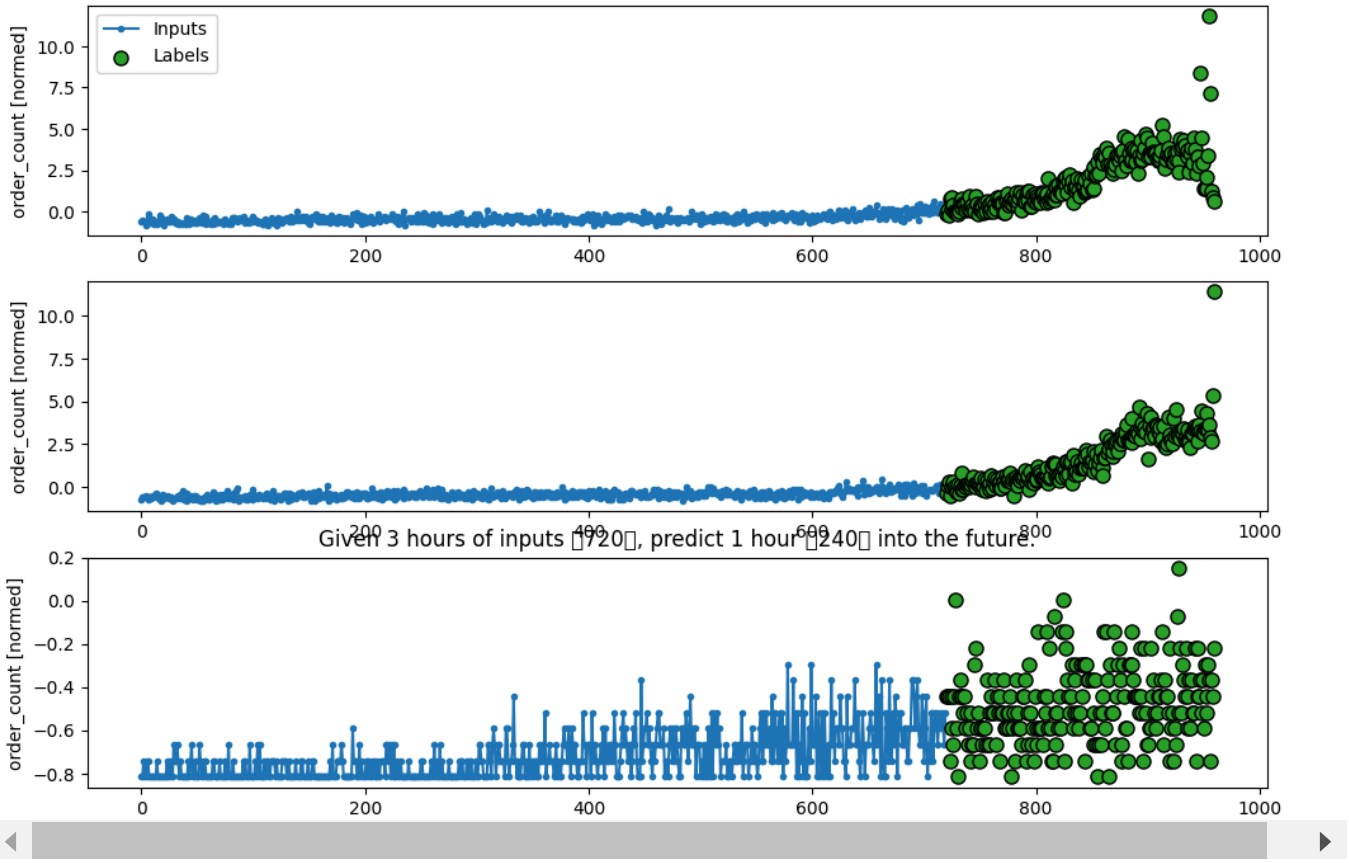
```
720
```

```
CONV_WIDTH = 720
conv_window = WindowGenerator(
        input_width=CONV_WIDTH,
        label_width=240,
        shift=240,
        label_columns=['order_count'])

conv_window
#每次输入三步input indices - 我们选3个小时 - 3小时 = 720
```

```
Total window size: 960
Input indices: [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395
 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413
 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431
 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485
 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503
 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521
 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539
 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557
 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575
 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593
 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611
 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629
 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647
 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665
 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683
 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701
 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719]
Label indices: [720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737
 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755
 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773
 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791
 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809
 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827
 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845
 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863
 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881
 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899
 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917
 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935
 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953
 954 955 956 957 958 959]
Label column name(s): ['order_count']
```

```
conv_window.plot()
plt.title("Given 3 hours of inputs (720), predict 1 hour (240) into the future.")
```

```
Text(0.5, 1.0, 'Given 3 hours of inputs （720）, predict 1 hour （240） into the future.')
/usr/local/lib/python3.10/dist-packages/IPython/core/events.py:89: UserWarning: Glyph 65288 (\N{FULLWIDTH LEFT PARENTHESIS}) missing from current font.
  func(*args, **kwargs)
/usr/local/lib/python3.10/dist-packages/IPython/core/events.py:89: UserWarning: Glyph 65289 (\N{FULLWIDTH RIGHT PARENTHESIS}) missing from current font.
  func(*args, **kwargs)
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 65288 (\N{FULLWIDTH LEFT PARENTHESIS}) missing from current fo
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 65289 (\N{FULLWIDTH RIGHT PARENTHESIS}) missing from current f
  fig.canvas.print_figure(bytes_io, **kw)
```




```
multi_step_dense = tf.keras.Sequential([
        # Shape: (time, features) => (time*features)
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(units=32, activation='tanh'),
