Homework 2

Group Members: Yash Shah, Zheng Wan, Linhui Sang

Import Libraries

In [ ]:
```
import numpy as np
import yfinance as yf
import zipfile
import pandas as pd
import requests
from io import BytesIO
from scipy.interpolate import interp1d
from scipy import integrate
import scipy.stats as ss
from scipy.stats import norm
import matplotlib.pyplot as plt
from google.colab import drive
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
```

Download Data

In [ ]:
```
import yfinance as yf

# Define the ticker symbol for the VIX (CBOE Volatility Index)
ticker_symbol = '^VIX'

# Define the start and end dates
start_date = '2016-01-01'
end_date = '2016-04-30'

# Download the data from Yahoo Finance
vix_data = yf.download(ticker_symbol, start=start_date, end=end_date)

# Display the first few rows of the downloaded data
print(vix_data.head())
```

```
[*********************100%%**********************]  1 of 1 completed
                Open       High        Low      Close  Adj Close  Volume
Date
2016-01-04  22.480000  23.360001  20.670000  20.700001  20.700001       0
2016-01-05  20.750000  21.059999  19.250000  19.340000  19.340000       0
2016-01-06  21.670000  21.860001  19.799999  20.590000  20.590000       0
2016-01-07  23.219999  25.860001  22.400000  24.990000  24.990000       0
2016-01-08  22.959999  27.080000  22.480000  27.010000  27.010000       0
```

In [ ]:
```
import zipfile
import pandas as pd
import requests
from io import BytesIO

# URL of the ZIP file
zip_file_url = 'https://github.com/larrysangfake/financial_econometrics/blob/main/SPX_2016_options

# Download the ZIP file
response = requests.get(zip_file_url)
zip_data = BytesIO(response.content)

# Path to the CSV file inside the ZIP file
csv_file_name = 'SPX_2016_options.csv'
```

```
# Open the ZIP file
with zipfile.ZipFile(zip_data, 'r') as zip_ref:
    # Extract the CSV file
    zip_ref.extract(csv_file_name, path='.')

# Read the CSV file into a DataFrame
df = pd.read_csv(csv_file_name, header=None)

# Now you can work with the DataFrame (df)
print(df.head())
```

```
        0         1       2       3   4   5     6       7       8     9   \
0  108105  20160104  736333  736344  11   1  1000  1007.4  1010.8  2000
1  108105  20160104  736333  736344  11   1  1025   982.3   985.8     0
2  108105  20160104  736333  736344  11   1  1050   957.3   960.9     0
3  108105  20160104  736333  736344  11   1  1075   932.3   935.9     0
4  108105  20160104  736333  736344  11   1  1100   907.4   911.1     0

      10  11       12        13  14  15        16
0  42291 NaN  2012.66 -0.015304   1   1  0.007121
1      0 NaN  2012.66 -0.015304   1   1  0.007121
2      0 NaN  2012.66 -0.015304   1   1  0.007121
3      0 NaN  2012.66 -0.015304   1   1  0.007121
4     20 NaN  2012.66 -0.015304   1   1  0.007121
```

In [ ]:
```
df.columns = [
    'ID', 'Date', 'Julian Date', 'Julian Maturity Date', 'Time Difference in Days', 'Call/Put',
    'Strike Price', 'bid', 'ask', 'Unnamed:9', 'Unnamed:10', 'volatility',
    'stock price', 'Unnamed:13', 'Unnamed: 14', 'Unnamed: 15', 'interest rate'
]

# Print the DataFrame to check the new column names
print(df.head())
```

```
       ID      Date  Julian Date  Julian Maturity Date  \
0  108105  20160104       736333                736344
1  108105  20160104       736333                736344
2  108105  20160104       736333                736344
3  108105  20160104       736333                736344
4  108105  20160104       736333                736344

   Time Difference in Days  Call/Put  Strike Price     bid     ask  Unnamed:9  \
0                       11         1          1000  1007.4  1010.8       2000
1                       11         1          1025   982.3   985.8          0
2                       11         1          1050   957.3   960.9          0
3                       11         1          1075   932.3   935.9          0
4                       11         1          1100   907.4   911.1          0

   Unnamed:10  volatility  stock price  Unnamed:13  Unnamed: 14  Unnamed: 15  \
0       42291         NaN      2012.66   -0.015304            1            1
1           0         NaN      2012.66   -0.015304            1            1
2           0         NaN      2012.66   -0.015304            1            1
3           0         NaN      2012.66   -0.015304            1            1
4          20         NaN      2012.66   -0.015304            1            1

   interest rate
0       0.007121
1       0.007121
2       0.007121
3       0.007121
4       0.007121
```

