```
# from google.colab import drive
# drive.mount('/content/drive')
```

```
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
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.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('/content/all_waybill_info_meituan_0322.csv')
# df = pd.read_csv('/content/all_waybill_info_meituan_0322.csv')
# df = pd.read_csv('/content/drive/MyDrive/NYU/24Fall1/CS Cap/all_waybill_info_meituan_0322.csv')
# df = pd.read_csv('/content/drive/MyDrive/NYU/24Fall1/CS Cap/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 7.9 MB/s eta 0:00:00
Installing collected packages: h3
Successfully installed h3-4.1.2
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-2-ef873380e98c> in <cell line: 32>()
     30 # %matplotlib widget
     31
---> 32 df = pd.read_csv('/content/all_waybill_info_meituan_0322.csv')
     33 # df = pd.read_csv('/content/all_waybill_info_meituan_0322.csv')
     34 # df = pd.read_csv('/content/drive/MyDrive/NYU/24Fall1/CS Cap/all_waybill_info_meituan_0322.csv')

                                 4 frames
/usr/local/lib/python3.10/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    871         if ioargs.encoding and "b" not in ioargs.mode:
    872             # Encoding
--> 873             handle = open(
    874                 handle,
    875                 ioargs.mode,
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
#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

#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'])
df2 = df2.drop(columns=['is_prebook'])


#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

df2['Day sin'] = np.sin(df2['platform_order_time'] * (2 * np.pi / day))
df2['Day cos'] = np.cos(df2['platform_order_time'] * (2 * np.pi / day))
```

```
def compute_h3_and_boundaries(row, resolution=9):
        lng = row['recipient_lng']/1000000
        lat = row['recipient_lat']/1000000
        h3_index = h3.latlng_to_cell(lat, lng, resolution)
        # print(h3_index)
        # boundaries = h3.cell_to_boundary(h3_index)
        # print(lng,lat,h3_index,boundaries)
        return pd.Series([h3_index])

df2['sender_lng'] = pd.to_numeric(df2['sender_lng'], errors='coerce')
df2['sender_lat'] = pd.to_numeric(df2['sender_lat'], errors='coerce')
df2['recipient_lng'] = pd.to_numeric(df2['recipient_lng'], errors='coerce')
df2['recipient_lat'] = pd.to_numeric(df2['recipient_lat'], errors='coerce')

df2[['H3_Index']] = df2.apply(compute_h3_and_boundaries, axis=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
# resolution=9 共1711个cell
```

```
from sklearn.preprocessing import LabelEncoder

# Label encode H3 indices into integer IDs
encoder = LabelEncoder()
df2['H3_Index_ID'] = encoder.fit_transform(df2['H3_Index'])    # Maps H3_Index to unique integer IDs
```

```
df2
```

```