        tf.keras.layers.Dense(units=32, activation='tanh'),
        tf.keras.layers.Dense(units=240), #输出多少个值
        # Add back the time dimension.
        # Shape: (outputs) => (1, outputs)
        tf.keras.layers.Reshape([240, 1]), #reshape输出
])


# multi_step_dense = tf.keras.Sequential([
#         tf.keras.layers.Flatten(),
#         tf.keras.layers.Dense(units=32),
#         tf.keras.layers.LeakyReLU(alpha=0.01),   # 设置 LeakyReLU 激活，alpha 表示负值的斜率
#         tf.keras.layers.Dense(units=32),
#         tf.keras.layers.LeakyReLU(alpha=0.01),
#         tf.keras.layers.Dense(units=240),
#         tf.keras.layers.Reshape([240, 1]),
# ])
```

```
print('Input shape:', conv_window.example[0].shape)
print('Output shape:', multi_step_dense(conv_window.example[0]).shape)
```

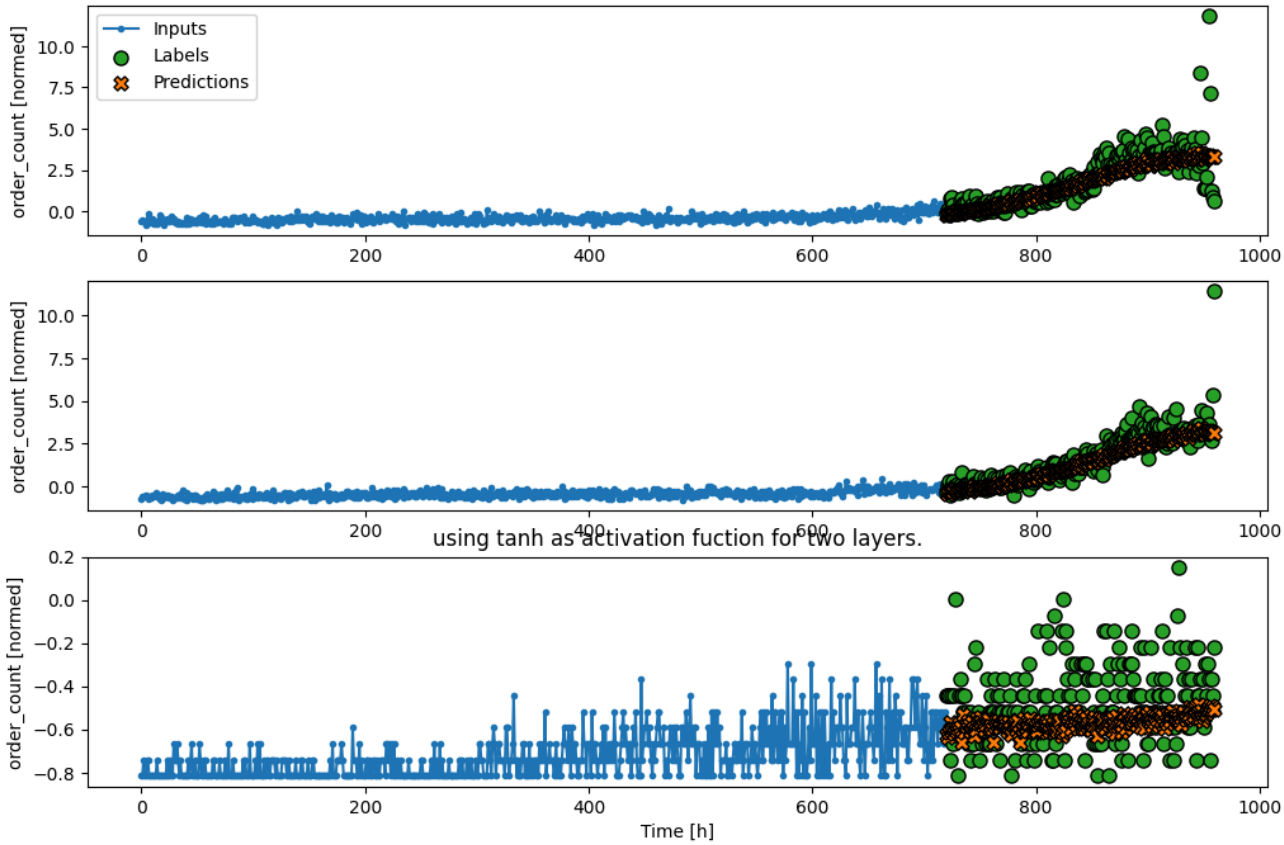
```
Input shape: (32, 720, 4)
Output shape: (32, 240, 1)
```

```
history = compile_and_fit(multi_step_dense, conv_window)

IPython.display.clear_output()
val_performance['Multi step dense'] = multi_step_dense.evaluate(conv_window.val)
performance['Multi step dense'] = multi_step_dense.evaluate(conv_window.test, verbose=0)

#using tanh as activation function: 43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0990 - mean_absolute_error: 0.2492
#using leaky relu:43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0999 - mean_absolute_error: 0.2432

```

```
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1274 - mean_absolute_error: 0.2810
```

```
conv_window.plot(multi_step_dense)
plt.title("using tanh as activation fuction for two layers.")
```

```
Text(0.5, 1.0, 'using tanh as activation fuction for two layers.')
```




```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
# 打印每个窗口的数量
print(f"Training set windows count: {len(lstm_w.train)}")
print(f"Validation set windows count: {len(lstm_w.val)}")
print(f"Test set windows count: {len(lstm_w.test)}")

```

```
Training set windows count: 471
Validation set windows count: 0
Test set windows count: 0
```

```
1
```

```
lstm_w = WindowGenerator(input_width=720, label_width=720, shift=1,label_columns=['order_count'])
```

```
wide_window = WindowGenerator(input_width=24, label_width=24, shift=1,label_columns=['order_count'])
```

```
lstm_model = tf.keras.models.Sequential([
        # Shape [batch, time, features] => [batch, time, lstm_units]
        tf.keras.layers.LSTM(32, return_sequences=True, input_shape=(720, 4)),
        # Shape => [batch, time, features]
        tf.keras.layers.Dense(units=1)
        # output = lstm_model(example_input)[:, -240:, :]
        # tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(1))

])

print('Input shape:', lstm_w.example[0].shape)