1. Data Cleaning

In [ ]:
```
cleaned_data = df.dropna(subset=['volatility'])
print(cleaned_data.head())
```

```
         ID      Date  Julian Date  Julian Maturity Date  \
128  108105  20160104       736333                736344
129  108105  20160104       736333                736344
130  108105  20160104       736333                736344
131  108105  20160104       736333                736344
132  108105  20160104       736333                736344

     Time Difference in Days  Call/Put  Strike Price    bid    ask  Unnamed:9  \
128                       11         1          1875  135.3  138.2        500
129                       11         1          1880  130.1  133.9         12
130                       11         1          1885  125.4  129.1          0
131                       11         1          1890  120.7  124.4          5
132                       11         1          1895  116.5  119.3          0

     Unnamed:10  volatility  stock price  Unnamed:13  Unnamed: 14  \
128         311    0.149708      2012.66   -0.015304            1
129          29    0.184996      2012.66   -0.015304            1
130           7    0.196155      2012.66   -0.015304            1
131          30    0.203980      2012.66   -0.015304            1
132           5    0.209911      2012.66   -0.015304            1

     Unnamed: 15  interest rate
128            1       0.007121
129            1       0.007121
130            1       0.007121
131            1       0.007121
132            1       0.007121
```

2. Create a new column

In [ ]:
```
cleaned_data['Average Price'] = (cleaned_data['bid'] + cleaned_data['ask']) / 2

# To see the updated DataFrame with the new 'Average Price' column
print(cleaned_data.head())
```

```
         ID      Date  Julian Date  Julian Maturity Date  \
128  108105  20160104       736333                736344
129  108105  20160104       736333                736344
130  108105  20160104       736333                736344
131  108105  20160104       736333                736344
132  108105  20160104       736333                736344

     Time Difference in Days  Call/Put  Strike Price    bid    ask  Unnamed:9  \
128                       11         1          1875  135.3  138.2        500
129                       11         1          1880  130.1  133.9         12
130                       11         1          1885  125.4  129.1          0
131                       11         1          1890  120.7  124.4          5
132                       11         1          1895  116.5  119.3          0

     Unnamed:10  volatility  stock price  Unnamed:13  Unnamed: 14  \
128         311    0.149708      2012.66   -0.015304            1
129          29    0.184996      2012.66   -0.015304            1
130           7    0.196155      2012.66   -0.015304            1
131          30    0.203980      2012.66   -0.015304            1
132           5    0.209911      2012.66   -0.015304            1

     Unnamed: 15  interest rate  Average Price
128            1       0.007121         136.75
129            1       0.007121         132.00
130            1       0.007121         127.25
131            1       0.007121         122.55
132            1       0.007121         117.90
<ipython-input-6-d0d5f6261de8>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['Average Price'] = (cleaned_data['bid'] + cleaned_data['ask']) / 2
```

3. Filter the columns

In [ ]:
```
filtered_df = cleaned_data[(cleaned_data['Average Price'] > 0.05)]

# Check the filtered data
print(filtered_df.head())
```

```
         ID      Date  Julian Date  Julian Maturity Date  \
128  108105  20160104       736333                736344
129  108105  20160104       736333                736344
130  108105  20160104       736333                736344
131  108105  20160104       736333                736344
132  108105  20160104       736333                736344

     Time Difference in Days  Call/Put  Strike Price    bid    ask  Unnamed:9  \
128                       11         1          1875  135.3  138.2        500
129                       11         1          1880  130.1  133.9         12
130                       11         1          1885  125.4  129.1          0
131                       11         1          1890  120.7  124.4          5
132                       11         1          1895  116.5  119.3          0

     Unnamed:10  volatility  stock price  Unnamed:13  Unnamed: 14  \
128         311    0.149708      2012.66   -0.015304            1
129          29    0.184996      2012.66   -0.015304            1
130           7    0.196155      2012.66   -0.015304            1
131          30    0.203980      2012.66   -0.015304            1
132           5    0.209911      2012.66   -0.015304            1

     Unnamed: 15  interest rate  Average Price
128            1       0.007121         136.75
129            1       0.007121         132.00
130            1       0.007121         127.25
131            1       0.007121         122.55
132            1       0.007121         117.90
```

4. Out the Money

In [ ]:
```
# Filter Out of the Money Call Options (S < K)
otm_calls = filtered_df[(filtered_df['Call/Put'] == 1) & (filtered_df['stock price'] < filtered_df

# Filter Out of the Money Put Options (S > K)
otm_puts = filtered_df[(filtered_df['Call/Put'] == -1) & (filtered_df['stock price'] > filtered_df

# Combine OTM calls and puts into a single DataFrame
otm_options = pd.concat([otm_calls, otm_puts])

# Check the resulting DataFrame
print(otm_options.head())
```

```
        ID      Date  Julian Date  Julian Maturity Date  \
156  108105  20160104       736333                736344
157  108105  20160104       736333                736344
158  108105  20160104       736333                736344
159  108105  20160104       736333                736344
160  108105  20160104       736333                736344

     Time Difference in Days  Call/Put  Strike Price   bid   ask  Unnamed:9  \
156                       11         1          2015  23.4  25.5       1852
157                       11         1          2020  20.6  22.4       1105
158                       11         1          2025  18.1  20.0       4957
159                       11         1          2030  15.9  17.7        187
160                       11         1          2035  13.8  15.6        180

     Unnamed:10  volatility  stock price  Unnamed:13  Unnamed: 14  \
156        4389    0.195975      2012.66    -0.015304            1
157        2124    0.190988      2012.66    -0.015304            1
158       29268    0.188562      2012.66    -0.015304            1
159        1741    0.186419      2012.66    -0.015304            1
160         628    0.184163      2012.66    -0.015304            1

     Unnamed: 15  interest rate  Average Price
156            1       0.007121          24.45
157            1       0.007121          21.50
158            1       0.007121          19.05
159            1       0.007121          16.80
160            1       0.007121          14.70
```