```

```
df2[['H3_Hex_Groups']]
```

```
df3 = df2
```

```
# 将 H3_Index 从十六进制转为二进制，并提取前19位
df3['H3_Binary'] = df3['H3_Index'].apply(lambda x: bin(int(x, 16))[2:].zfill(64))    # 转为64位二进制
df3['H3_Binary_19'] = df3['H3_Binary'].str[:19]    # 提取前19位

# 提取后45位的分组并转换成十六进制
df3['H3_Binary_Last45'] = df3['H3_Binary'].str[-45:]    # 后45位
df3['H3_Binary_Groups'] = df3['H3_Binary_Last45'].apply(lambda x: [x[i:i+3] for i in range(0, 45, 3)])
df3['H3_Hex_Groups'] = df3['H3_Binary_Groups'].apply(lambda groups: [int(g, 2) for g in groups if g != '111'])    # 转为0-6的数值并过滤掉7

# 定义CNN所需的输入维度（例如3x3矩阵）
target_shape = (3, 3)

# 将每个坐标数据reshape成CNN所需的输入格式
df3['H3_CNN_Input'] = df3['H3_Hex_Groups'].apply(lambda x: np.array(x[:9]).reshape(target_shape) if len(x) >= 9 else np.pad(np.array(x), (0, 9
# 选择并保存最终所需列
df_output = df3[['platform_order_time', 'recipient_lng', 'recipient_lat', 'sender_lng', 'sender_lat',
                 'platform_order_time_date', 'Day sin', 'Day cos', 'H3_Index', 'H3_Long',
```

```
                            'H3_Binary_19','H3_Hex_Groups', 'H3_CNN_Input']]

# 保存到一个输出文件
# df_output.to_pickle("filtered_all_waybill_h3_cnn_input.pkl")  # 保存为pickle格式方便后续加载

# print("已保存所选列到文件：filtered_all_waybill_h3_cnn_input.pkl")
```

```
##########################################
# Define the embedding dimension for H3 indices
embedding_dim = 32  # Size of each embedding vector, can be tuned
h3_embedding_layer = tf.keras.layers.Embedding(
        input_dim=len(encoder.classes_),  # Vocabulary size = unique H3 indices
        output_dim=embedding_dim,
        name="H3_Embedding"
)
```

```
from tensorflow.keras import layers, models, Input
##################################################

# Define inputs for model
h3_index_input = Input(shape=(1,), name="H3_Index_ID")  # Integer-encoded H3 index
platform_order_time_input = Input(shape=(1,), name="Platform_Order_Time")  # Time-based feature
cnn_input = Input(shape=(3, 3, 1), name="H3_CNN_Input")  # 3x3 matrix for spatial features

# H3 Embedding
h3_embedding = h3_embedding_layer(h3_index_input)
h3_embedding = layers.Flatten()(h3_embedding)  # Flatten the output to 2D

# CNN for spatial feature extraction
x = layers.Conv2D(32, (2, 2), activation='relu')(cnn_input)
x = layers.Flatten()(x)
cnn_features = layers.Dense(32, activation='relu')(x)  # Processed spatial features

# Combine all inputs
merged = layers.Concatenate()([h3_embedding, cnn_features, platform_order_time_input])

# Fully connected layers
x = layers.Dense(64, activation='relu')(merged)
x = layers.Dense(32, activation='relu')(x)
output = layers.Dense(1, activation='linear')(x)

# Build and compile model
model = tf.keras.Model(inputs=[h3_index_input, cnn_input, platform_order_time_input], outputs=output)
model.compile(optimizer='adam', loss='mse')

# Model summary
model.summary()
```

```
# Prepare data for inputs
h3_index_data = df3['H3_Index_ID'].values.reshape(-1, 1)
cnn_data = np.stack(df3['H3_CNN_Input'].values)  # Assuming H3_CNN_Input is already reshaped to (3, 3)
platform_order_time_data = df3['platform_order_time'].values.reshape(-1, 1)

# Train model
model.fit([h3_index_data, cnn_data, platform_order_time_data], target_data, epochs=10, batch_size=32)
```

```
# import tensorflow as tf
# import numpy as np
# Embedding
model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(input_dim=1170, output_dim=3, input_length=8))
# input_dim=1000 means the vocabulary size is 1000
# output_dim=64 means each word is represented by a 64-dimensional vector
# input_length=10 means the sequence length for each input sample is 10

# input_array = df3['H3_Hex_Groups']
# input_array = np.array(df3['H3_Hex_Groups'].tolist())  # Ensure it's a NumPy array
#######################################3
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder to convert categorical values into integers
encoder = LabelEncoder()
df3['H3_Hex_Groups_Encoded'] = encoder.fit_transform(df3['H3_Hex_Groups'])

# Convert to NumPy array
input_array = np.array(df3['H3_Hex_Groups_Encoded'].tolist())

input_array = input_array.reshape(-1, 8)  # Reshaping to (batch_size, sequence_length)

#####################################3
```

```
# input_array = np.random.randint(1000, size=(32, 8))  # batch of 32 sequences, each of length 10
# model.compile('he_uniform', 'mse')
model.compile('rmsprop', 'mse')
output_array = model.predict(input_array)
print(output_array.shape)  # Expected shape: (32, 10, 64)
```

```
# # 输入层
# h3_index_input = layers.Input(shape=(1,), dtype=tf.string, name="H3_Index_Input")
# platform_order_time_input = layers.Input(shape=(1,), name="Platform_Order_Time")

# # 嵌入 H3_Index
# h3_index_ids = df_h3_lookup(h3_index_input)
# h3_embedding = h3_embedding_layer(h3_index_ids)

# # 合并输入特征
# merged = layers.Concatenate()([h3_embedding, cnn_features, platform_order_time_input])

# # 增加全连接层
# x = layers.Dense(64, activation='relu')(merged)
# x = layers.Dense(32, activation='relu')(x)
# output = layers.Dense(1, activation='linear')(x)

# # 构建模型
# model = tf.keras.Model(inputs=[h3_index_input, cnn_input, platform_order_time_input], outputs=output)
# model.compile(optimizer='adam', loss='mse')
```

```
from tensorflow.keras import layers, models
import tensorflow as tf

# 定义 CNN 特征提取层
input_shape = (3, 3, 1)  # 3x3 矩阵，单通道
cnn_input = layers.Input(shape=input_shape, name="H3_CNN_Input")

# 添加卷积和池化层
x = layers.Conv2D(32, (2, 2), activation='relu')(cnn_input)
x = layers.Flatten()(x)
cnn_features = layers.Dense(32, activation='relu')(x)  # 处理后作为 cnn_features

# 输入层定义
h3_index_input = layers.Input(shape=(1,), dtype=tf.string, name="H3_Index_Input")
platform_order_time_input = layers.Input(shape=(1,), name="Platform_Order_Time")

# 嵌入 H3_Index
h3_index_ids = df_h3_lookup(h3_index_input)  # StringLookup 转换
h3_embedding = h3_embedding_layer(h3_index_ids)  # 嵌入 H3_Index

# 合并输入特征
merged = layers.Concatenate()([h3_embedding, cnn_features, platform_order_time_input])

# 增加全连接层
x = layers.Dense(64, activation='relu')(merged)
x = layers.Dense(32, activation='relu')(x)
output = layers.Dense(1, activation='linear')(x)

# 构建模型
model = tf.keras.Model(inputs=[h3_index_input, cnn_input, platform_order_time_input], outputs=output)
model.compile(optimizer='adam', loss='mse')

# 检查模型结构
model.summary()
```

```
# # import numpy as np
# # import tensorflow as tf

# df_h3_lookup = tf.keras.layers.StringLookup()
```

```

