基于LSTM的南京市天气预测与分析

# 序言

天气预测是科学领域中的一个重要课题，天气预测不仅能为农业生产提供可靠的指导，还能为交通、能源等领域的调度和规划提供参考依据。特别是在气候变化愈发明显的今天，准确的天气预报能够帮助居民企业更好地应对突发天气，减少自然灾害对生产和生活的影响。

随着机器学习和深度学习的飞速发展，传统的天气预测方法逐渐被以人工智能为核心的新技术所取代。其中，长短期记忆网络（LSTM）作为一种擅长处理时间序列数据的模型，在天气预测中发挥了显著作用。本次研究中，爬取了基于南京市过去三个月的天气数据，用pytorch框架构建了一个LSTM模型，并利用该模型预测10月份南京一个月的风力和平均气温变化趋势。

# 背景

## 1. 天气预测的应用背景

天气预测一直是气象学中的核心课题，它涉及大气物理学、数学建模、数据分析等多个学科。传统的天气预测依赖于大量的气象观测数据和物理模型，例如基于数值天气预报模型的分析方法，这些方法虽然可以提供较为准确的天气预报，但对计算能力要求较高，而且处理复杂的气候数据时容易出现误差累积。

近年来，随着数据量的增加和计算能力的提升，基于机器学习的天气预测逐渐引起了广泛的关注。深度学习技术的快速发展，提供了处理大量气象数据的新途径。LSTM网络是一种循环神经网络，能够通过记忆机制解决长期依赖问题，因此适用于时间序列数据的预测任务。在本次报告中，通过爬取天气网站的历史三个月的天气数据，选取南京市的天气数据，使用LSTM模型进行未来的天气预测。

## 2. 数据集描述

本次研究的数据集包括2024年6月至2024年9月南京市的天气数据，数据包含多个关键气象指标：

最高气温：当天的最高温度，单位为摄氏度（°C）。

最低气温：当天的最低温度，单位为摄氏度（°C）。

天气状况：晴天、多云、雷阵雨等。

风向：东风、西风、东北风等。

风力：风力等级，取值为1到6级。

平均气温：当天的平均温度。

数据通过清洗和预处理后，成为模型的输入特征，供LSTM模型进行训练和预测。

# 原理

## 1. LSTM 网络原理

LSTM（Long Short-Term Memory）是循环神经网络（RNN）的变体。与传统的RNN相比，LSTM引入了三个门控机制：输入门、遗忘门和输出门，旨在解决RNN在处理长时间序列数据时出现的梯度消失和梯度爆炸问题。

LSTM的核心机制是通过记忆单元来控制信息的传递。具体而言，LSTM通过选择性地记住重要信息并丢弃不必要的信息，实现对长时间依赖关系的建模。在天气预测中，气象数据通常表现出显著的时间依赖性，例如当天的温度、湿度和风速与前几天的天气情况密切相关。因此，LSTM特别适合此类时间序列数据的预测。

输入门：控制当前输入的天气特征信息有多少可以进入到记忆单元。

遗忘门：决定上一个时间步的信息是否应该被遗忘。

输出门：决定从记忆单元输出多少信息，传递给下一个时间步。

## 2. 模型结构

在本次天气预测任务中，设计了一个改进的LSTM模型

| **模型层次** | **类型** | **输入维度** | **输出维度** | **备注** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 输入层 | LSTM | 6 | 64 | 特征数量为6，包括最高气温、最低气温、天气编码等 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LSTM 层 | LSTM | 64 | 64 | 使用两层 LSTM，增加模型的学习能力 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dropout 层 | Dropout | 64 | 64 | 采用0.2的dropout，防止过拟合 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 全连接层 | Linear | 64 | 2 | 输出预测的风力和平均气温 |

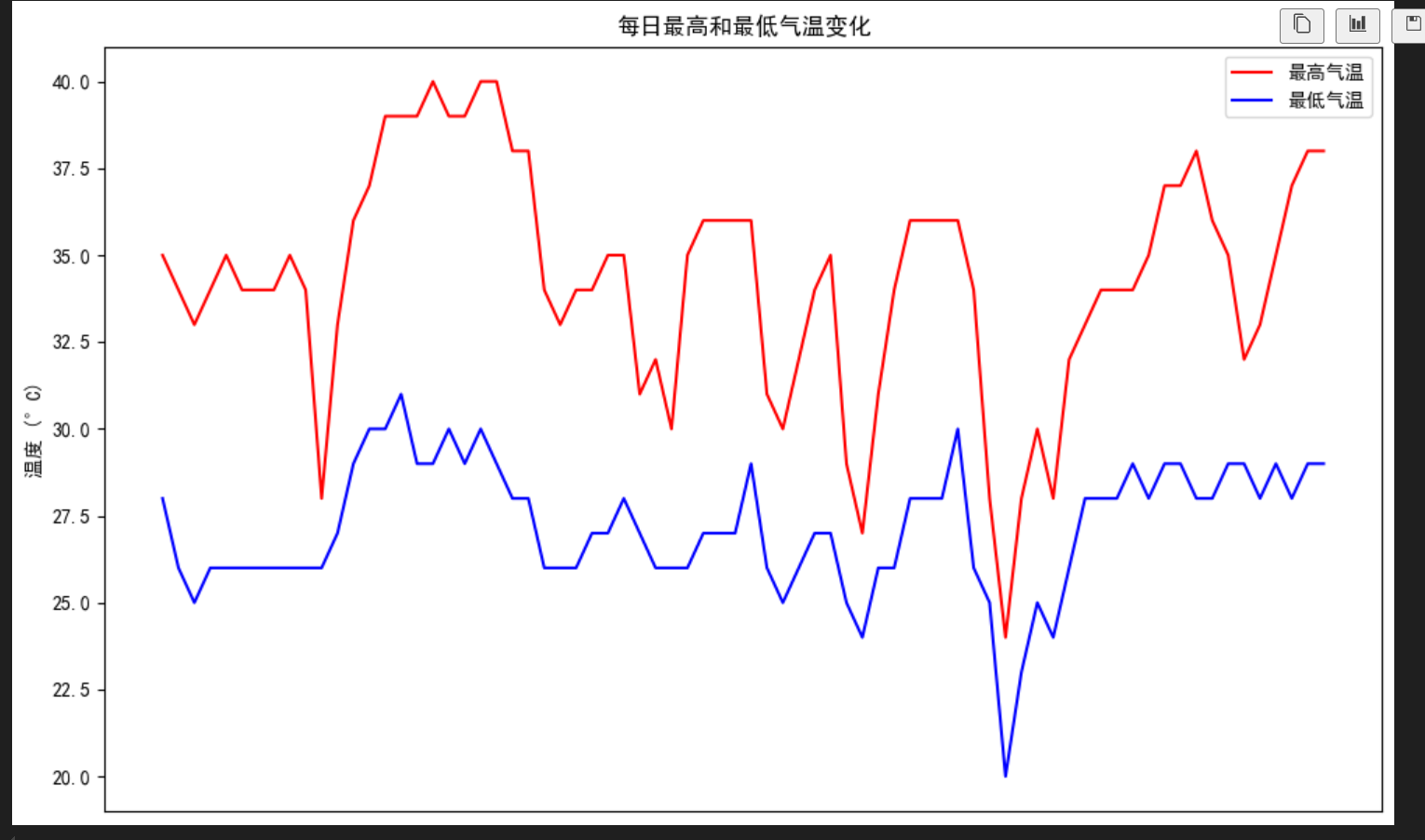
该模型的输入为过去7天的气象数据，输出为未来一天的风力和平均气温。通过循环的方式，能够预测未来31天的天气状况。