5.The implied volatility

In [ ]:
```
#Black and Scholes
def BlackScholes(CallPutFlag,S,X,v,r,T):

    d1 = (np.log(S/X)+(r+v*v/2.)*T)/(v*np.sqrt(T))

    d2 = d1-v*np.sqrt(T)
    if CallPutFlag=="C":
        P = S*norm.cdf(d1) - X*np.exp(-r*T)*norm.cdf(d2)

    else:
        P = -S*norm.cdf(-d1) + X*np.exp(-r*T)*norm.cdf(-d2)
    return P
```

In [ ]:
```
def ivol(K,IV,Kall):
    if Kall>=K[len(K)-1]:
        Kall=K[len(K)-1]
    if Kall<=K[0]:
        Kall=K[0]
    funy = interp1d(K,IV, kind='cubic', fill_value="extrapolate")
    y=funy(Kall)

    if (np.sum(y<0)>0):
        if Kall>=K[len(K)-1]:
            Kall=K[len(K)-1]
        if Kall<=K[0]:
            Kall=K[0]
        funy = interp1d(K,IV, kind='linear', fill_value="extrapolate")
        y=funy(Kall)
    return(y)
```

In [ ]:
```
def RiskNeutralVolatilitySkewKurt_JVKR(Kvector, IVvector, S0, T, r):
    kmin=.1*S0;
    kmax=3.5*S0;

    def V1(K):
        V1=2*(1-np.log(K/S0))*BlackScholes("C",S0, K, ivol(Kvector,IVvector,K), r, T)/np.power(K,
        return(V1)
    def V2(K):
        V2=2*(1+np.log(S0/K))*BlackScholes("P",S0, K, ivol(Kvector,IVvector,K), r, T)/np.power(K,
        return(V2)
```

```
    def W1(K):
        W1=(6*np.log(K/S0)-3*np.power(np.log(K/S0),2))*BlackScholes("C",S0, K, ivol(Kvector,IVvec
        return(W1)

    def W2(K):
        W2=(6*np.log(S0/K)+3*np.power(np.log(S0/K),2))*BlackScholes("P",S0, K, ivol(Kvector,IVvec
        return(W2)

    def X1(K):
        X1=(((12*np.power(np.log(K/S0),2) - 4*np.power(np.log(K/S0),3)))*(BlackScholes("C",S0, K,
        return(X1)

    def X2(K):
        X2=(((12*np.power(np.log(S0/K),2) + 4*np.power(np.log(S0/K),3)))*(BlackScholes("P",S0, K,
        return(X2)
    V=integrate.quad(V1,S0,kmax)[0]+integrate.quad(V2,kmin,S0)[0]
    W=integrate.quad(W1,S0,kmax)[0]-integrate.quad(W2,kmin,S0)[0]
    X=integrate.quad(X1,S0,kmax)[0]+integrate.quad(X2,kmin,S0)[0]
    mu=np.exp(r*T)-1-np.exp(r*T)*V/2-np.exp(r*T)*W/6-np.exp(r*T)*X/24;
    #print(V,W,X,mu)

    vol=np.sqrt(1/T * V);
    skew=( np.exp(r*T)*W - 3*mu*np.exp(r*T)*V + 2*np.power(mu,3)) / np.power(np.exp(r*T)*V - np.
    kurt=( np.exp(r*T)*X - 4*mu*np.exp(r*T)*W + 6*np.exp(r*T)*np.power(mu,2)*V - 3*np.power(mu,

    return([vol,skew,kurt]);
```

In [ ]:
```
# Convert dates and calculate time to maturity in years
otm_options['Date'] = pd.to_datetime(otm_options['Date'], format='%Y%m%d')
otm_options['Time to Maturity'] = otm_options['Time Difference in Days'] / 365.25

# Define a function to apply risk-neutral volatility, skewness, and kurtosis calculation
def apply_risk_neutral_vol_skew_kurt(group):
    # Extract group-level constants
    S0 = group['stock price'].iloc[0]
    r = group['interest rate'].iloc[0]
    T = group['Time to Maturity'].iloc[0]

    # Calculate implied volatility, skewness, and kurtosis
    if len(group) < 2:
        return pd.Series([np.nan] * len(group), index=group.index)  # Return NaNs for groups with
    else:
        vol, skew, kurt = RiskNeutralVolatilitySkewKurt_JVKR(group['Strike Price'].values, group['
        return pd.Series([vol] * len(group), index=group.index)  # Repeat the calculated volatili

# Apply the function to each group and create a new DataFrame with the results
volatility_results = otm_options.groupby(['Date', 'Julian Maturity Date']).apply(apply_risk_neutra

# Merge the results back to the original DataFrame
otm_options['Implied Volatility'] = volatility_results.reset_index(level=[0, 1], drop=True)

# Display the results
print(otm_options[['Date', 'Julian Maturity Date', 'Implied Volatility']].head())
```

```
          Date  Julian Maturity Date  Implied Volatility
156 2016-01-04                736344            0.195965
157 2016-01-04                736344            0.195965
158 2016-01-04                736344            0.195965
159 2016-01-04                736344            0.195965
160 2016-01-04                736344            0.195965
```