```

```
# import tensorflow as tf

# # 创建 StringLookup 以便从 H3_Index 构建词汇表
# df_h3_lookup = tf.keras.layers.StringLookup()
# df_h3_lookup.adapt(df3['H3_Index'].astype(str))  # 转为字符串以兼容 StringLookup

# # 构建 H3_Index 嵌入层
# embedding_dim = 32
# h3_embedding_layer = tf.keras.layers.Embedding(
#         input_dim=len(df_h3_lookup.get_vocabulary()),  # 词汇表大小
#         output_dim=embedding_dim
# )
```

```

from tensorflow.keras import layers

# 定义 CNN 层，用于 3x3 H3 索引数据的特征提取
input_shape = (3, 3, 1)  # 3x3 矩阵，单通道
cnn_input = layers.Input(shape=input_shape, name="H3_CNN_Input")
x = layers.Conv2D(32, (2, 2), activation='relu')(cnn_input)
x = layers.Flatten()(x)
cnn_features = layers.Dense(32, activation='relu')(x)  # 替换 cnn_output 为可调用层对象
```

```
# ratings = tfds.load("movielens/100k-ratings", split="train")
```

```
# # df_h3_lookup.adapt(df3.map(lambda x: x["H3_Long"]))

# # print(f"Vocabulary: {df_h3_lookup.get_vocabulary()[:3]}")
# # 确保 H3_Long 是一个 Series 并且以适合 `adapt` 的形式传递
# df_h3_lookup.adapt(df3["H3_Long"].astype(str))
# print(f"Vocabulary: {df_h3_lookup.get_vocabulary()[:3]}")
```

```
# df_h3_bins = 200_000

# df_h3_hashing = tf.keras.layers.Hashing(
#         num_bins=df_h3_bins
# )
```

```
# # df_h3_embedding = tf.keras.layers.Embedding(
# #         # Let's use the explicit vocabulary lookup.
# #         input_dim=df_h3_lookup.vocab_size(),
# #         output_dim=32
# # )
# df_h3_embedding = tf.keras.layers.Embedding(
#         input_dim=len(df_h3_lookup.get_vocabulary()),  # 使用词汇表大小作为输入维度
#         output_dim=32
# )

```

```
from tensorflow.keras import layers

# 定义 CNN 层，用于 3x3 H3 索引数据的特征提取
input_shape = (3, 3, 1)  # 3x3 矩阵，单通道
cnn_input = layers.Input(shape=input_shape)
x = layers.Conv2D(32, (2, 2), activation='relu')(cnn_input)
x = layers.Flatten()(x)
cnn_output = layers.Dense(embedding_dim, activation='relu')(x)

```

```
h3_index_input = layers.Input(shape=(1,), dtype=tf.string, name="H3_Index_Input")
platform_order_time_input = layers.Input(shape=(1,), name="Platform_Order_Time")

# 嵌入 H3_Index
h3_index_ids = df_h3_lookup(h3_index_input)
h3_embedding = h3_embedding_layer(h3_index_ids)

# 处理 H3_CNN_Input
h3_cnn_input = layers.Input(shape=(3, 3, 1), name="H3_CNN_Input")
cnn_features = cnn_output(h3_cnn_input)

# 合并输入特征
merged = layers.Concatenate()([h3_embedding, cnn_features, platform_order_time_input])

# 增加全连接层
x = layers.Dense(64, activation='relu')(merged)
x = layers.Dense(32, activation='relu')(x)
output = layers.Dense(1, activation='linear')(x)

# 构建模型
model = tf.keras.Model(inputs=[h3_index_input, h3_cnn_input, platform_order_time_input], outputs=output)
model.compile(optimizer='adam', loss='mse')
```

```
# 使用 `model.fit()` 进行模型训练
# 假设 df_output 中的 target 列是我们想要预测的值
# model.fit([df_output['H3_Index'], df_output['H3_CNN_Input'], df_output['platform_order_time']], df_output['target'])

```