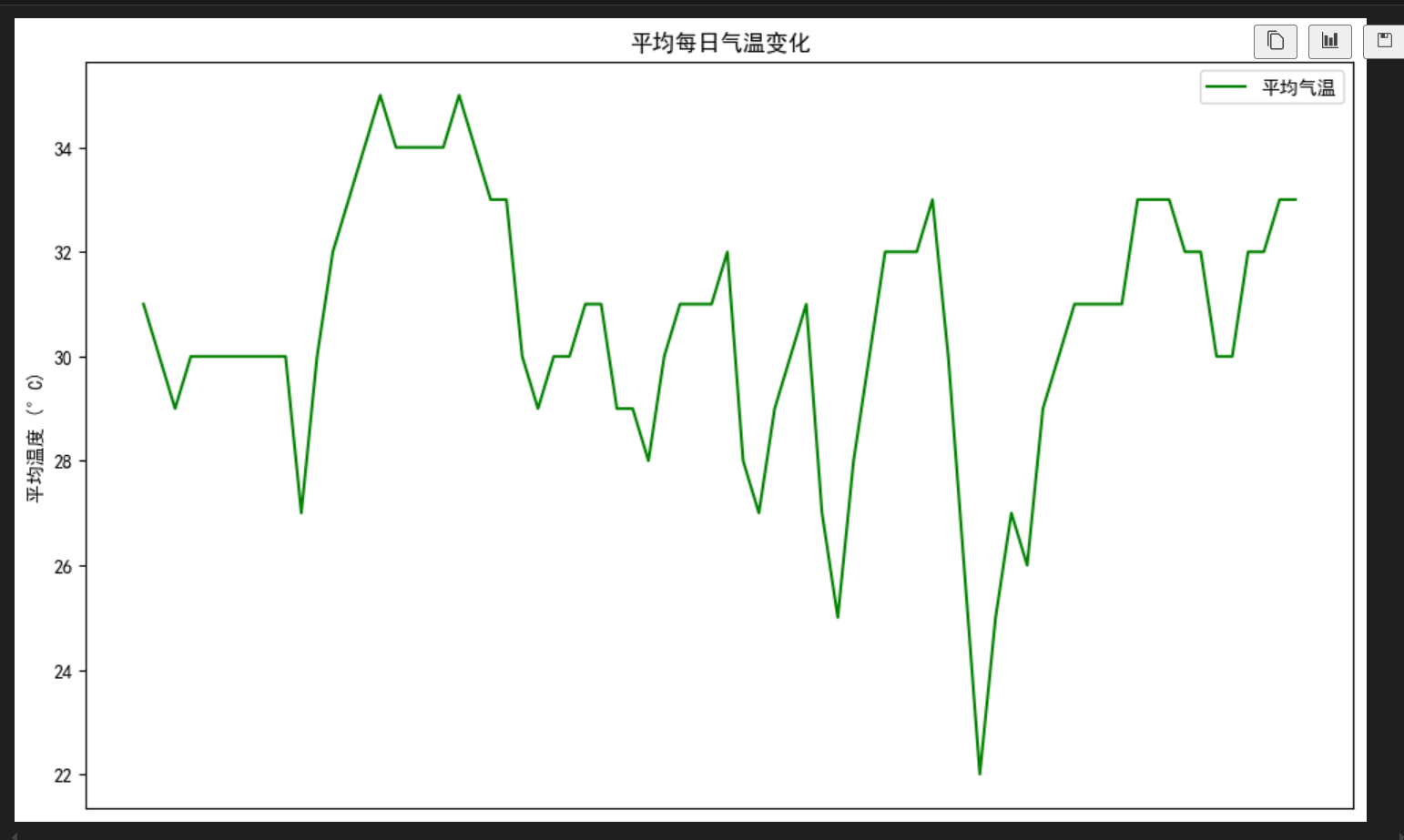
可视化分析

在数据预处理中，对原始数据进行了清理和编码。为了更好地理解数据的分布与变化趋势，进行了可视化分析

## 1. 气温变化趋势

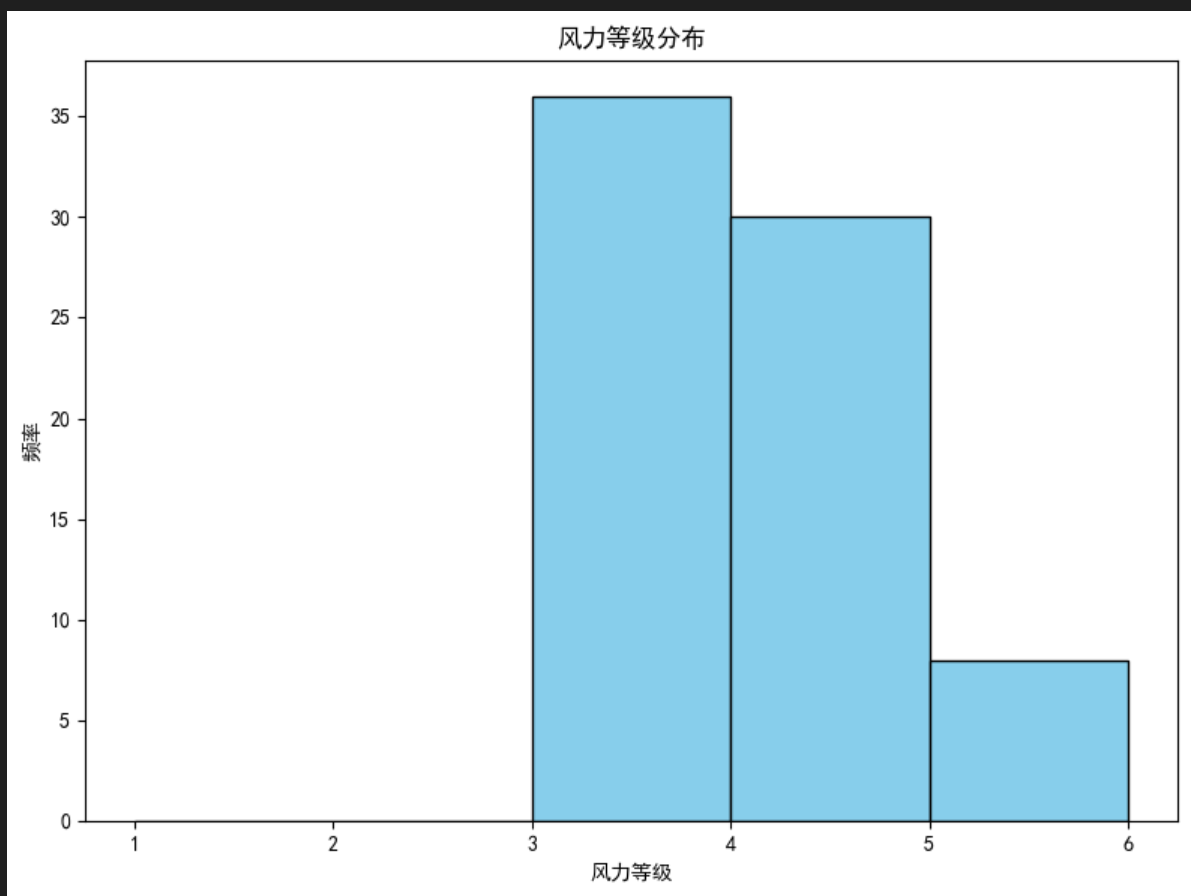
气温的变化能够反映季节的更替与天气的波动。通过绘制每日的最高气温与最低气温曲线，可以看出南京市在夏季的气温波动情况。图中红色曲线表示每日最高气温，蓝色曲线表示最低气温，从曲线的走势可以看出，气温在7月中旬出现了较为明显的上升，最高温度接近40°C，而8月末气温开始回落，预示着秋季的到来。





