Consider the problem at hand, the data characteristics, and the available expertise when choosing between these methods. It may also be beneficial to experiment with both approaches and compare their performance on your specific problem.

Can you write me out one in Python with an SPY.csv as input

Certainly! Below, I'll provide you with a Python script that uses an LSTM model paired with ARIMA for time series forecasting. We'll use the SPY.csv file as input, assuming it contains historical stock price data for SPY (S&P 500 E-mini futures).

First, make sure you have the required libraries installed:

#pip install pandas numpy matplotlib scikit-learn statsmodels tensorflow

Here's the Python script:

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

from statsmodels.tsa.arima.model import ARIMA

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from tensorflow.keras.optimizers import Adam

*# Load the data*

df = pd.read\_csv('SPY.csv', index\_col='Date', parse\_dates=True)

df = df['Close'].dropna()

*# Normalize the data*

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(df.values.reshape(-1, 1))

*# Create training and testing datasets*

def create\_dataset(dataset, time\_step=1):

dataX, dataY = [], []

for i in range(len(dataset) - time\_step - 1):

a = dataset[i:(i + time\_step), 0]

dataX.append(a)

dataY.append(dataset[i + time\_step, 0])

return np.array(dataX), np.array(dataY)

time\_step = 100

X\_train, y\_train = create\_dataset(scaled\_data, time\_step)

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

*# Train LSTM model*

model = Sequential()

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(time\_step, 1)))

model.add(LSTM(units=50))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer=Adam(lr=0.001))

model.fit(X\_train, y\_train, epochs=100, batch\_size=32)

*# Predict using LSTM*

lstm\_pred = model.predict(X\_train)

lstm\_pred = scaler.inverse\_transform(lstm\_pred)

*# Fit ARIMA model*

arima\_model = ARIMA(df, order=(5,1,2))

arima\_result = arima\_model.fit()

arima\_pred = arima\_result.forecast(steps=len(y\_train))[0]

arima\_pred = scaler.inverse\_transform(arima\_pred.reshape(-1, 1))

*# Combine LSTM and ARIMA predictions*

combined\_pred = (lstm\_pred + arima\_pred) / 2

*# Evaluate the model*

rmse = np.sqrt(mean\_squared\_error(y\_train, combined\_pred))

print(f'RMSE: {rmse}')

*# Plot the results*

import matplotlib.pyplot as plt

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

plt.plot(df.index[-len(y\_train):], y\_train, label='Actual')

plt.plot(df.index[-len(y\_train):], combined\_pred, label='Predicted')

plt.title('SPY Stock Price Prediction')

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()  
  
#ADD Straddle long at assumed Strike Price  
  
import numpy as np

import pandas as pd

from datetime import datetime

# Load historical option data (this should be a DataFrame with columns: 'Strike', 'Expiry', 'CallPrice', 'PutPrice')

# For example, you might load this from a CSV file:

# df\_options = pd.read\_csv('historical\_option\_prices.csv')

# Define the straddle long strategy

def straddle\_long\_strategy(df\_options, current\_price, strike\_price, expiry\_date):

# Filter options with the same expiry and strike price

options\_at\_price = df\_options[(df\_options['Expiry'] == expiry\_date) & (df\_options['Strike'] == strike\_price)]

# Calculate the intrinsic value of the call and put options

call\_intrinsic\_value = np.maximum(current\_price - strike\_price, 0)

put\_intrinsic\_value = np.maximum(strike\_price - current\_price, 0)

# Calculate the premium paid for the call and put options

call\_premium = options\_at\_price['CallPrice'].values[0]

put\_premium = options\_at\_price['PutPrice'].values[0]

# Calculate the net cost of the straddle

straddle\_cost = call\_premium + put\_premium

# Check if the strategy is profitable

if call\_intrinsic\_value + put\_intrinsic\_value > straddle\_cost:

return True, straddle\_cost, call\_intrinsic\_value, put\_intrinsic\_value

else:

return False, straddle\_cost, call\_intrinsic\_value, put\_intrinsic\_value

# Example usage

current\_stock\_price = 100 # Current stock price

strike\_price = 100 # Strike price for the options

expiry\_date = '2023-12-17' # Expiry date for the options

# Assuming you have a DataFrame with historical option prices

# df\_options = pd.read\_csv('historical\_option\_prices.csv')

# Check if the straddle long strategy is profitable

is\_profitable, cost, call\_value, put\_value = straddle\_long\_strategy(df\_options, current\_stock\_price, strike\_price, expiry\_date)

if is\_profitable:

print(f"Straddle long strategy is profitable. Cost: {cost}, Call intrinsic value: {call\_value}, Put intrinsic value: {put\_value}")

# Here you would add the code to execute the trade

else:

print("Straddle long strategy is not profitable.")

# Note: This script does not include the actual trade execution, risk management, or real-time data fetching.