In [ ]:
```
# Merge the results back to the original DataFrame
otm_options = pd.merge(otm_options, volatility_results, on=['Date', 'Julian Maturity Date'], how='

# Display the results
print(otm_options[['Date', 'Julian Maturity Date', 'Implied Volatility']].head())
```

In [ ]:
```
import pandas as pd
import numpy as np
from scipy.interpolate import interp1d

# Assuming otm_options is predefined
```

```
unique_dates = otm_options['Julian Date'].unique()
unique_maturities = otm_options['Julian Maturity Date'].unique()

results = []

for date in unique_dates:
    for maturity in unique_maturities:
        subset = otm_options[(otm_options['Julian Date'] == date) & (otm_options['Julian Maturity

        if len(subset) > 1:
            K = subset['Strike Price'].values
            IV = subset['volatility'].values  # Ensure this column is correctly filled

            # Safely apply interpolation
            try:
                # Assuming ivol is already defined and handles extrapolation or errors internally
                interp_func = interp1d(K, IV, kind='linear', fill_value='extrapolate', bounds_erro
                subset = subset.copy()  # Work on a copy to avoid SettingWithCopyWarning
                subset['implied_volatility'] = subset['Strike Price'].apply(lambda x: interp_func
                results.append(subset)
            except Exception as e:
                print(f"Error in interpolation for date {date} and maturity {maturity}: {e}")

# Combine all results into a single DataFrame
result_df = pd.concat(results, ignore_index=True)

print(result_df.head())
```

```
       ID      Date  Julian Date  Julian Maturity Date  \
0  108105  20160104       736333                736344
1  108105  20160104       736333                736344
2  108105  20160104       736333                736344
3  108105  20160104       736333                736344
4  108105  20160104       736333                736344

   Time Difference in Days  Call/Put  Strike Price   bid   ask  Unnamed:9  \
0                       11         1          2015  23.4  25.5       1852
1                       11         1          2020  20.6  22.4       1105
2                       11         1          2025  18.1  20.0       4957
3                       11         1          2030  15.9  17.7        187
4                       11         1          2035  13.8  15.6        180

   Unnamed:10  volatility  stock price  Unnamed:13  Unnamed: 14  Unnamed: 15  \
0        4389    0.195975      2012.66   -0.015304            1            1
1        2124    0.190988      2012.66   -0.015304            1            1
2       29268    0.188562      2012.66   -0.015304            1            1
3        1741    0.186419      2012.66   -0.015304            1            1
4         628    0.184163      2012.66   -0.015304            1            1

   interest rate  Average Price  implied_volatility
0       0.007121          24.45            0.195975
1       0.007121          21.50            0.190988
2       0.007121          19.05            0.188562
3       0.007121          16.80            0.186419
4       0.007121          14.70            0.184163
```

6. 30-day volatility

In [ ]:

```
# Check how often we have exactly 30 days to maturity
days_30 = otm_options[otm_options['Time Difference in Days'] == 30]

# See how many such entries exist
print(f"Entries with exactly 30 days to maturity: {days_30.shape[0]}")

# Optional: View some of these entries to verify
print(days_30.head())
```

```
Entries with exactly 30 days to maturity: 3088
           ID       Date  Julian Date  Julian Maturity Date  \
18638  108105 2016-01-06       736335                736365
18639  108105 2016-01-06       736335                736365
18640  108105 2016-01-06       736335                736365
18641  108105 2016-01-06       736335                736365
18642  108105 2016-01-06       736335                736365

       Time Difference in Days  Call/Put  Strike Price   bid   ask  Unnamed:9  \
18638                       30         1          1995  37.4  38.1         48
18639                       30         1          2000  34.6  35.3         72
18640                       30         1          2005  32.0  32.6         35
18641                       30         1          2010  29.4  30.0         48
18642                       30         1          2015  26.9  27.5       1064

       Unnamed:10  volatility  stock price  Unnamed:13  Unnamed: 14  \
18638          62    0.184500      1990.26   -0.013115            1
18639        1606    0.182070      1990.26   -0.013115            1
18640          49    0.179860      1990.26   -0.013115            1
18641          51    0.177407      1990.26   -0.013115            1
18642        1044    0.174921      1990.26   -0.013115            1

       Unnamed: 15  interest rate  Average Price  Time to Maturity  \
18638            1       0.006932          37.75          0.082136
18639            1       0.006932          34.95          0.082136
18640            1       0.006932          32.30          0.082136
18641            1       0.006932          29.70          0.082136
18642            1       0.006932          27.20          0.082136

       Implied Volatility
18638             0.18447
18639             0.18447
18640             0.18447
18641             0.18447
18642             0.18447
```