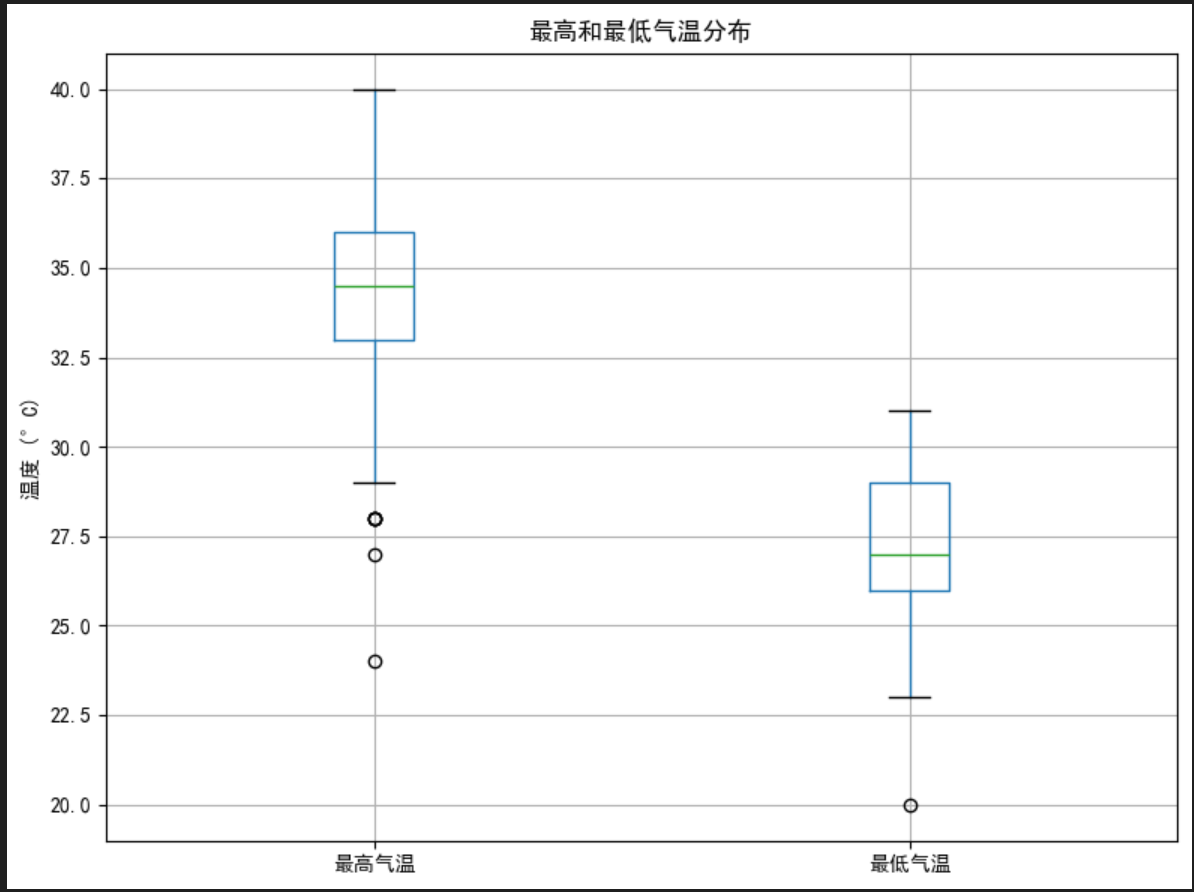
## 2. 风力等级分布

风力的变化可以直接影响到空气质量、出行安全等多个方面。通过对数据集中的风力等级进行直方图分析，结果显示，南京市的风力主要集中在3级和4级，属于较为温和的风速。6级风力出现的频率较低，表明南京市在夏季风力相对稳定。



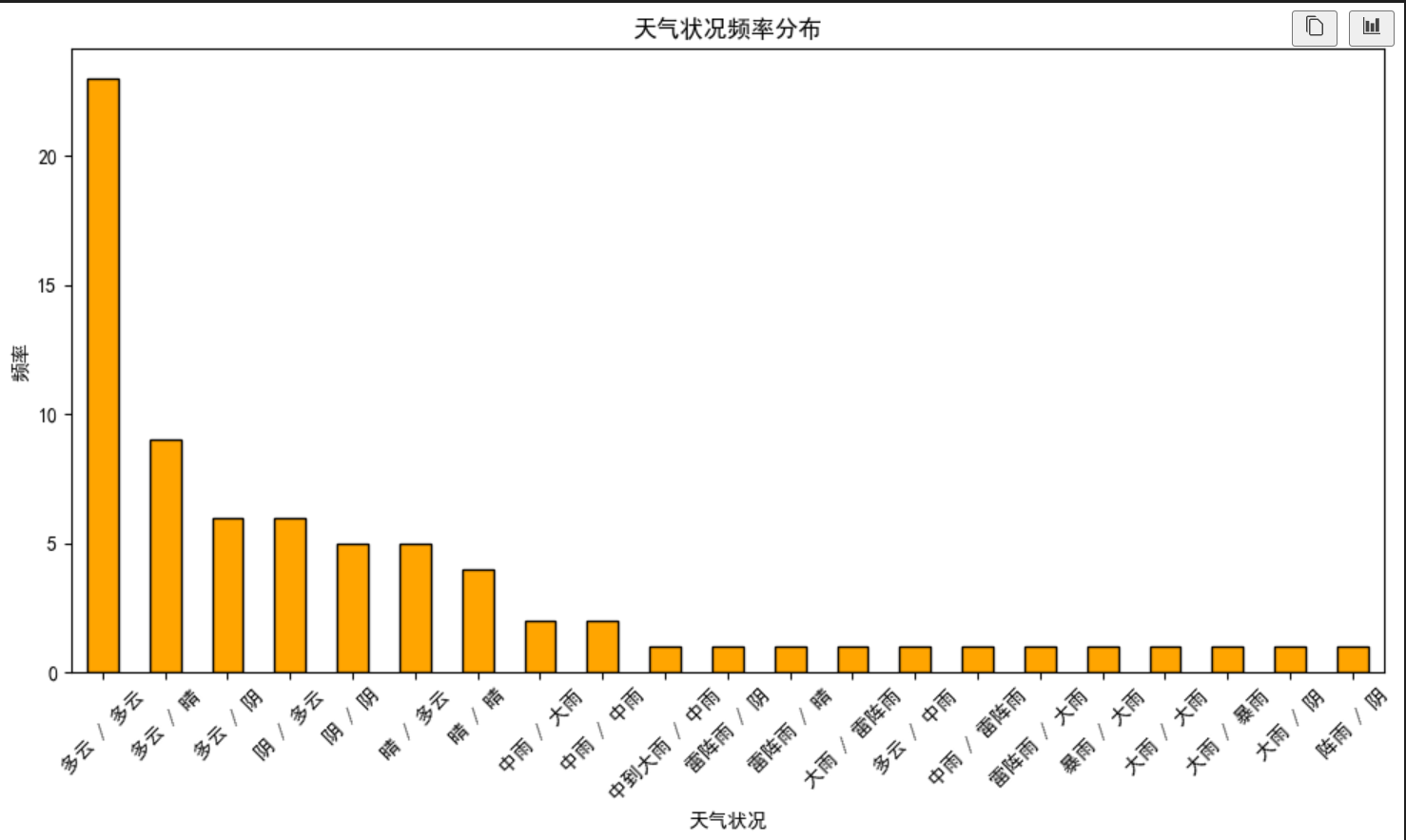
## 3. 气温分布箱线图

使用箱线图对最高气温和最低气温的分布进行分析，可以更直观地看到气温的上下四分位数、中位数以及异常值。最高气温的分布范围主要集中在33°C至37°C之间，最低气温则集中在26°C至30°C之间。少数异常值可能对应极端天气情况。