print('Output shape:', lstm_model(lstm_w.example[0]).shape)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-00000af2d6ad> in <cell line: 1>()
----> 1 lstm_model = tf.keras.models.Sequential([
      2     # Shape [batch, time, features] => [batch, time, lstm_units]
      3     tf.keras.layers.LSTM(32, return_sequences=True, input_shape=(720, 4)),
      4     # Shape => [batch, time, features]
      5     tf.keras.layers.Dense(units=1)

NameError: name 'tf' is not defined
```

```
1
```

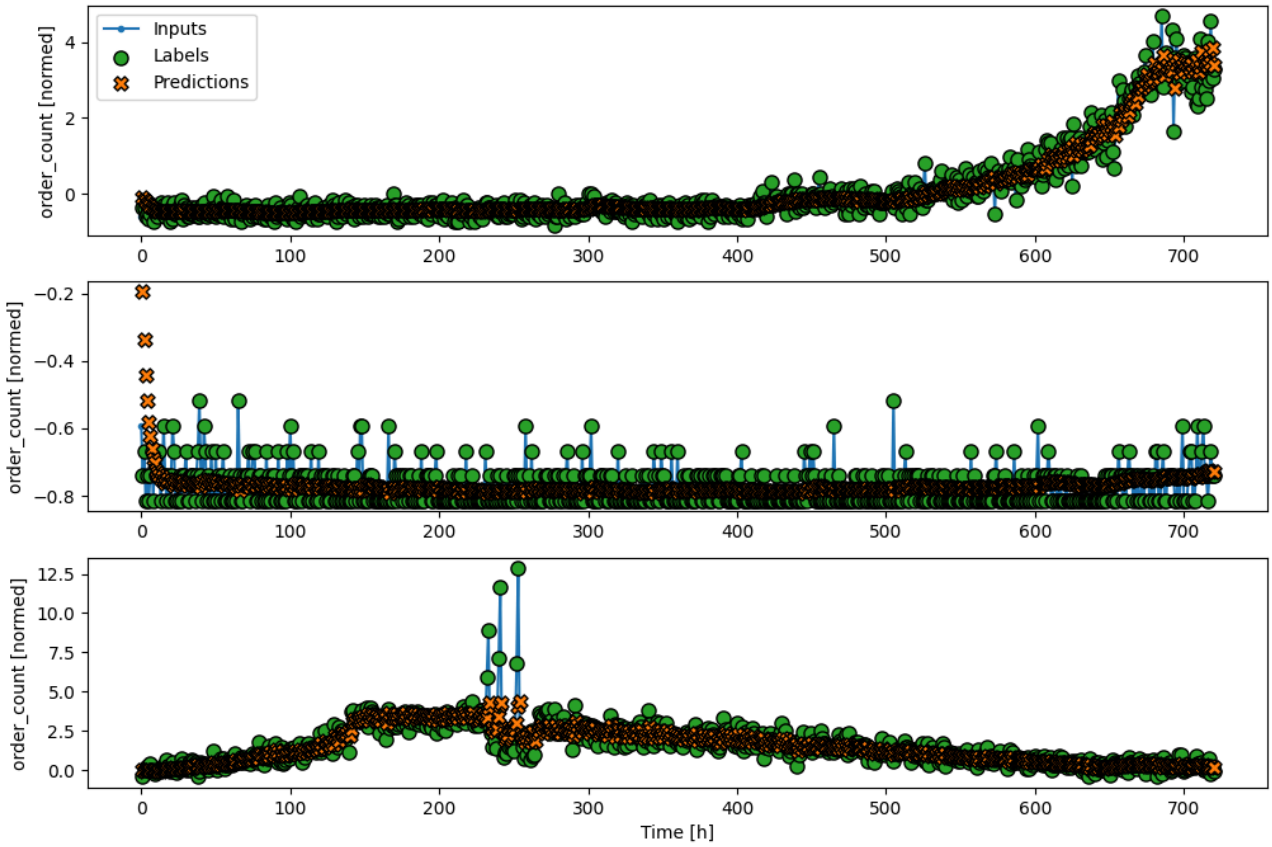
```
history = compile_and_fit(lstm_model, lstm_w)

IPython.display.clear_output()
val_performance['LSTM'] = lstm_model.evaluate(lstm_w.val)
performance['LSTM'] = lstm_model.evaluate(lstm_w.test, verbose=0)
```

```
50/50 ━━━━━━━━━━━━━━━━━━━━ 5s 101ms/step - loss: 0.1033 - mean_absolute_error: 0.2504
```

```
1
```

```
lstm_w.plot(lstm_model)
```

```
Inputs shape from plot func: (32, 720, 4)
Labels shape from plot func: (32, 720, 1)
```




```
MAX_EPOCHS = 20

def compile_and_fit(model, window, patience=2):
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                      patience=patience,
                                                      mode='min')

```

```
    model.compile(loss=tf.keras.losses.MeanSquaredError(),
                              optimizer=tf.keras.optimizers.Adam(),
                              metrics=[tf.keras.metrics.MeanAbsoluteError()])

    history = model.fit(window.train, epochs=MAX_EPOCHS,
                                      validation_data=window.val,
                                      callbacks=[early_stopping])
    return history
```

```
history = compile_and_fit(lstm_model, w)

IPython.display.clear_output()
val_performance['LSTM'] = lstm_model.evaluate(w.val)
performance['LSTM'] = lstm_model.evaluate(w.test, verbose=0)
```

```
Epoch 1/20
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-102-7be2087692b9> in <cell line: 1>()
----> 1 history = compile_and_fit(lstm_model, w)
      2
      3 IPython.display.clear_output()
      4 val_performance['LSTM'] = lstm_model.evaluate(w.val)
      5 performance['LSTM'] = lstm_model.evaluate(w.test, verbose=0)

                                  2 frames
/usr/local/lib/python3.10/dist-packages/keras/src/losses/losses.py in mean_squared_error(y_true, y_pred)
   1284     y_true = ops.convert_to_tensor(y_true, dtype=y_pred.dtype)
   1285     y_true, y_pred = squeeze_or_expand_to_same_rank(y_true, y_pred)
-> 1286     return ops.mean(ops.square(y_true - y_pred), axis=-1)
   1287
   1288

ValueError: Dimensions must be equal, but are 240 and 5760 for '{{node compile_loss/mean_squared_error/sub}} = Sub[T=DT_FLOAT]
(compile_loss/mean_squared_error/Cast, sequential_3_1/dense_3_1/Add)' with input shapes: [?,240,1], [?,5760,1].