In [ ]:
```
print(otm_options.groupby('Date')['Time Difference in Days'].agg([min, max]))
```

```
            min   max
Date
2016-01-04    4  1082
2016-01-05    3  1081
2016-01-06    2  1080
2016-01-07    8  1079
2016-01-08    7  1078
...         ...   ...
2016-04-25    2   970
2016-04-26    3   969
2016-04-27    2   968
2016-04-28    6   967
2016-04-29    5   966

[82 rows x 2 columns]
```

In [ ]:
```
# Check the distribution of 'Time Difference in Days' across all data
print(otm_options['Time Difference in Days'].describe())
```

```
count    272078.000000
mean        137.898290
std         203.938153
min           2.000000
25%          32.000000
50%          60.000000
75%         130.000000
max        1082.000000
Name: Time Difference in Days, dtype: float64
```

In [ ]:
```
import pandas as pd
from scipy.interpolate import interp1d
import numpy as np

# Sample data setup
data = {
```

```
    'Date': ['2024-01-01', '2024-01-01', '2024-01-01', '2024-01-02', '2024-01-02', '2024-01-02'],
    'Time Difference in Days': [25, 28, 35, 27, 30, 33],
    'Implied Volatility': [0.20, 0.19, 0.18, 0.22, 0.21, 0.20]
}
sample_df = pd.DataFrame(data)
sample_df['Date'] = pd.to_datetime(sample_df['Date'])

def interpolate_iv_at_30(group):
    days = group['Time Difference in Days'].values
    ivs = group['Implied Volatility'].values

    if 30 in days:
        return ivs[np.where(days == 30)[0][0]]
    else:
        interp_function = interp1d(days, ivs, kind='linear', bounds_error=False, fill_value="extr
        interpolated_value = float(interp_function(30))

        if min(days) < 30 < max(days):
            return interpolated_value
        else:
            closest_idx = (np.abs(days - 30)).argmin()
            return ivs[closest_idx]

# Apply the function and store the result
result_sample = sample_df.groupby('Date').apply(interpolate_iv_at_30)

# Merge results back into the original DataFrame
sample_df['IV at 30 Days'] = sample_df['Date'].map(result_series)

# Print the result
print(sample_df[['Date', 'IV at 30 Days']].drop_duplicates())
```

```
        Date  IV at 30 Days
0 2024-01-01       0.187143
3 2024-01-02       0.210000
```

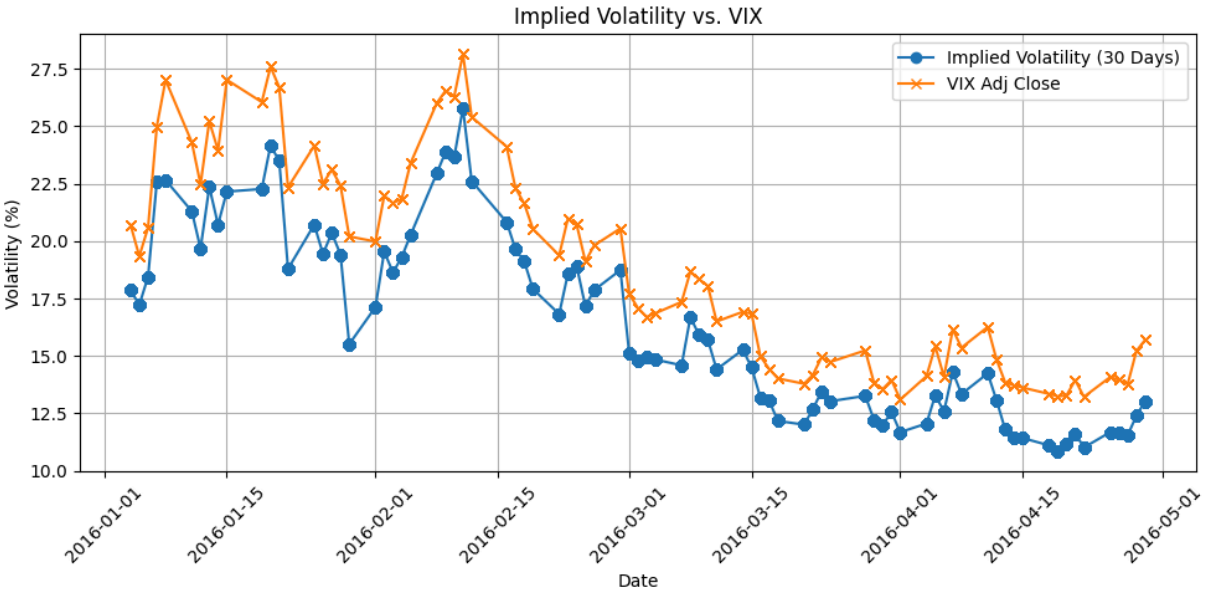
In [ ]:
```
result = otm_options.groupby('Date').apply(interpolate_iv_at_30)

# Merge results back into the original DataFrame
otm_options['IV at 30 Days'] = otm_options['Date'].map(result)

# Print the result
print(otm_options[['Date', 'IV at 30 Days']].drop_duplicates())
```

```
             Date  IV at 30 Days
156    2016-01-04       0.178551
7325   2016-01-05       0.172282
14661  2016-01-06       0.184470
21993  2016-01-07       0.225709
29751  2016-01-08       0.226559
...           ...            ...
582448 2016-04-25       0.116834
589881 2016-04-26       0.116640
597494 2016-04-27       0.115469
605102 2016-04-28       0.124032
612548 2016-04-29       0.130302

[82 rows x 2 columns]
```

