

Common Task Report

Name : Adwait Kulkarni

Overview:

Forecasting daily climate events for specific locations like Belvedere Castle in Central Park, New York; Midway International Airport in Chicago, Illinois; Bergstrom International Airport in Austin, Texas; and Miami, Florida, holds immense practical value in numerous aspects of daily life. These predictions influence decisions ranging from travel routes to everyday plans, impacting millions of individuals worldwide. Hence, the choice of models for such predictions is crucial, necessitating efficiency, accuracy, and sustainability with limited resources.

A key factor in the accuracy of these models lies in their architectural design and the quality of the training data. Weather-related forecasting tasks, benefiting from extensive historical datasets, offer an ideal testing ground for developing and refining predictive models. For this project, the focus is on predicting daily maximum temperatures for the mentioned locations, abbreviated as NYC, MDW, AUS, and MIA.

Various factors influence daily maximum temperatures, including historical trends, wind patterns, precipitation, humidity, and atmospheric pressure. Even the previous day's temperature can serve as a predictor in many cases. The project involves collecting historical data from diverse sources, processing it, analyzing different potential predictors, and evaluating a range of models for temperature prediction. Ultimately, the chosen model's predictions will be utilized for manual and automated trading activities on the Kalshi platform.

Tasks to be done by each week:

1. Week 1: Collect historical data for each city from 5 different sources while performing manual trading on Kalshi.
2. Week 2: Train an ML model on the collected dataset, evaluate the accuracy. Predict the temperature for the next days and use this to perform trading on Kalshi
3. Week 3: Set up an API to automatically place trades on your behalf using your model predictions.

Week1

Data Collection & Manual Prediction :

Open Meteo Data also provides an API that facilitates access to a wealth of meteorological information. Known for its simplicity and ease of use, the Open Meteo Data API offers straightforward endpoints for retrieving weather data, making it suitable for both novice and experienced developers. The API's responsiveness and flexibility made it an attractive choice for me. Their API did have a daily limit of 500 calls. I also tried the API functionalities of other sources such as Worldweather online, CLI-DAP of purdue, visual crossing(not included in the data folder in submission), meteostat but the data obtained from NOAA and open meteo was much clean and easy to access. I decided to use openmeteo as the source of data for model training.

Handshake

VMock

BU Login

Job Listings BU

Mail

Drive

Gradescope

Courses

CS 581 | Piazza

CAS CS 542

ARAMARK

Problems - LeetCode

Mary - Adwait

Alumni Mentorship...

Logins & Passwords...

CAS CS 585

Adobe Acrobat

Kalshi

Kalshi

Demo

Markets

Ideas

Search markets

Continue sign up to start trading with real money →

Watchlist

Watch

Portfolio

Financials

Highest temperature in Austin

--C ↑49

71° or below

Highest temperature in Chicago

--C ↑49

34° or below

Highest temperature in Miami

--C ↓25

83° or below

Highest temperature in NYC

--C ↑40

43° or below

Portfolio

\$3,435.74

↑\$10.74 (0.31%) today

\$3,435.74 in funds

47.50

of 12.5K volume

Unlock



.625% dollar rebate

Position

Resting

Closed

Select closed markets

Market	Avg	Qty	Total return
 Highest temperature in Chicago On Feb 27, 2024			ROI \$188.00
73° to 74°	Yes 6C	200	\$188.00 (+1566.67%)
 Highest temperature in Chicago On Mar 19, 2024			ROI \$478.88
50° or below	No 1C	142	\$140.58 (+9900.00%)
51° to 52°	No 1C	131	\$129.69 (+9900.00%)
53° to 54°	Yes 1C	870	-\$8.70 (-100.00%)
55° to 56°	No 1C	180	\$178.20 (Help)

Watchlist

Watch

Portfolio

Financials

Highest temperature in Austin

--C

79° or below

Highest temperature in Chicago

--C

38° or below

Highest temperature in Miami

--C

73° or below

Highest temperature in NYC

--C

46° or below

Highest temperature in NYC

On Feb 27, 2024


ROI -\$0.83

58° to 59°

Yes 1C

83

-\$0.83 (-100.00%)

 Highest temperature in NYC
On Feb 28, 2024


ROI -\$1,000.00

59° to 60°

Yes 91C

1100

-\$1,000.00 (-100.00%)

 Highest temperature in NYC
On Mar 20, 2024

ROI \$537.56

50° or below

No 50C

1041

\$524.42 (+101.52%)

51° to 52°

No 1C

4

\$3.96 (+9900.00%)