## 4. 天气状况频率分布

南京市的夏季天气变化较为频繁，通过分析天气状况的频率分布，可以发现多云和晴天的天气最为常见。雷阵雨天气出现的频率较低，说明在夏季南京的降水量相对适中。



# 模型预测

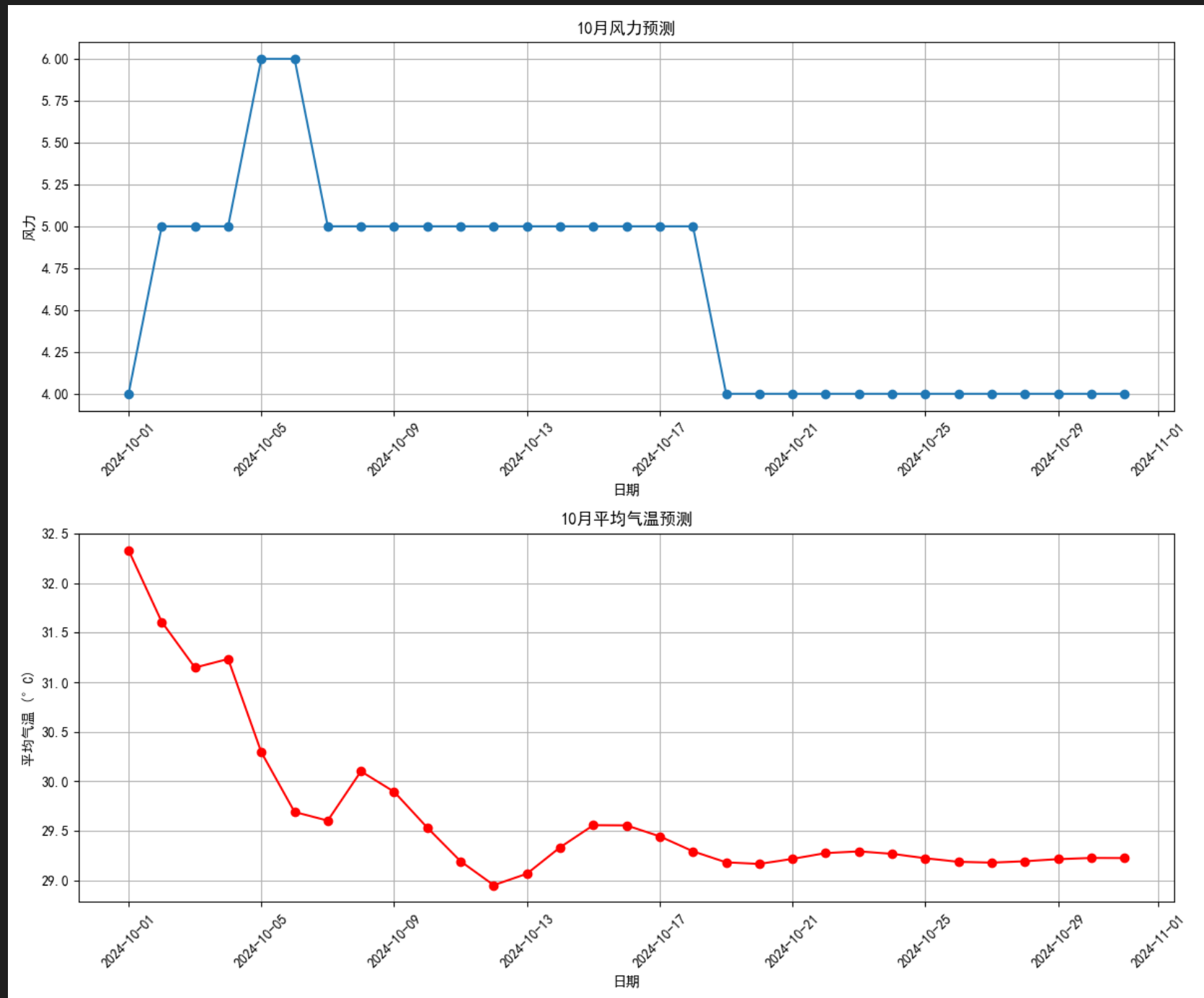
使用改进后的LSTM模型对2024年10月南京市的风力和平均气温进行了预测。模型预测的目标是每天的风力等级和平均气温

## 1. 风力预测

模型预测显示，10月的风力主要稳定在4级至5级之间。10月初风力较大，在10月4日达到6级，但随后逐渐回落至4级。预测结果表明，10月份南京市的风力相对温和，极端天气的可能性较小。