```

```
1
```

```
1
```

```
1
```

```
def plot_all(self, model=None, plot_col='order_count'):
        inputs, labels = self.example
        plt.figure(figsize=(15, 5))    # 设置单个大图的宽高比例
        plot_col_index = self.column_indices[plot_col]

        # 绘制整个输入数据
        for n in range(len(inputs)):    # 循环整个批次数据
                plt.plot(self.input_indices, inputs[n, :, plot_col_index],
                                 label='Inputs' if n == 0 else "", marker='.', zorder=-10)

        if self.label_columns:
                label_col_index = self.label_columns_indices.get(plot_col, None)
        else:
                label_col_index = plot_col_index

        if label_col_index is not None:
                # 绘制所有标签数据
                for n in range(len(labels)):
                        plt.scatter(self.label_indices, labels[n, :, label_col_index],
                                            edgecolors='k', label='Labels' if n == 0 else "", c='#2ca02c', s=64)

        if model is not None:
                predictions = model(inputs)
                # 绘制所有预测数据
                for n in range(len(predictions)):
                        plt.scatter(self.label_indices, predictions[n, :, label_col_index],
                                            marker='X', edgecolors='k', label='Predictions' if n == 0 else "",
                                            c='#ff7f0e', s=64)

        plt.ylabel(f'{plot_col} [normed]')
        plt.xlabel('Time [h]')
        plt.legend()
        plt.show()

# 添加到 WindowGenerator 类中
WindowGenerator.plot_all = plot_all
```

```
1
```

```
1
```

```
# df3 = df2
# print(df3)

# 四舍五入到最近的0.5分钟
# df3['time_0.5min'] = df3['platform_order_time_date'].dt.round('30S')  # '30S' 表示 30 秒，即 0.5 分钟

# # 按照 'time_0.5min' 和 'h3_long' 列进行分组，计算每组的订单数量
# df3 = df3.groupby(['platform_order_time','time_0.5min', 'h3_long']).size().reset_index(name='order_count')

# # 打印结果
# print(df3)
```

```
# df2['time_slot'] = df2['platform_order_time_date'].dt.floor('15min')  # 按 15 分钟切分
# order_counts = df2.groupby('time_slot').size()  # 统计每个时间段的订单量

# # 绘制图表
# plt.figure(figsize=(12, 6))
# order_counts.plot(kind='line')

# plt.title('Order Count per 15 Minutes')
# plt.xlabel('Time Slot')
# plt.ylabel('Order Count')
# plt.xticks(rotation=45)
# plt.tight_layout()
# plt.show()
```

```
class ConvLSTMCell(nn.Module):

    def __init__(self, input_dim, hidden_dim, kernel_size, bias):
        """
        Initialize ConvLSTM cell.

        Parameters
        ----------
        input_dim: Number of channels of input tensor.
        hidden_dim: Number of channels of hidden state.
        kernel_size: (int, int) Size of the convolutional kernel.
        bias: bool Whether or not to add the bias.
        """

        super(ConvLSTMCell, self).__init__()

        self.input_dim = input_dim             #经度，纬度，h3格子，should be 3
        self.hidden_dim = hidden_dim           #可以暂定16

        self.kernel_size = kernel_size
        self.padding = kernel_size[0] // 2, kernel_size[1] // 2
        self.bias = bias

        self.conv = nn.Conv2d(in_channels=self.input_dim + self.hidden_dim,
                              out_channels=4 * self.hidden_dim,
                              kernel_size=self.kernel_size,
                              padding=self.padding,
                              bias=self.bias)

    def forward(self, input_tensor, cur_state):
        h_cur, c_cur = cur_state

        combined = torch.cat([input_tensor, h_cur], dim=1)  # concatenate along channel axis

        combined_conv = self.conv(combined)
        cc_i, cc_f, cc_o, cc_g = torch.split(combined_conv, self.hidden_dim, dim=1)
        i = torch.sigmoid(cc_i)
        f = torch.sigmoid(cc_f)
        o = torch.sigmoid(cc_o)
        g = torch.tanh(cc_g)

        c_next = f * c_cur + i * g
        h_next = o * torch.tanh(c_next)

        return h_next, c_next

    def init_hidden(self, batch_size, image_size):
        height, width = image_size
        return (torch.zeros(batch_size, self.hidden_dim, height, width, device=self.conv.weight.device),
                torch.zeros(batch_size, self.hidden_dim, height, width, device=self.conv.weight.device))
```

```

```

```

```

```
# df2.describe().transpose()
```

```
features = ['platform_order_time', 'recipient_lng', 'recipient_lat', 'Day sin', 'Day cos', 'h3_long']
df_features = df2[features]

scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(df_features[features].values)
```

```
print(df_features)

#  创建时间序列数据集
def  create_dataset(data,  time_steps):
        X,  y  =  [],  []
        for  i  in  range(len(data)  -  time_steps):
                X.append(data[i:(i  +  time_steps),  :-1])    #  所有特征，但不包括最后一列（订单量）
                y.append(data[i  +  time_steps  -  1,  -1])    #  预测下一个订单量
        return  np.array(X),  np.array(y)

#  将数据分为输入和输出
X,  y  =  create_dataset(data_scaled,  time_steps=10)    #  选择时间步长为10

#  将  X  调整为五维  (样本数,  时间步,  特征数,  高度,  宽度)
#  假设每个样本在空间上是一个1x1的“图像”
X  =  X.reshape((X.shape[0],  X.shape[1],  X.shape[2],  1,  1))    #  (样本数,  时间步,  特征数,  1,  1)
y  =  y.reshape(-1,  1)    #  输出调整为  (样本数,  1)









def  create_dataset(data,  time_steps):
        X,  y  =  [],  []
        for  i  in  range(len(data)  -  time_steps):
                X.append(data[i:(i  +  time_steps),  :-1])    #  所有特征，但不包括最后一列
                y.append(data[i  +  time_steps  -  1,  -1])    #  预测下一个订单量
        return  np.array(X),  np.array(y)

#  将  DataFrame  转换为  numpy  数组并归一化
data_array  =  df_features.values.astype(np.float32)
X,  y  =  create_dataset(data_array,  time_steps=10)    #  选择时间步长为10

#  X  的形状为  (样本数,  时间步,  特征数)
X  =  X.reshape((X.shape[0],  10,  len(features)  -  1))    #  (样本数,  时间步,  特征数)
```

```
         platform_order_time  recipient_lng  recipient_lat   Day sin   Day cos  \
0                 1665934777      174580449       45824914 -0.818192 -0.574946
1                 1665935139      174481572       45876511 -0.833042 -0.553210
2                 1665935365      174527950       45815592 -0.842021 -0.539445
3                 1665935379      174547139       45897170 -0.842570 -0.538587
4                 1665935707      174529156       45880736 -0.855176 -0.518338
...                      ...            ...            ...       ...       ...
546360            1666627171      174545570       45868856 -0.864969 -0.501825
546361            1666627173      174569259       45879156 -0.865042 -0.501699
546362            1666627181      174941500       46045530 -0.865334 -0.501196
546363            1666627181      174536704       45905336 -0.865334 -0.501196
546364            1666627188      174549457       45891735 -0.865589 -0.500756

                   h3_long
0       617883434135846911
1       617883427761291263
2       617883434159964159
3       617883433536847871
4       617883433612607487
...                    ...