7. Plot

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd

otm_options['IV at 30 Days'] *= 100  # Convert to percentage

# Merge VIX data with your options data
combined_df = pd.merge(otm_options, vix_data, on='Date', how='inner')

# Plotting both implied volatility and VIX
plt.figure(figsize=(10, 5))
plt.plot(combined_df['Date'], combined_df['IV at 30 Days'], label='Implied Volatility (30 Days)',
```

```
plt.plot(combined_df['Date'], combined_df['Adj Close'], label='VIX Adj Close', marker='x')
plt.title('Implied Volatility vs. VIX')
plt.xlabel('Date')
plt.ylabel('Volatility (%)')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Calculate the correlation
correlation = combined_df['IV at 30 Days'].corr(combined_df['Close'])
print("Correlation between IV at 30 Days and VIX Close:", correlation)
```




```
Correlation between IV at 30 Days and VIX Close: 0.9925817967753203
```

This high correlation indicates that we almost copied VIX from our calculation, but it requires some further transformation to make a more perfect match.

Paper Code

In [ ]:
```
# options_df =pd.read_csv('./options_combined.csv')
```

In [ ]:
```
# pd.set_option('display.max_columns', None)
# options_df.head()
```

Out[ ]:

	date	exdate	cp_flag	optiondate	expirationdate	week_day	strike_price	maturity	moneyness	index_l
0	20160104	20160115	1	736333	736344	2	1875	11	0.931603	201
1	20160104	20160115	1	736333	736344	2	1880	11	0.934087	201
2	20160104	20160115	1	736333	736344	2	1890	11	0.939056	201
3	20160104	20160115	1	736333	736344	2	1900	11	0.944024	201
4	20160104	20160115	1	736333	736344	2	1910	11	0.948993	201

In [ ]:
```
# options_df_filtered = options_df[(options_df['volume'] != 0) | (options_df['best_bid'] >= 0.125)
```

In [ ]:
```
# options_df_filtered.head()
```

Out[ ]:

	date	exdate	cp_flag	optiondate	expirationdate	week_day	strike_price	maturity	moneyness	index_l
0	20160104	20160115	1	736333	736344	2	1875	11	0.931603	201
1	20160104	20160115	1	736333	736344	2	1880	11	0.934087	201
2	20160104	20160115	1	736333	736344	2	1890	11	0.939056	201
3	20160104	20160115	1	736333	736344	2	1900	11	0.944024	201
4	20160104	20160115	1	736333	736344	2	1910	11	0.948993	201

In [ ]:
```
# options_df_filtered[options_df_filtered['callprice'] - options_df_filtered['putprice'] == option
```

Out[ ]:

	date	exdate	cp_flag	optiondate	expirationdate	week_day	strike_price	maturity	moneyness	index_level	inc

In [ ]:
```
# m = options_df_filtered['maturity'].unique()
# m.sort()
# m
```

Out[ ]:
```
array([  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])
```

In [ ]:
```
# options_temp = options_df_filtered.copy()

# options_temp.loc[(options_temp['maturity'] >= 20) & (options_temp['maturity'] <= 60), 'maturity_
# options_temp.loc[(options_temp['maturity'] > 60) & (options_temp['maturity'] <= 240), 'maturity_
```

In [ ]:
```
# options_temp['maturity_category'].value_counts()
```

Out[ ]:
```
maturity_category
short    862610
long     411371
Name: count, dtype: int64
```

In [ ]:
```
# moneyness_bins = [0.8, 0.9, 0.97, 1.03, 1.1, 1.6]
# moneyness_labels = ['[0.8, 0.9)', '[0.9, 0.97)', '[0.97, 1.03)', '[1.03, 1.1)', '[1.1, 1.6)']