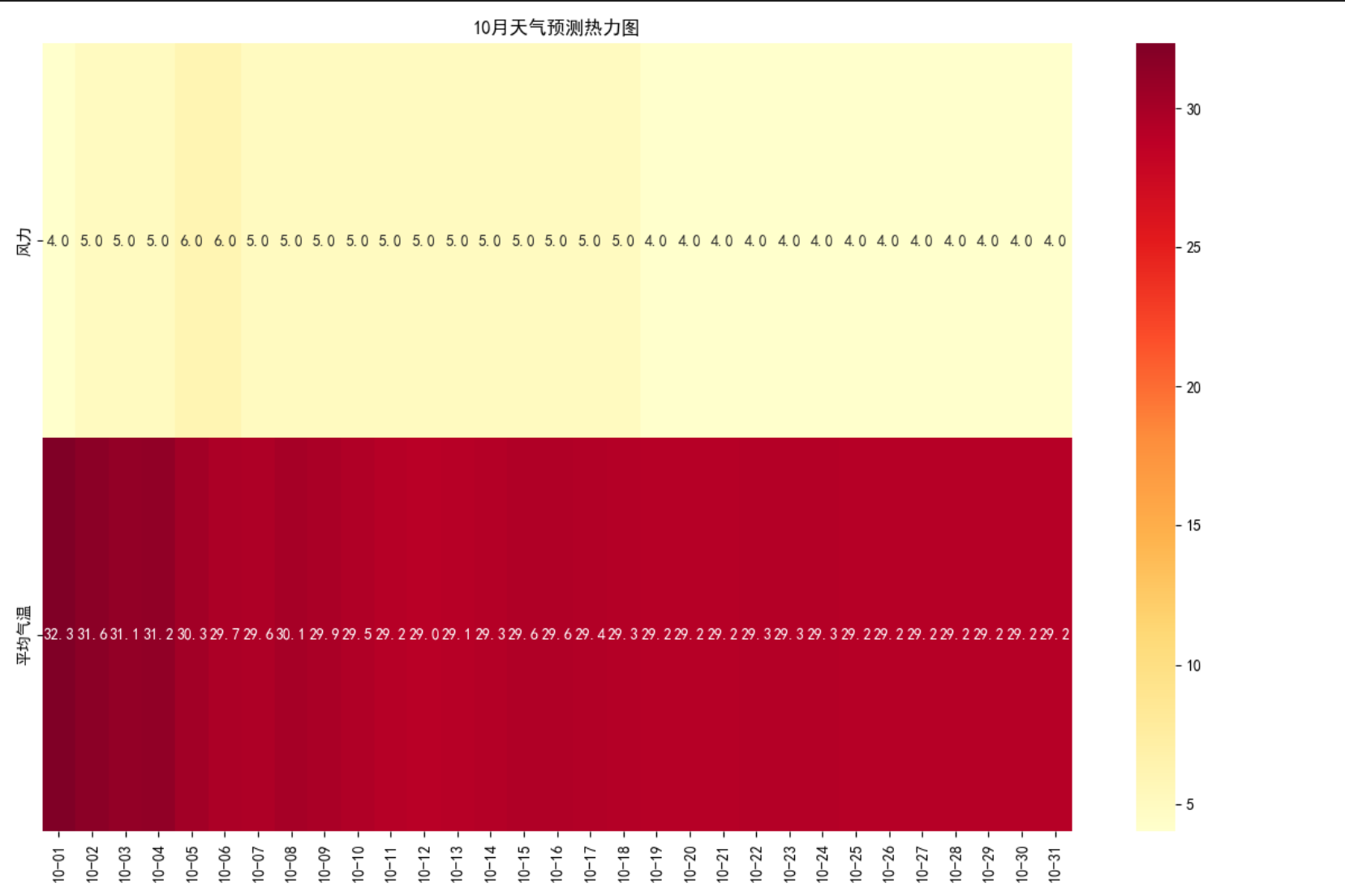
## 2. 平均气温预测

通过对未来31天的平均气温预测可以看出，南京市10月的平均气温呈现逐步下降的趋势。10月初的气温约为32°C，随后逐步降至29°C左右，表明秋季的降温趋势较为明显。这与南京市历年的气温变化规律基本一致。



## 3. 热力图分析

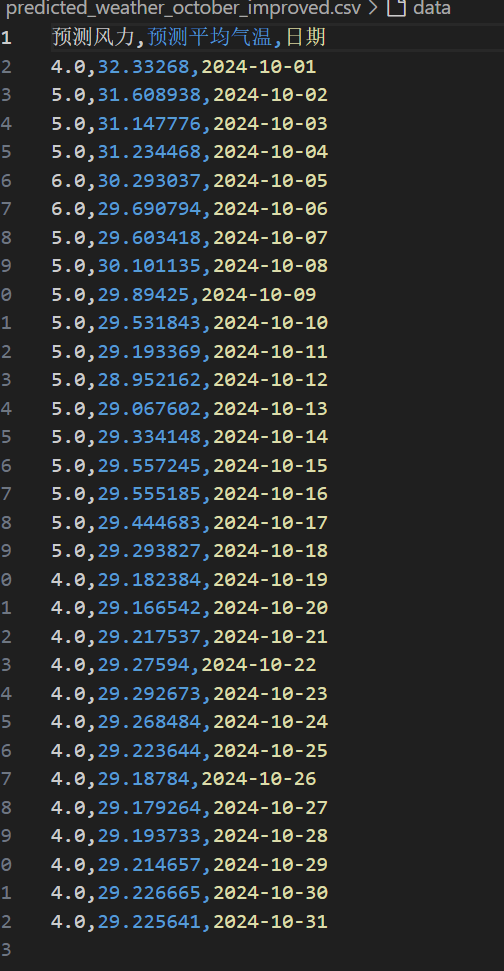
为了更好地展示风力与气温的关系，绘制了10月份的风力与平均气温的热力图。从热力图中可以看出，风力在10月初较强，而气温逐步下降，整体呈现出风力与气温独立变化的趋势。



## 1. 模型表现

改进的LSTM模型在风力和气温预测方面表现良好。通过均方误差（MSE）评估模型的表现，训练集上的风力MSE为0.1312，平均气温MSE为0.7087。较低的MSE值表明模型能够较好地拟合训练数据，并在预测未来天气时具有较好的准确性。

## 未来天气趋势

1. 

根据模型的预测结果，南京市在2024年10月的天气整体上较为平稳，风力等级在4级左右波动，平均气温呈现出从32°C下降到29°C的表明，基于LSTM的天气预测模型在处理时间序列数据时具有较强的鲁棒性和适应性。通过合理的特征选择与模型优化，能够为天气预测领域提供更为精准且高效的解决方案。