546360  617883434119331839
546361  617883433607626751
546362  617394156985647103
546363  617883433529769983
546364  617883433537110015

[546365 rows x 6 columns]
```

```
1
```

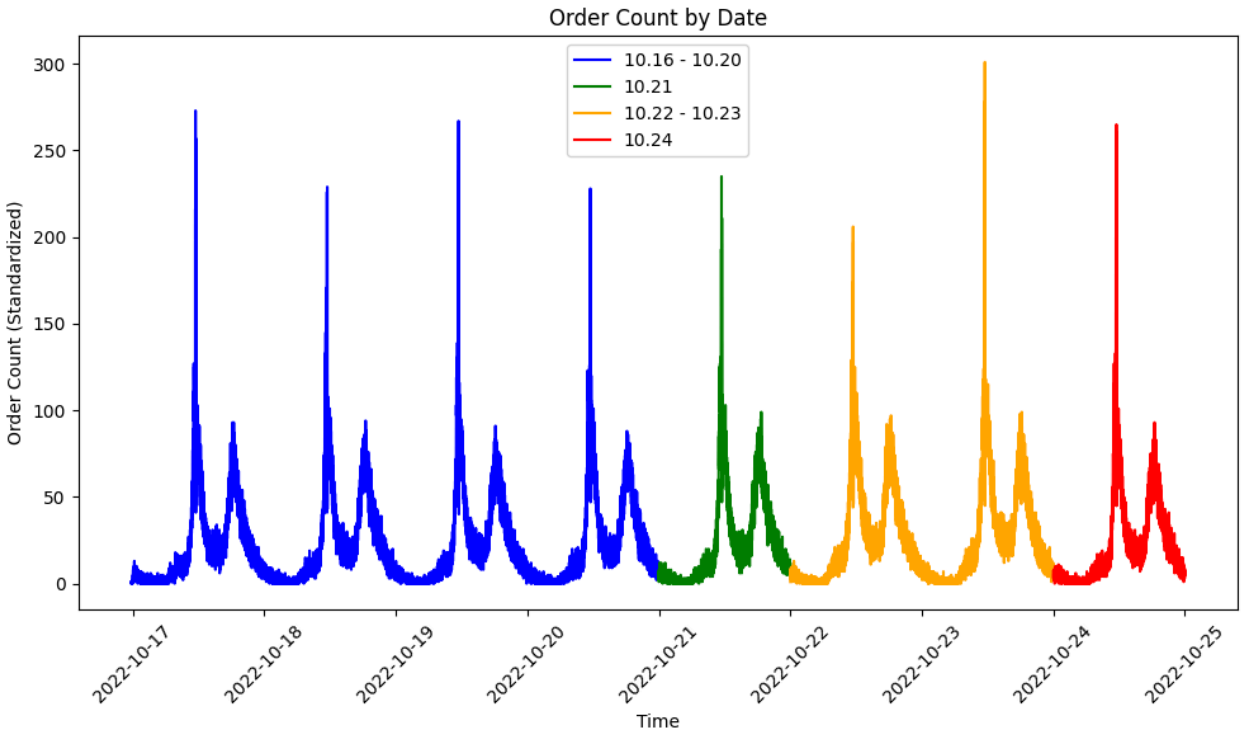
```
1
```

```
1
```

```
1
```

```
1
```

```
1
2
3
```




```
1
2
```

```
1
```

```
1 n = len(df2)
2 # train_df = df2[0:int(n*0.7)]
3 # val_df = df[int(n*0.7):int(n*0.9)]
4 # test_df = df2[int(n*0.9):]
5
6 num_features = 5    #lat,lon,sin,cos,timestamp
7
8 train_df = df2[
9         (df2['platform_order_time_date'] >= '2022-10-16') &
10        (df2['platform_order_time_date'] < '2022-10-21')
11 ]
12 val_df = df2[
13        (df2['platform_order_time_date'] >= '2022-10-21') &
14        (df2['platform_order_time_date'] < '2022-10-22')
15 ]
16 test_df = df2[
17        (df2['platform_order_time_date'] >= '2022-10-24') &
18        (df2['platform_order_time_date'] < '2022-10-25')
19 ]
20
21 train_df.drop(['is_prebook'], axis=1, inplace=True)
22 train_df.drop(['order_push_time'], axis=1, inplace=True)
23 train_df.drop(['estimate_meal_prepare_time'], axis=1, inplace=True)
24 train_df.drop(['sender_lng'], axis=1, inplace=True)
25 train_df.drop(['sender_lat'], axis=1, inplace=True)
26 train_df.drop(['platform_order_time_date'], axis=1, inplace=True)
27 train_df.drop(['order_push_time_date'], axis=1, inplace=True)
28 train_df.drop(['estimate_meal_prepare_time_date'], axis=1, inplace=True)
29
30 val_df.drop(['is_prebook'], axis=1, inplace=True)
31 val_df.drop(['order_push_time'], axis=1, inplace=True)
32 val_df.drop(['estimate_meal_prepare_time'], axis=1, inplace=True)
33 val_df.drop(['sender_lng'], axis=1, inplace=True)
34 val_df.drop(['sender_lat'], axis=1, inplace=True)
35 val_df.drop(['platform_order_time_date'], axis=1, inplace=True)
36 val_df.drop(['order_push_time_date'], axis=1, inplace=True)
37 val_df.drop(['estimate_meal_prepare_time_date'], axis=1, inplace=True)
38
39 test_df.drop(['is_prebook'], axis=1, inplace=True)
40 test_df.drop(['order_push_time'], axis=1, inplace=True)
41 test_df.drop(['estimate_meal_prepare_time'], axis=1, inplace=True)
42 test_df.drop(['sender_lng'], axis=1, inplace=True)
43 test_df.drop(['sender_lat'], axis=1, inplace=True)
44 test_df.drop(['platform_order_time_date'], axis=1, inplace=True)
45 test_df.drop(['order_push_time_date'], axis=1, inplace=True)
46 test_df.drop(['estimate_meal_prepare_time_date'], axis=1, inplace=True)
47
```

```
train_mean = train_df.mean()
train_std = train_df.std()

train_df = (train_df - train_mean) / train_std #fit完还是df
val_df = (val_df - train_mean) / train_std
test_df = (test_df - train_mean) / train_std


# scaler = StandardScaler()
# train_df = scaler.fit_transform(train_df) #fit完变成了array了


print(num_features)
```

```
<ipython-input-93-fc0b0ddbbee4>:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df.drop(['is_prebook'], axis=1, inplace=True)
<ipython-input-93-fc0b0ddbbee4>:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df.drop(['order_push_time'], axis=1, inplace=True)
<ipython-input-93-fc0b0ddbbee4>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df.drop(['estimate_meal_prepare_time'], axis=1, inplace=True)
<ipython-input-93-fc0b0ddbbee4>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df.drop(['sender_lng'], axis=1, inplace=True)
<ipython-input-93-fc0b0ddbbee4>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df.drop(['sender_lat'], axis=1, inplace=True)
<ipython-input-93-fc0b0ddbbee4>:26: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df.drop(['platform_order_time_date'], axis=1, inplace=True)
<ipython-input-93-fc0b0ddbbee4>:27: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df.drop(['order_push_time_date'], axis=1, inplace=True)
<ipython-input-93-fc0b0ddbbee4>:28: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df.drop(['estimate_meal_prepare_time_date'], axis=1, inplace=True)
<ipython-input-93-fc0b0ddbbee4>:30: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  val_df.drop(['is_prebook'], axis=1, inplace=True)
<ipython-input-93-fc0b0ddbbee4>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  val_df.drop(['order_push_time'], axis=1, inplace=True)
<ipython-input-93-fc0b0ddbbee4>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  val_df.drop(['estimate_meal_prepare_time'], axis=1, inplace=True)
<ipython-input-93-fc0b0ddbbee4>:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
class WindowGenerator():
    def __init__(self, input_width, label_width, shift,
                 train_df=train_df, val_df=val_df, test_df=test_df,
                 label_columns=None):