# options_temp['moneyness_category'] = pd.cut(options_temp['moneyness'], bins=moneyness_bins, labe

# options_temp.groupby(['moneyness_category', 'maturity_category'])['impl_volatility'].agg(['count
```

```
/var/folders/jl/1kyt0fzd167dx2t_r6lh3tw00000gn/T/ipykernel_28205/4055701561.py:6: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  options_temp.groupby(['moneyness_category', 'maturity_category'])['impl_volatility'].agg(['count', 'mean', 'std'])
```

Out[ ]:

		count	mean	std
moneyness_category	**maturity_category**			
[0.8, 0.9)	**long**	72098	0.228038	0.030762
	short	118169	0.262003	0.045869
[0.9, 0.97)	**long**	82171	0.175370	0.030482
	short	237250	0.179726	0.040257
[0.97, 1.03)	**long**	106945	0.134289	0.031925
	short	356230	0.118971	0.038994
[1.03, 1.1)	**long**	59067	0.111526	0.027575
	short	110834	0.114556	0.035372
[1.1, 1.6)	**long**	13539	0.123511	0.028640
	short	4202	0.178557	0.058081

In [ ]:
```
# options_df_filtered.columns
```

Out[ ]:
```
Index(['date', 'exdate', 'cp_flag', 'optiondate', 'expirationdate', 'week_day',
       'strike_price', 'maturity', 'moneyness', 'index_level', 'index_return',
       'div_yield', 'riskfree', 'riskfree_option', 'best_bid', 'best_offer',
       'callprice', 'putprice', 'midquote', 'volume', 'open_interest',
       'impl_volatility', 'vega'],
      dtype='object')
```

In [ ]:
```
# final_df = options_df_filtered[['optiondate', 'maturity', 'strike_price', 'moneyness', 'impl_vol
# final_df.head()
```

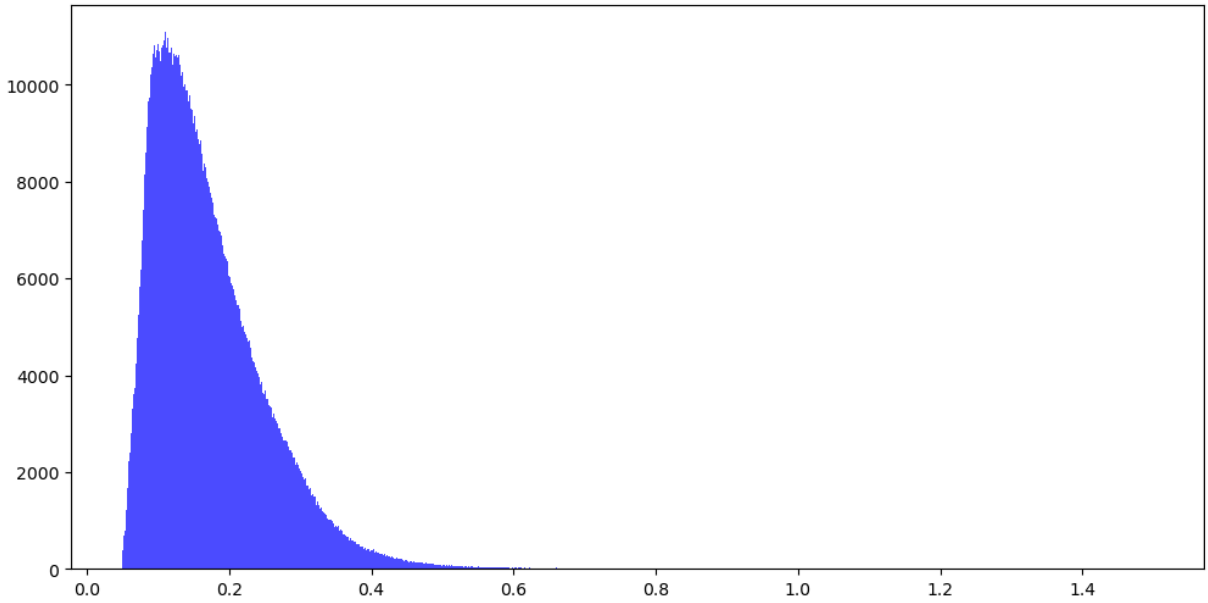
Out[ ]:

	optiondate	maturity	strike_price	moneyness	impl_volatility
0	736333	11	1875	0.931603	0.149723
1	736333	11	1880	0.934087	0.184999
2	736333	11	1890	0.939056	0.203981
3	736333	11	1900	0.944024	0.211677
4	736333	11	1910	0.948993	0.217900

In [ ]:
```
# final_df['impl_volatility'].describe()
```

Out[ ]:
```
count    1.772317e+06
mean     1.717228e-01
std      8.268354e-02
min      5.000100e-02
25%      1.113900e-01
50%      1.536130e-01
75%      2.132350e-01
max      1.498417e+00
Name: impl_volatility, dtype: float64
```

In [ ]:
```
# import matplotlib.pyplot as plt
# import seaborn as sns