53° to 54°

Yes 2C

82

-\$1.64 (-100.00%)

55° to 56°

No 1C

4

\$3.96 (+9900.00%)

57° to 58°

No 1C

7

-\$0.07 (-100.00%)

59° or above

No 1C

7

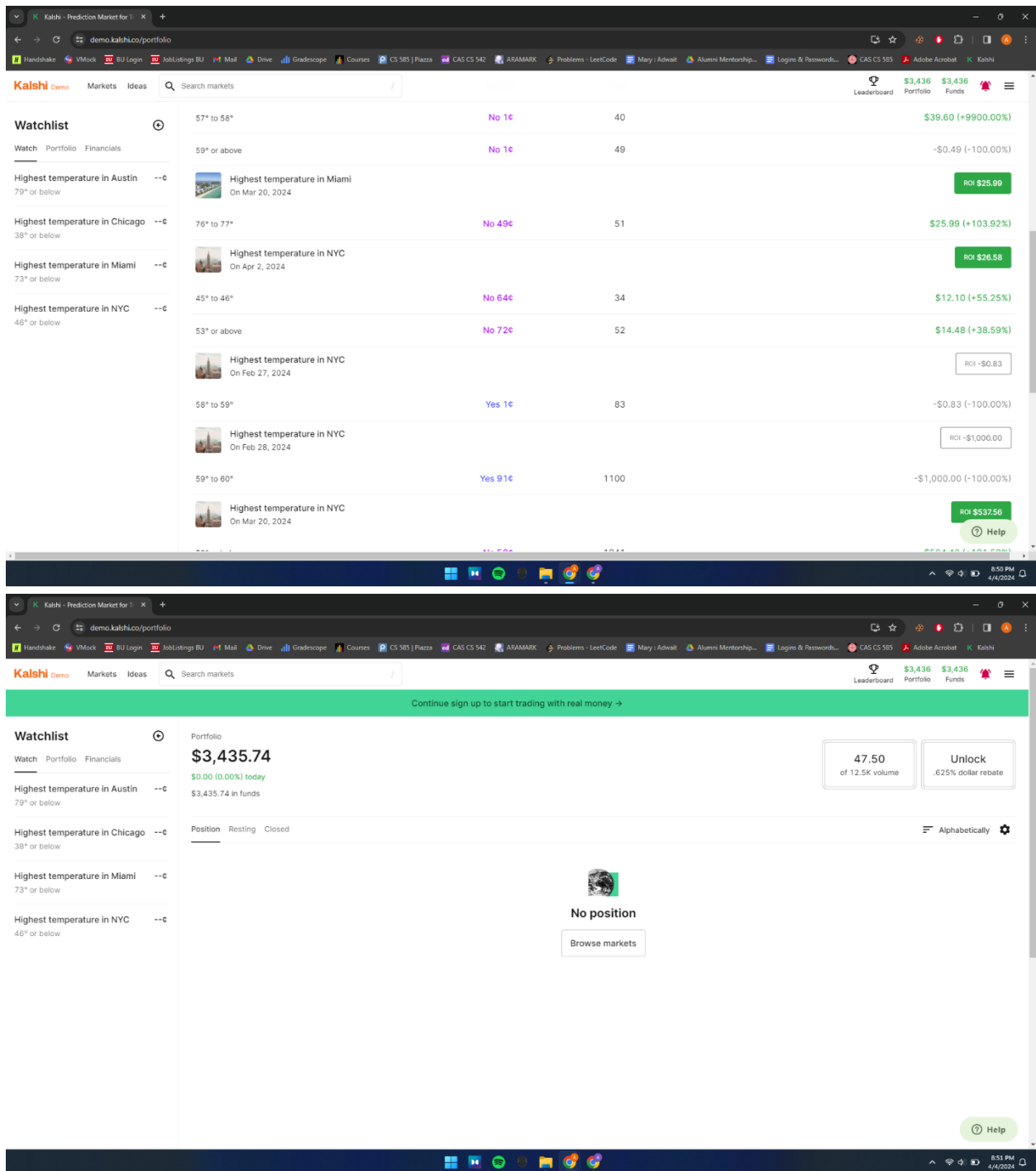
\$6.93 (+9900.00%)

Company

Social

Product

Help



Week2

Feature Selection and Model Training:

To enhance the accuracy of this temperature prediction model I strategically selected essential meteorological parameters identified within the dataset that was collected. Critical features such as relative humidity, precipitation, rain, snowfall, wind speed, and wind gusts were selected due to major influence on temperature at a place/environment.