        # Store the raw data.
        self.train_df = train_df
        self.val_df = val_df
        self.test_df = test_df

        # Work out the label column indices.
        self.label_columns = label_columns
        if label_columns is not None:
            self.label_columns_indices = {name: i for i, name in
                                          enumerate(label_columns)}
        self.column_indices = {name: i for i, name in
                               enumerate(train_df.columns)}
```

```
        # Work out the window parameters.
        self.input_width = input_width
        self.label_width = label_width
        self.shift = shift

        self.total_window_size = input_width + shift

        self.input_slice = slice(0, input_width)
        self.input_indices = np.arange(self.total_window_size)[self.input_slice]

        self.label_start = self.total_window_size - self.label_width
        self.labels_slice = slice(self.label_start, None)
        self.label_indices = np.arange(self.total_window_size)[self.labels_slice]

    def __repr__(self):
        return '\n'.join([
                f'Total window size: {self.total_window_size}',
                f'Input indices: {self.input_indices}',
                f'Label indices: {self.label_indices}',
                f'Label column name(s): {self.label_columns}'])
```

```
def split_window(self, features):
    inputs = features[:, self.input_slice, :]
    labels = features[:, self.labels_slice, :]
    if self.label_columns is not None:
        labels = tf.stack(
                [labels[:, :, self.column_indices[name]] for name in self.label_columns],
                axis=-1)

    # Slicing doesn't preserve static shape information, so set the shapes
    # manually. This way the `tf.data.Datasets` are easier to inspect.
    inputs.set_shape([None, self.input_width, None])
    labels.set_shape([None, self.label_width, None])

    return inputs, labels

WindowGenerator.split_window = split_window
```

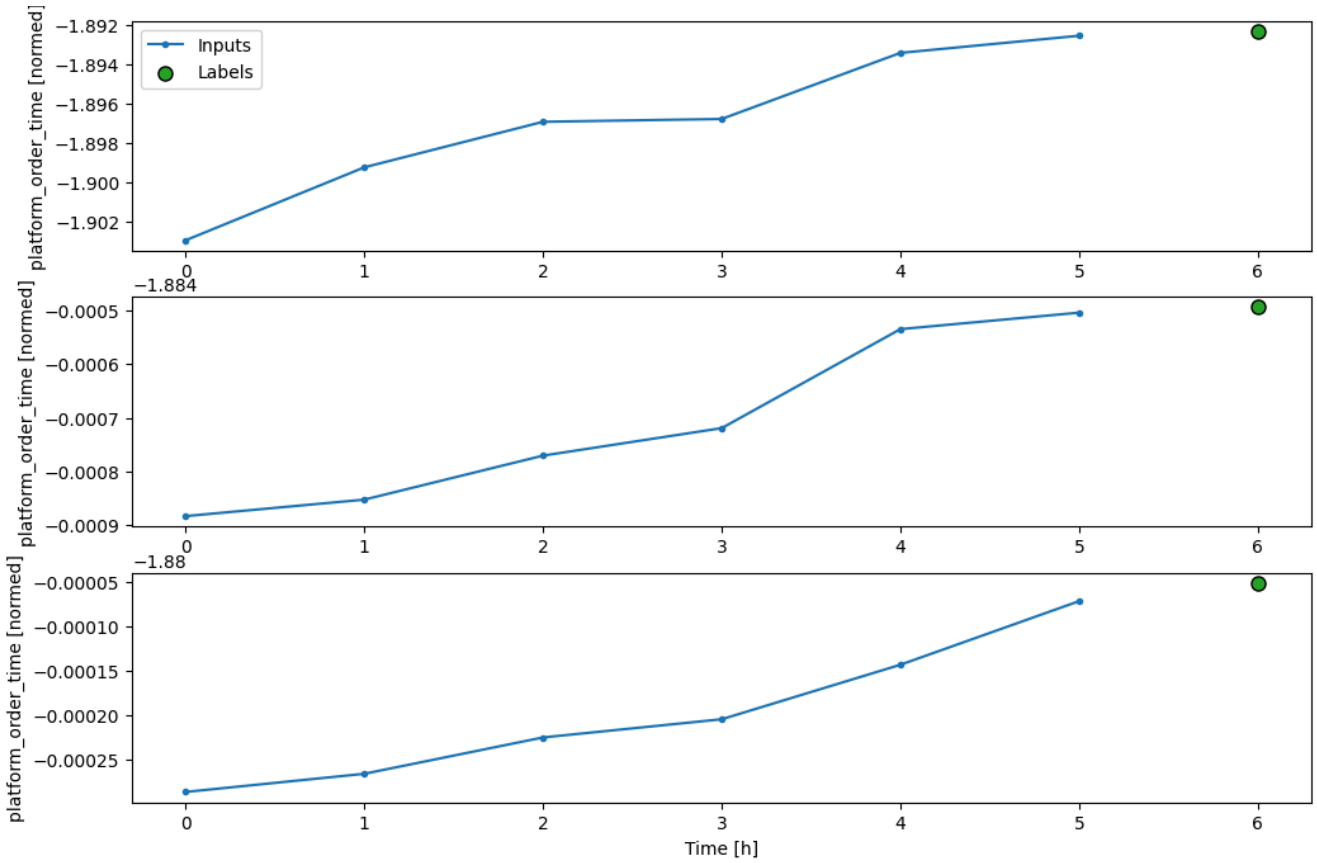
```
w2 = WindowGenerator(input_width=6, label_width=1, shift=1,
                                    label_columns=['platform_order_time'])