## 4. 模型优化与改进方向

尽管本次研究中所使用的LSTM模型已经取得了较好的预测效果，但仍有一些方面可以进一步优化和改进，以提升预测的准确性 目前的模型主要使用了气温、天气状况、风力等特征作为输入，而实际影响天气变化的因素远不止这些。湿度、降水量、大气压力等气象因素对气温和风力有着显著的影响。可以通过引入更多的气象特征来提升模型的学习能力和预测效果。本次研究的数据集涵盖了南京市三个月的天气数据，而实际的天气预测任务通常需要更长时间跨度的数据来建模。未来可以考虑使用更加丰富的数据集，例如历年同期的数据，从而增强模型对长期趋势的捕捉能力。当前的模型使用了两层LSTM和一个全连接层，适用于小规模的数据集。如果数据规模进一步扩大，可以增加模型的复杂度，如使用多层LSTM、双向LSTM等，提升模型的拟合能力。 在天气预测任务中，不同模型可以对不同的气象特征进行建模，因此可以考虑将多种机器学习模型进行融合。使用卷积神经网络（CNN）提取天气图片中的空间特征，结合LSTM进行时间序列建模，从而进一步提升预测效果。 天气预测中的多任务学习模型可以同时预测多个相关气象变量。

# 结论

LSTM模型通过其记忆机制，能够有效处理气象数据的时间序列特征，尤其适合预测具有时间依赖关系的变量。模型在本次研究中表现出良好的预测性能，能够有效捕捉未来气温和风力的变化趋势，预测结果与南京市实际 的气候变化规律基本吻合。 气象数据的特征维度直接影响到预测模型的表现。在本次研究中，主要使用了气温、风力、天气状况等基本气象指标，虽然取得了不错的效果，但如果能够引入更多与天气变化相关的特征，如湿度、气压等，模型的预测精度可能会进一步提升。