Following feature identification, preprocessing steps to refine the data for model training. This involved handling missing values and scaling the features to ensure uniformity and optimal performance. This resulted in providing the model with clean and standardized input, thereby facilitating accurate temperature predictions.

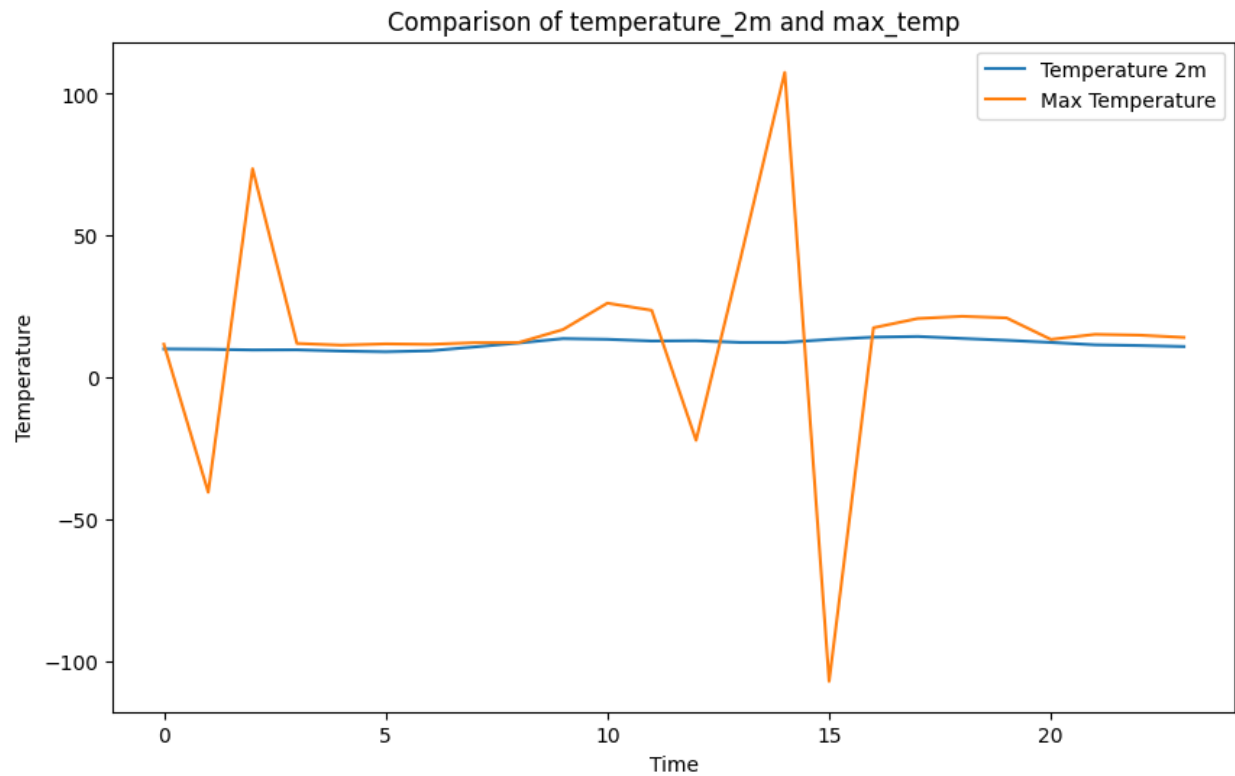
I trained 3 models a Linear Regression , SVR and LSTM for this to get a relative comparison of which might perform better. After evaluating the three models – linear regression, SVR, and LSTM – for temperature prediction, it was evident that linear regression performed the poorest, followed by SVR. LSTM gave the best results among the 3 models and was selected as the primary model for temperature forecasting.

Basic LSTM cell consisting of forget, input, output gates and providing candidate memory, memory cell (long term memory) and output(hidden layer or short-term memory). The LSTM model that was used could be summarized as follows:

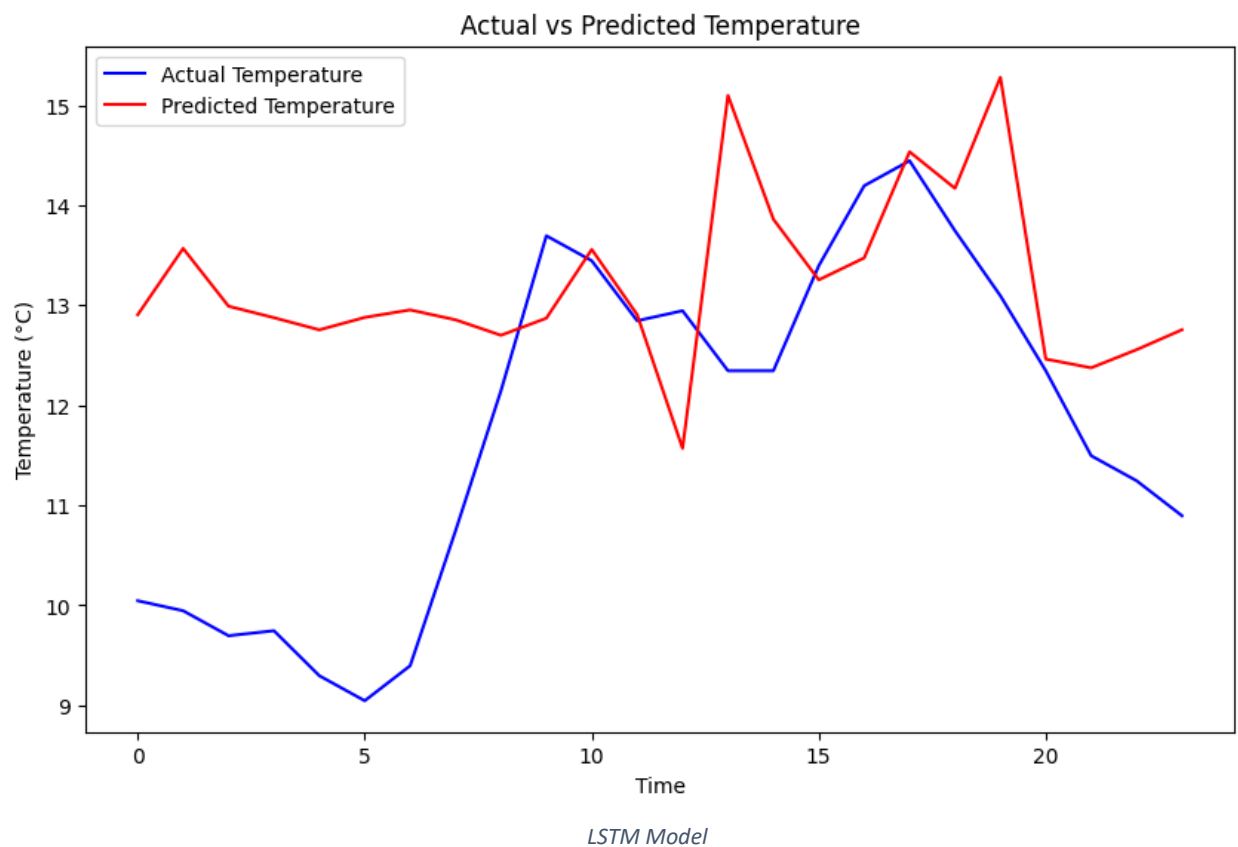
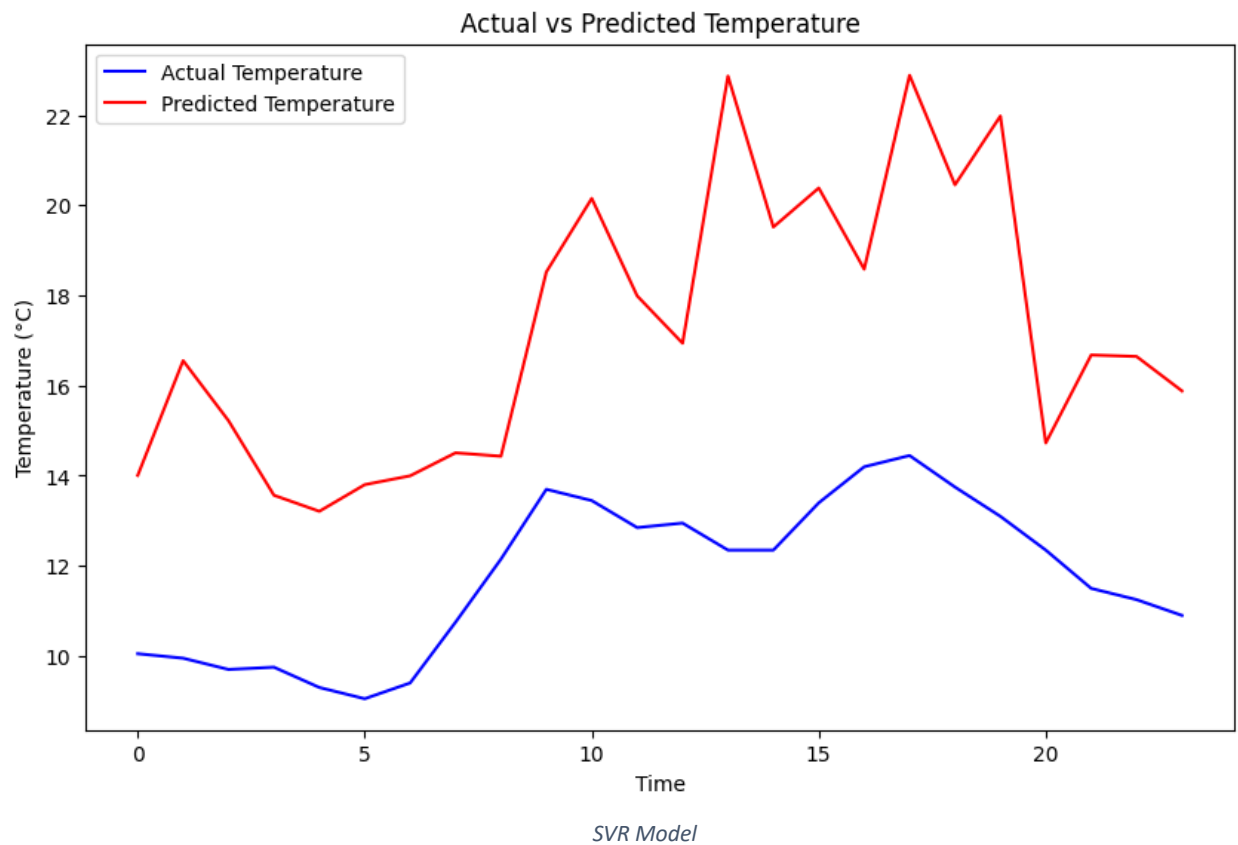
- Data Preparation: The meteorological data, including features like relative humidity, precipitation, rain, snowfall, wind speed, and wind gusts, is loaded into pandas DataFrame objects 'X' (features) and 'y' (target variable - temperature).
- Data Scaling and Splitting: The features ('X') and target variable ('y') are scaled using MinMaxScaler to normalize them between 0 and 1. The dataset is then split into training and testing sets using the train_test_split function from sklearn.model_selection.
- Reshaping for LSTM: As LSTM requires input data in a specific shape (samples, time steps, features), the training and testing feature sets ('X_train' and 'X_test') are reshaped accordingly.
- Model Definition: A Sequential model is initialized, and an LSTM layer with 50 units is added as the first layer. The 'input_shape' parameter is set based on the shape of the training data. A Dense layer with one unit (for regression) is added as the output layer.
- Model Compilation: The model is compiled using the Adam optimizer and mean squared error (MSE) as the loss function.
- Model Training: The model is trained on the training data ('X_train' and 'y_train') for 30 epochs with a batch size of 32.
- Model Evaluation: After training, the model's performance is evaluated using the testing data ('X_test' and 'y_test'), and the mean squared error (MSE) is computed as the evaluation metric.