# Stack three slices, the length of the total window.
example_window = tf.stack([np.array(train_df[:w2.total_window_size]),
                                                np.array(train_df[100:100+w2.total_window_size]),
                                                np.array(train_df[200:200+w2.total_window_size])])

example_inputs, example_labels = w2.split_window(example_window)

print('All shapes are: (batch, time, features)')
print(f'Window shape: {example_window.shape}')
print(f'Inputs shape: {example_inputs.shape}')
print(f'Labels shape: {example_labels.shape}')
```

```
All shapes are: (batch, time, features)
Window shape: (3, 7, 5)
Inputs shape: (3, 6, 5)
Labels shape: (3, 1, 1)
```

```
w2.example = example_inputs, example_labels

def plot(self, model=None, plot_col='platform_order_time', max_subplots=3):
    inputs, labels = self.example
    plt.figure(figsize=(12, 8))
    plot_col_index = self.column_indices[plot_col]
    max_n = min(max_subplots, len(inputs))
    for n in range(max_n):
        plt.subplot(max_n, 1, n+1)
        plt.ylabel(f'{plot_col} [normed]')
        plt.plot(self.input_indices, inputs[n, :, plot_col_index],
                         label='Inputs', marker='.', zorder=-10)

        if self.label_columns:
            label_col_index = self.label_columns_indices.get(plot_col, None)
        else:
            label_col_index = plot_col_index

        if label_col_index is None:
            continue

        plt.scatter(self.label_indices, labels[n, :, label_col_index],
                            edgecolors='k', label='Labels', c='#2ca02c', s=64)
        if model is not None:
            predictions = model(inputs)
```

```
            plt.scatter(self.label_indices,  predictions[n,  :,  label_col_index],
                                marker='X',  edgecolors='k',  label='Predictions',
                                c='#ff7f0e',  s=64)

        if  n  ==  0:
            plt.legend()

    plt.xlabel('Time  [h]')

WindowGenerator.plot  =  plot
```

```
w2.plot()
w2.plot(plot_col='p  (mbar)')
```




```
1
```

```
1
```

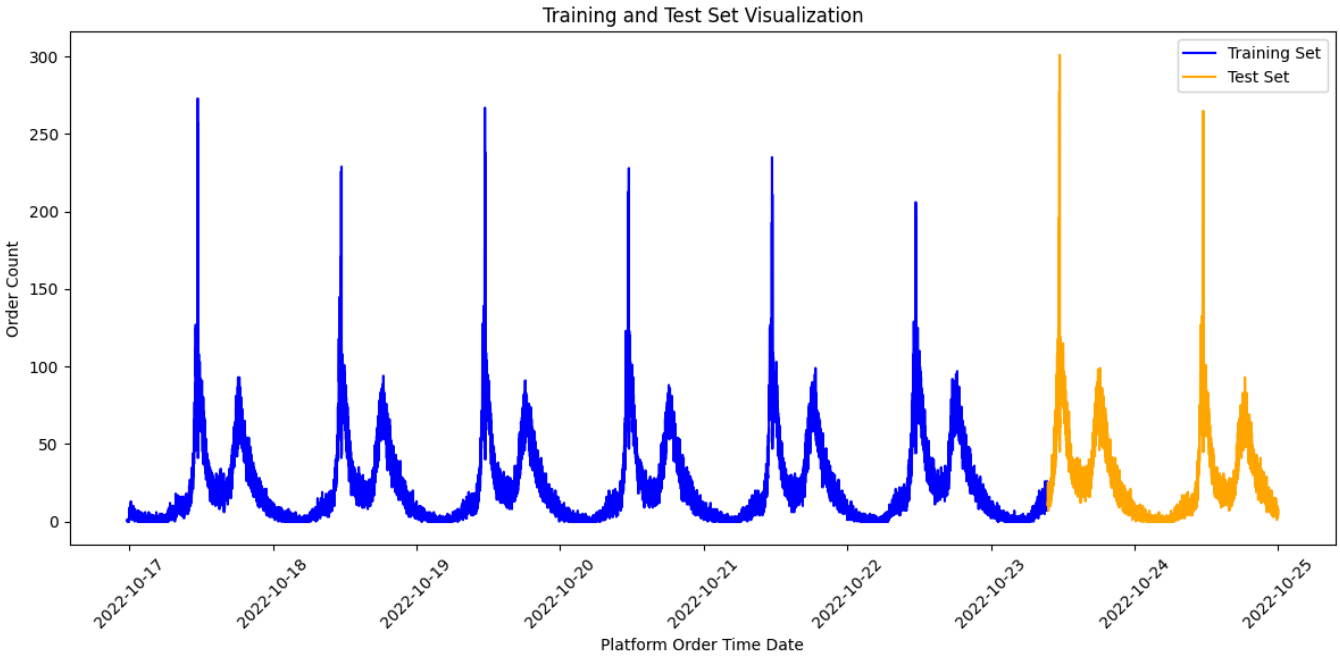
```
1
```

```
1
```

```
df_grouped_15min  =  df_grouped_15min.sort_values(by='platform_order_time_date')

#  df_grouped_15min_train_set,  df_grouped_15min_test_set  =  train_test_split(df_grouped_15min,  test_size=0.2,  random_state=42)

train_size  =  int(len(df_grouped_15min)  *  0.8)
df_grouped_15min_train_set  =  df_grouped_15min[:train_size]    #  前80%训练集
df_grouped_15min_test_set  =  df_grouped_15min[train_size:]      #  后20%测试集

#  定义特征和目标
X_train  =  df_grouped_15min_train_set[['platform_order_time_date']]    #  特征
y_train  =  df_grouped_15min_train_set['order_count']                                    #  目标

X_test  =  df_grouped_15min_test_set[['platform_order_time_date']]        #  特征
y_test  =  df_grouped_15min_test_set['order_count']                                      #  目标

#  输出训练集和测试集的大小
print("训练集大小:",  X_train.shape,  y_train.shape)
print("测试集大小:",  X_test.shape,  y_test.shape)

plt.figure(figsize=(12,  6))
plt.plot(df_grouped_15min_train_set['platform_order_time_date'],  df_grouped_15min_train_set['order_count'],  label='Training  Set',  color='blue')
plt.plot(df_grouped_15min_test_set['platform_order_time_date'],  df_grouped_15min_test_set['order_count'],  label='Test  Set',  color='orange')

plt.title('Training  and  Test  Set  Visualization')
```

```
plt.xlabel('Platform Order Time Date')
plt.ylabel('Order Count')
plt.xticks(rotation=45)

plt.legend()
plt.tight_layout()
plt.show()
```

```
训练集大小: (18464, 1) (18464,)
测试集大小: (4617, 1) (4617,)
```




```
grouped_1min_train = df_grouped_15min_train_set['order_count'].values.reshape(-1, 1)

X_train = []
y_train = []

# time_steps = 30 #loss:0.96
time_steps = 400 #loss:1.0255
#time_steps = 2500 8 hours

# 构建输入和目标
for i in range(time_steps, len(grouped_1min_train)):
        X_train.append(grouped_1min_train[i-time_steps:i, 0])    # 取过去 50 个时间步
        y_train.append(grouped_1min_train[i, 0])                 # 当前时间步的值


X_train, y_train = np.array(X_train), np.array(y_train)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)      # (样本数, 时间步数, 特征数)

print("训练集大小:", X_train.shape, y_train.shape)
```

```
训练集大小: (18064, 400, 1) (18064,)
```

```
grouped_1min_test = df_grouped_15min_test_set['order_count'].values.reshape(-1, 1)