基于深度学习的天气预测技术具有广泛的应用前景。除了气象部门可以使用该技术提高天气预报的精度外，农业、交通、能源等领域也可以利用天气预测的结果优化资源配置和生产调度。农业可以根据未来的天气情况提前安排农作物的播种和收割时间，交通部门可以提前采取措施应对恶劣天气，能源企业则可以根据气温变化预测未来的电力需求，从而合理调度能源供给。

尽管本次研究在天气预测方面取得了较好的结果，但仍存在局限性。首先，数据集的规模相对较小，三个月的数据在时间跨度上较为有限；其次，模型使用的特征也较为单一，未来可以通过引入更多气象数据和更复杂的模型结构来进一步提高预测精度，达到更精准的预测效果。

附录：模型结构与训练日志以及代码

在模型训练过程中，记录了每个训练周期的损失变化情况

Epoch 1 Train Loss: 1.0108 Val Loss: 1.1256

Epoch 2 Train Loss: 1.0490 Val Loss: 1.1020

Epoch 3 Train Loss: 1.0687 Val Loss: 1.1151

Epoch 4 Train Loss: 1.0500 Val Loss: 1.1095

Epoch 5 Train Loss: 1.0547 Val Loss: 1.1038

Epoch 6 Train Loss: 0.9318 Val Loss: 1.0908

Epoch 7 Train Loss: 1.0628 Val Loss: 1.0913

Epoch 8 Train Loss: 1.0509 Val Loss: 1.0844

...

从训练日志可以看出，随着训练的进行，模型的训练损失和验证损失逐渐下降，表明模型在学习过程中逐渐拟合了训练数据，并能够较好地在验证集上进行泛化。

import pandas as pd

import matplotlib.pyplot as plt

# 设置中文字体显示

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

# 读取数据并清理'天气'列

file\_path = 'nanjing\_weather\_last\_3\_months.csv'

weather\_data = pd.read\_csv(file\_path)

weather\_data['天气'] = weather\_data['天气'].str.replace(r'\r\n\s+', ' / ', regex=True)

weather\_data['天气'] = weather\_data['天气'].str.replace(r' / /', ' / ', regex=True)

weather\_data.head()

weather\_data.describe()

# 气温变化图

plt.figure(figsize=(10, 6))

plt.plot(weather\_data['最高气温'], label='最高气温', color='red')

plt.plot(weather\_data['最低气温'], label='最低气温', color='blue')

plt.xlabel('') # 去掉X轴标签

plt.xticks([]) # 去掉X轴刻度

plt.ylabel('温度 (°C)')

plt.title('每日最高和最低气温变化')

plt.legend()

plt.tight\_layout()

plt.show()

# 每日气温图

plt.figure(figsize=(10, 6))

plt.plot(weather\_data['平均气温'], label='平均气温', color='green')

plt.xlabel('')

plt.xticks([])

plt.ylabel('平均温度 (°C)')

plt.title('平均每日气温变化')

plt.legend()

plt.tight\_layout()

plt.show()

# 风力分布直方图

plt.figure(figsize=(8, 6))

plt.hist(weather\_data['风力'], bins=range(1, weather\_data['风力'].max() + 1), color='skyblue', edgecolor='black')

plt.xlabel('风力等级')

plt.ylabel('频率')

plt.title('风力等级分布')

plt.tight\_layout()

plt.show()

# 最高和最低气温箱线图

plt.figure(figsize=(8, 6))

weather\_data[['最高气温', '最低气温']].boxplot()

plt.title('最高和最低气温分布')

plt.ylabel('温度 (°C)')

plt.tight\_layout()

plt.show()

# 天气状况频率图

weather\_counts = weather\_data['天气'].value\_counts()

plt.figure(figsize=(10, 6))

weather\_counts.plot(kind='bar', color='orange', edgecolor='black')

plt.title('天气状况频率分布')

plt.xlabel('天气状况')

plt.ylabel('频率')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

import torch

from torch.utils.data import DataLoader, TensorDataset

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

weather\_data = pd.read\_csv('nanjing\_weather\_last\_3\_months.csv')

weather\_data['天气'] = weather\_data['天气'].str.replace(r'\r\n\s+', ' / ', regex=True)

weather\_data['天气'] = weather\_data['天气'].str.replace(r' / /', ' / ', regex=True)

# 使用LabelEncoder将天气列和风向列进行编码

label\_encoder\_weather = LabelEncoder()

label\_encoder\_wind = LabelEncoder()

weather\_data['天气编码'] = label\_encoder\_weather.fit\_transform(weather\_data['天气'])

weather\_data['风向编码'] = label\_encoder\_wind.fit\_transform(weather\_data['风向'])

#StandardScaler进行标准化

scaler = StandardScaler()

scaled\_features = scaler.fit\_transform(weather\_data[['最高气温', '最低气温', '天气编码', '风向编码', '风力', '平均气温']])

scaled\_data = pd.DataFrame(scaled\_features, columns=['最高气温', '最低气温', '天气编码', '风向编码', '风力', '平均气温'])

def create\_sequences(data, seq\_length):

xs = []

ys = []

for i in range(len(data) - seq\_length):

x = data.iloc[i:i+seq\_length].values

y = data.iloc[i+seq\_length][['风力', '平均气温']].values

xs.append(x)

ys.append(y)

return np.array(xs), np.array(ys)