Thereafter overfitting was observed in the model resulting in poor performance and regularization and dropout methods were used to mitigate this issue.

The following photos show the performance(Actual vs Predicted Temperature Values) of the models Linear Regression, SVR and LSTM in the same order.



Linear Regression Model



Week3

KalshiAPI Trading:

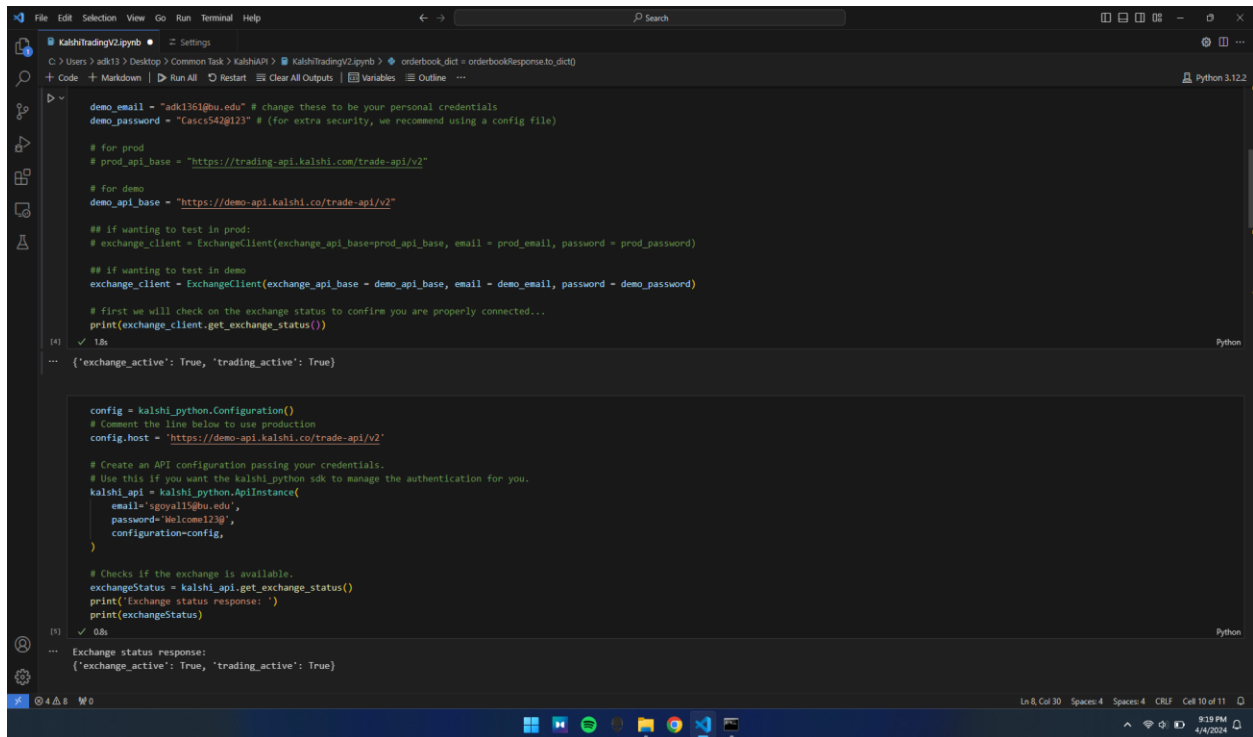
The Python script(in KalshiAPI folder) was used for interfacing with the Kalshi trading API to execute orders based on predicted maximum temperatures across various cities. Its functionality encompasses several key steps. Firstly, the script initiates by configuring credentials and connecting to the Kalshi API, enabling access to trading functionalities. Thereafter, it a .CSV file is given as input, primarily focusing on the predicted maximum temperatures data present for different cities in the .CSV file.

Moving forward, the script systematically identifies the appropriate market ticker for each city based on its predicted maximum temperature. It thereafter retrieves the corresponding order book from the Kalshi API, providing insights into available trading offers.

Upon scrutinizing the order book, the script discerns the best offer price ('yes' offers) and promptly submits buy orders for a predefined quantity. To optimize the likelihood of execution, the order price is strategically set slightly lower than the best offer price. This meticulous approach ensures prudent trading decisions aligned with weather forecasts.

Throughout the execution, the script logs details of each order submitted, including essential information such as the market, order type, quantity, and price. In instances where a matching market ticker cannot be found for a city, the script transparently communicates this occurrence,

ensuring comprehensive oversight of the trading process.



The screenshot shows a Jupyter Notebook titled 'KalshiTradingV2.ipynb' in the Visual Studio Code editor. The notebook is running on Python 3.12.2. The code is organized into two cells. The first cell contains configuration and client initialization code, and the second cell contains the configuration and status check code.

```
demo_email = "adk1351@bu.edu" # change these to be your personal credentials
demo_password = "Casc542@123" # (for extra security, we recommend using a config file)

# for prod
prod_api_base = "https://trading-api.kalshi.com/trade-api/v2"

# for demo
demo_api_base = "https://demo-api.kalshi.co/trade-api/v2"

## if wanting to test in prod:
# exchange_client = ExchangeClient(exchange_api_base=prod_api_base, email = prod_email, password = prod_password)

## if wanting to test in demo
exchange_client = ExchangeClient(exchange_api_base=demo_api_base, email = demo_email, password = demo_password)

# first we will check on the exchange status to confirm you are properly connected...
print(exchange_client.get_exchange_status())
```

Output for the first cell:

```
{
  "exchange_active": True,
  "trading_active": True
}
```

```
config = kalshi_python.Configuration()
# Comment the line below to use production
config.host = "https://demo-api.kalshi.co/trade-api/v2"

# Create an API configuration passing your credentials.
# Use this if you want the kalshi_python sdk to manage the authentication for you.
kalshi_api = kalshi_python.ApiInstance(
    email="sgoyal15@bu.edu",
    password="Wolcom@123",
    configuration=config,
)

# Checks if the exchange is available.
exchangeStatus = kalshi_api.get_exchange_status()
print('Exchange status response: ')
print(exchangeStatus)
```