X_test = []
y_test = []

for i in range(time_steps, len(grouped_1min_test)):
        X_test.append(grouped_1min_test[i-time_steps:i, 0])
        y_test.append(grouped_1min_test[i, 0])


X_test, y_test = np.array(X_test), np.array(y_test)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)


print("测试集大小:", X_test.shape, y_test.shape)
```

```
测试集大小: (4217, 400, 1) (4217,)
```

```
# importing libraries
from keras.models import Sequential
```

```
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import Dropout
from keras.layers import GRU, Bidirectional
from keras.optimizers import SGD
from sklearn import metrics
from sklearn.metrics import mean_squared_error
```

```
# initializing the RNN
regressor = Sequential()

# adding RNN layers and dropout regularization
regressor.add(SimpleRNN(units = 50,
                        activation = "tanh",
                        return_sequences = True,
                        input_shape = (X_train.shape[1],1)))
regressor.add(Dropout(0.2))

regressor.add(SimpleRNN(units = 50,
                        activation = "tanh",
                        return_sequences = True))

regressor.add(SimpleRNN(units = 50,
                        activation = "tanh",
                        return_sequences = True))

regressor.add( SimpleRNN(units = 50))

# adding the output layer
regressor.add(Dense(units = 1,activation='sigmoid'))

# compiling RNN
regressor.compile(optimizer = SGD(learning_rate=0.01,
                                  decay=1e-6,
                                  momentum=0.9,
                                  nesterov=True),
                  loss = "mean_squared_error")

# fitting the model
regressor.fit(X_train, y_train, epochs = 3, batch_size = 2)
regressor.summary()
```

```
Epoch 1/3
9032/9032 ━━━━━━━━━━━━━━━━━━━━ 2865s 317ms/step - loss: 0.9684
Epoch 2/3
9032/9032 ━━━━━━━━━━━━━━━━━━━━ 2917s 319ms/step - loss: 0.9530
Epoch 3/3
9032/9032 ━━━━━━━━━━━━━━━━━━━━ 2897s 318ms/step - loss: 1.0254
Model: "sequential_7"
```

Layer (type)	Output Shape	Param #
simple_rnn_28 (SimpleRNN)	(None, 400, 50)	2,600
dropout_7 (Dropout)	(None, 400, 50)	0
simple_rnn_29 (SimpleRNN)	(None, 400, 50)	5,050
simple_rnn_30 (SimpleRNN)	(None, 400, 50)	5,050
simple_rnn_31 (SimpleRNN)	(None, 50)	5,050
dense_7 (Dense)	(None, 1)	51

```
Total params: 35,604 (139.08 KB)
Trainable params: 17,801 (69.54 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 17,803 (69.55 KB)
```

```
y_RNN = regressor.predict(X_test)
print(y_RNN)
```

```
132/132 ━━━━━━━━━━━━━━━━━━━━ 23s 150ms/step
[[0.00012935]
 [0.00012935]
 [0.00012935]
 ...
 [0.0001293 ]
 [0.0001293 ]
 [0.0001293 ]]
```

```
grouped_1min_train = df_grouped_15min_train_set['order_count'].values.reshape(-1, 1)

X_train = []
y_train = []
```

```
# time_steps = 30 #loss:0.96
time_steps = 3 #loss:1.0255
#time_steps = 2500 8 hours

# 构建输入和目标
for i in range(time_steps, len(grouped_1min_train)):
        X_train.append(grouped_1min_train[i-time_steps:i, 0])    # 取过去 50 个时间步
        y_train.append(grouped_1min_train[i, 0])                                # 当前时间步的值


X_train, y_train = np.array(X_train), np.array(y_train)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)      # (样本数, 时间步数, 特征数

print("训练集大小:", X_train.shape, y_train.shape)

grouped_1min_test = df_grouped_15min_test_set['order_count'].values.reshape(-1, 1)

X_test = []
y_test = []

for i in range(time_steps, len(grouped_1min_test)):
        X_test.append(grouped_1min_test[i-time_steps:i, 0])
        y_test.append(grouped_1min_test[i, 0])


X_test, y_test = np.array(X_test), np.array(y_test)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)


print("测试集大小:", X_test.shape, y_test.shape)
```

```
训练集大小: (18461, 3, 1) (18461,)
测试集大小: (4614, 3, 1) (4614,)
```

```
# initializing the RNN
regressor = Sequential()

# adding RNN layers and dropout regularization
regressor.add(SimpleRNN(units = 500,
                                            activation = "tanh",
                                            return_sequences = True,
                                            input_shape = (X_train.shape[1],1)))
regressor.add(Dropout(0.2))

regressor.add( SimpleRNN(units = 500))

# adding the output layer
regressor.add(Dense(units = 1))

# compiling RNN
regressor.compile(optimizer = SGD(learning_rate=0.01
```