# 时间序列长度

seq\_length = 7

X, y = create\_sequences(scaled\_data, seq\_length)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

train\_dataset = TensorDataset(torch.tensor(X\_train, dtype=torch.float32), torch.tensor(y\_train, dtype=torch.float32))

val\_dataset = TensorDataset(torch.tensor(X\_val, dtype=torch.float32), torch.tensor(y\_val, dtype=torch.float32))

train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)

val\_loader = DataLoader(val\_dataset, batch\_size=32, shuffle=False)

class ImprovedLSTMModel(torch.nn.Module):

def \_\_init\_\_(self, input\_size, hidden\_layer\_size, output\_size, num\_layers=2, dropout=0.2):

super(ImprovedLSTMModel, self).\_\_init\_\_()

self.hidden\_layer\_size = hidden\_layer\_size

self.lstm = torch.nn.LSTM(input\_size, hidden\_layer\_size, num\_layers=num\_layers, batch\_first=True, dropout=dropout)

self.dropout = torch.nn.Dropout(dropout)

self.linear = torch.nn.Linear(hidden\_layer\_size, output\_size)

def forward(self, input\_seq):

lstm\_out, \_ = self.lstm(input\_seq)

lstm\_out = self.dropout(lstm\_out[:, -1, :])

predictions = self.linear(lstm\_out)

return predictions

input\_size = 6 # 特征数量

hidden\_layer\_size = 64

output\_size = 2 # 预测风力和平均气温

model = ImprovedLSTMModel(input\_size, hidden\_layer\_size, output\_size)

# 损失函数和优化器

loss\_function = torch.nn.MSELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

epochs = 100

best\_val\_loss = float('inf')

patience = 10

no\_improve = 0

for epoch in range(epochs):

model.train()

train\_loss = 0

for seq, labels in train\_loader:

optimizer.zero\_grad()

y\_pred = model(seq)

loss = loss\_function(y\_pred, labels)

loss.backward()

optimizer.step()

train\_loss += loss.item()

model.eval()

val\_loss = 0

with torch.no\_grad():

for seq, labels in val\_loader:

y\_pred = model(seq)

val\_loss += loss\_function(y\_pred, labels).item()

train\_loss /= len(train\_loader)

val\_loss /= len(val\_loader)

print(f'Epoch {epoch+1} Train Loss: {train\_loss:.4f} Val Loss: {val\_loss:.4f}')

if val\_loss < best\_val\_loss:

best\_val\_loss = val\_loss

torch.save(model.state\_dict(), 'best\_model.pth')

no\_improve = 0

else:

no\_improve += 1

if no\_improve >= patience:

print("Early stopping!")

break

model.load\_state\_dict(torch.load('best\_model.pth'))

# 预测

def predict\_future(model, last\_sequence, scaler, days\_to\_predict=31):

model.eval()

predictions = []

current\_seq = last\_sequence.copy()

for \_ in range(days\_to\_predict):

with torch.no\_grad():

input\_tensor = torch.tensor(current\_seq[-seq\_length:], dtype=torch.float32).unsqueeze(0)

pred = model(input\_tensor).squeeze().numpy()

predictions.append(pred)

# 更新序列

new\_day = np.zeros(input\_size)

new\_day[4:6] = pred # 只更新风力和平均气温

current\_seq = np.vstack([current\_seq, new\_day])

predictions = np.array(predictions)

# 只反归一化风力和平均气温

predictions\_unscaled = np.zeros\_like(predictions)

predictions\_unscaled[:, 0] = predictions[:, 0] \* scaler.scale\_[4] + scaler.mean\_[4] # 风力

predictions\_unscaled[:, 1] = predictions[:, 1] \* scaler.scale\_[5] + scaler.mean\_[5] # 平均气温

# 对风力进行四舍五入

predictions\_unscaled[:, 0] = np.round(predictions\_unscaled[:, 0])

return predictions\_unscaled

last\_seq = scaled\_data.iloc[-seq\_length:].values

future\_predictions = predict\_future(model, last\_seq, scaler)

future\_dates = pd.date\_range(start="2024-10-01", periods=31, freq='D')

predicted\_weather = pd.DataFrame(future\_predictions, columns=['预测风力', '预测平均气温'])

predicted\_weather['日期'] = future\_dates

output\_file = 'predicted\_weather\_october\_improved.csv'

predicted\_weather.to\_csv(output\_file, index=False)

print(f'改进后的预测结果已保存到 {output\_file}')

# 计算训练集的MSE

model.eval()

train\_predictions = []

train\_actual = []

with torch.no\_grad():

for seq, labels in train\_loader:

pred = model(seq)

train\_predictions.extend(pred.numpy())

train\_actual.extend(labels.numpy())

train\_predictions = np.array(train\_predictions)

train\_actual = np.array(train\_actual)

# 反归一化预测值和实际值

train\_predictions\_unscaled = np.zeros\_like(train\_predictions)

train\_actual\_unscaled = np.zeros\_like(train\_actual)

for i in range(2): # 0: 风力, 1: 平均气温

train\_predictions\_unscaled[:, i] = train\_predictions[:, i] \* scaler.scale\_[i+4] + scaler.mean\_[i+4]

train\_actual\_unscaled[:, i] = train\_actual[:, i] \* scaler.scale\_[i+4] + scaler.mean\_[i+4]

mse\_wind = mean\_squared\_error(train\_actual\_unscaled[:, 0], train\_predictions\_unscaled[:, 0])

mse\_temp = mean\_squared\_error(train\_actual\_unscaled[:, 1], train\_predictions\_unscaled[:, 1])

print(f"训练集上的风力MSE: {mse\_wind:.4f}")

print(f"训练集上的平均气温MSE: {mse\_temp:.4f}")

plt.figure(figsize=(12, 10))

plt.subplot(2, 1, 1)

plt.plot(future\_dates, predicted\_weather['预测风力'], marker='o')

plt.title('10月风力预测')

plt.xlabel('日期')

plt.ylabel('风力')

plt.xticks(rotation=45)

plt.grid(True)

plt.subplot(2, 1, 2)

plt.plot(future\_dates, predicted\_weather['预测平均气温'], marker='o', color='r')

plt.title('10月平均气温预测')

plt.xlabel('日期')

plt.ylabel('平均气温 (°C)')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.savefig('october\_weather\_prediction.png')

plt.close()

# 热力图可视化

plt.figure(figsize=(12, 8))

sns.heatmap(predicted\_weather[['预测风力', '预测平均气温']].T,

annot=True,

fmt='.1f',

cmap='YlOrRd',

yticklabels=['风力', '平均气温'],

xticklabels=future\_dates.strftime('%m-%d'))

plt.title('10月天气预测热力图')

plt.tight\_layout()

plt.savefig('october\_weather\_heatmap.png')

plt.close()