Output for the second cell:

```
Exchange status response:
{'exchange_active': True, 'trading_active': True}
```

The bottom status bar shows the cursor is at Line 8, Column 30, with 4 spaces and 4 characters. The system clock indicates 9:19 PM on 4/2/2024.

```
File Edit Selection View Go Run Terminal Help
KalshiTradingV2.py • Settings
C:\Users\adk13\Desktop> Common Task > KalshiAPI > KalshiTradingV2.py • orderbook_dict = orderbookResponse.to_dict()
+ Code + Markdown + Run All + Restart + Clear All Outputs + Variables + Outline ... Python 3.12.2

# Parse the event response to find matching market ticker
# This code block assumes a function to parse and match the market ticker based on predicted temperature
matching_market_ticker = determine_market_ticker_from_response(eventResponse, predicted_max_temp)

# Now you have the matching market ticker, you can retrieve the order book
marketTicker = f'{eventTicker}-{matching_market_ticker}'
orderbookResponse = kalshi_api.get_market_orderbook(marketTicker) # Replace with actual API call

print(f'\nOrderbook for market: {marketTicker}')
# Assuming pprint is imported or defined
pprint(orderbookResponse)
# print(dir(orderbookResponse))

112 ✓ 20s Python

...

Orderbook for market: HIGHHAUS-24APR04-B86.5
('orderbook': {'no': [], 'yes': [[99, 1]]})

Orderbook for market: HIGHCHI-24APR04-T46
('orderbook': {'no': [], 'yes': [[99, 1]]})

Orderbook for market: HIGHMIA-24APR04-B86.5
('orderbook': {'no': [], 'yes': [[99, 1]]})

Orderbook for market: HIGHNY-24APR04-T54
('orderbook': {'no': [], 'yes': [[99, 1]]})

yes_offers = orderbookResponse.orderbook.yes
orderbook_dict = orderbookResponse.to_dict()
yes_offers = orderbook_dict['orderbook']['yes']

113 ✓ 00s Python

orderbook_dict = orderbookResponse.to_dict()
yes_offers = orderbook_dict['orderbook']['yes']
for index, row in df.iterrows():
    city = row['location']
    predicted_max_temp = row['predicted_max_temp']

    ))
    print(f'\nOrder submitted for {city}: ')
    pprint(orderResponse)
else:
    print(f'No "yes" offers available for {city}, no orders will be placed.')
else:
    print(f'No matching market ticker found for {city} based on the predicted temperature.')

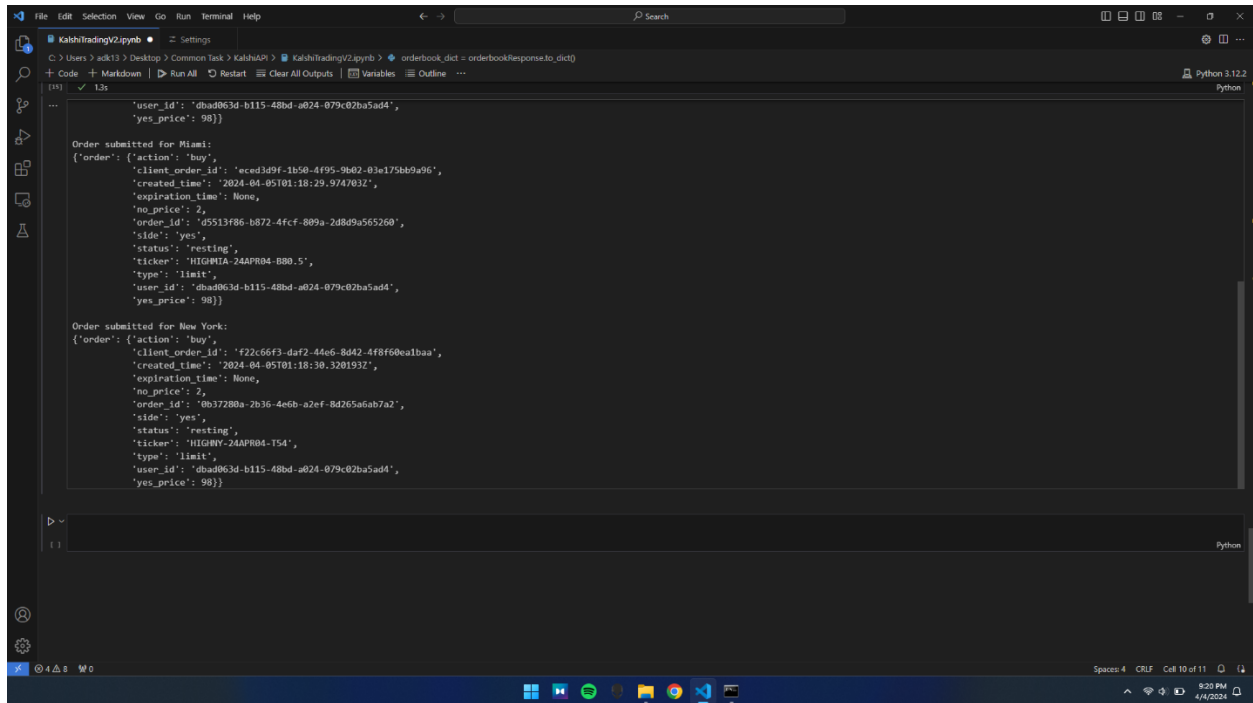
115 ✓ 13s Python

...

Order submitted for Austin:
{'order': {'action': 'buy',
  'client_order_id': '730d99faf-ed89-480c-8905-1036e5924951',
  'created_time': '2024-04-05T01:18:29.307094Z',
  'expiration_time': None,
  'no_price': 2,
  'order_id': '09fbfa10-0fad-4a88-baa3-cf83c3be74e6',
  'side': 'yes',
  'status': 'resting',
  'ticker': 'HIGHHAUS-24APR04-B86.5',
  'type': 'limit',
  'user_id': 'dbad863d-b115-48bd-a024-079c02ba5ad4',
  'yes_price': 98}}

Order submitted for Chicago:
{'order': {'action': 'buy',
  'client_order_id': '84d6e1e5-0050-49f0-a098-a6425436e437',
  'created_time': '2024-04-05T01:18:29.633352Z',
  'expiration_time': None,
  'no_price': 2,
  'order_id': 'b60cc942-f173-417d-abce-0b0dae092beb',
  'side': 'yes',
  'status': 'resting',
  'ticker': 'HIGHCHI-24APR04-T46',
  'type': 'limit',
  'user_id': 'dbad863d-b115-48bd-a024-079c02ba5ad4',
  'yes_price': 98}}

Order submitted for Miami:
```



```
...  
'user_id': 'dbad063d-b115-48bd-a024-079c02ba5ad4',  
'yes_price': 98})  
  
Order submitted for Miami:  
{  
  'order': {  
    'action': 'buy',  
    'client_order_id': 'eced3d9f-1b50-4f95-9b02-03e175bb9a96',  
    'created_time': '2024-04-05T01:18:29.974703Z',  
    'expiration_time': None,  
    'no_price': 2,  
    'order_id': 'd5513f86-b872-4fcf-809a-2d8d9a565260',  
    'side': 'yes',  
    'status': 'resting',  
    'ticker': 'HIGWHY-2AAPR04-B80.5',  
    'type': 'limit',  
    'user_id': 'dbad063d-b115-48bd-a024-079c02ba5ad4',  
    'yes_price': 98}}  
  
Order submitted for New York:  
{  
  'order': {  
    'action': 'buy',  
    'client_order_id': 'f22c66f3-daf2-44e6-8042-4f8f60ca1baa',  
    'created_time': '2024-04-05T01:18:30.320193Z',  
    'expiration_time': None,  
    'no_price': 2,  
    'order_id': '0b37200a-2b36-4e0b-a2ef-8d265a6ab7a2',  
    'side': 'yes',  
    'status': 'resting',  
    'ticker': 'HIGWHY-2AAPR04-T54',  
    'type': 'limit',  
    'user_id': 'dbad063d-b115-48bd-a024-079c02ba5ad4',  
    'yes_price': 98}}  
}
```

Conclusion:

This project aimed to predict daily maximum temperatures for key locations such as Central Park in New York, Chicago, Austin, and Miami, using historical weather data and machine learning models. I began by gathering data from trusted sources like NOAA and Open Meteo Data in Week 1, ensuring the availability of reliable datasets.

Weeks 2 and 3 focused on refining the models and integrating them with the Kalshi trading API for automated trading. After rigorous evaluation, I found that LSTM outperformed Linear Regression and SVR models in temperature prediction accuracy. However, further improvements in the LSTM model can be done to obtain near perfect predictions.

This project also tackles a practical problem in ML application, walking us through the steps of model training. From gathering data to making trades, it provides insights into real-world implementation and the challenges faced. For instance, dealing with limitations in various data collection APIs, such as restrictions on calls or duration. These factors were carefully considered during data processing. The use of the Kalshi trading API also demonstrates how temperature forecasts can be applied in real-life scenarios.