# plt.figure(figsize=(12, 6))
# plt.hist(final_df['impl_volatility'], bins=1500, color='blue', alpha=0.7)
# plt.show()
```

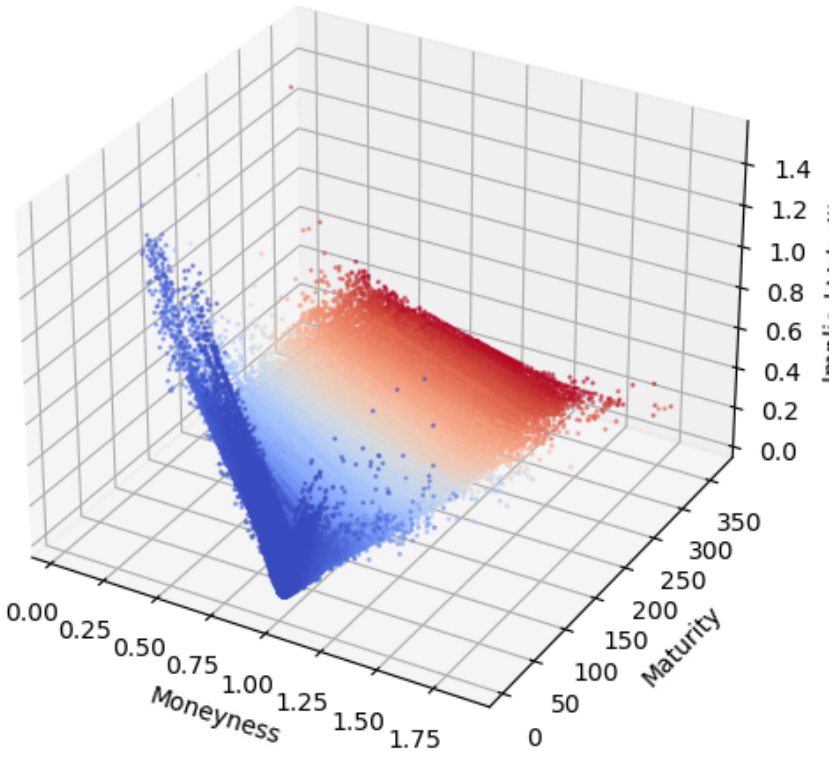


In [ ]:
```
# final_df.sort_values(['optiondate', 'maturity', 'strike_price', 'moneyness'], inplace=True)
# final_df.head()
```

Out[ ]:

	optiondate	maturity	strike_price	moneyness	impl_volatility
40	736333	11	1710	0.849622	0.439677
41	736333	11	1715	0.852106	0.432749
42	736333	11	1725	0.857075	0.421268
43	736333	11	1730	0.859559	0.414337
44	736333	11	1745	0.867012	0.397865

In [ ]:
```
# fig = plt.figure(figsize=(12, 6))
# ax = fig.add_subplot(111, projection='3d')
# ax.scatter(final_df['moneyness'], final_df['maturity'], final_df['impl_volatility'], c=final_df[
# ax.set_xlabel('Moneyness')
# ax.set_ylabel('Maturity')
# ax.set_zlabel('Implied Volatility')

# plt.show()
```



In [ ]:
```
# date_groups = final_df.groupby('optiondate')
# keys = list(date_groups.groups.keys())
```

In [ ]:
```
# from scipy.interpolate import griddata

# final_df['impl_volatility_1'] = np.nan
# final_df['impl_volatility_5'] = np.nan
# final_df['impl_volatility_21'] = np.nan

# lags = [1, 5, 21]

# for lag in lags:
#     for i in range(len(keys)-lag):
#         date_group1 = date_groups.get_group(keys[i])
#         date_group2 = date_groups.get_group(keys[i+lag])

#         X = date_group1['moneyness']
#         Y = date_group1['maturity']
#         Z = griddata((date_group2['moneyness'], date_group2['maturity']), date_group2['impl_vola

#         final_df.loc[date_group1.index, 'impl_volatility_'+str(lag)] = Z

# final_df.head()
```

Out[ ]:

	optiondate	maturity	strike_price	moneyness	impl_volatility	impl_volatility_1	impl_volatility_5	impl_volatility
40	736333	11	1710	0.849622	0.439677	0.312946	0.431986	0.309
41	736333	11	1715	0.852106	0.432749	0.312946	0.424897	0.309
42	736333	11	1725	0.857075	0.421268	0.312946	0.414703	0.309
43	736333	11	1730	0.859559	0.414337	0.312946	0.414703	0.309
44	736333	11	1745	0.867012	0.397865	0.312946	0.393329	0.309

In [ ]:
```
# final_df.tail()
```

Out[ ]:

	optiondate	maturity	strike_price	moneyness	impl_volatility	impl_volatility_1	impl_volatility_5	impl_vo
1770531	737790	353	3600	1.114282	0.114598	NaN	NaN	
1770532	737790	353	3700	1.145234	0.109926	NaN	NaN	
1770533	737790	353	3800	1.176187	0.108696	NaN	NaN	
1770534	737790	353	3900	1.207139	0.108730	NaN	NaN	
1770535	737790	353	4000	1.238091	0.110797	NaN	NaN	

In [ ]:
```
# final_df.dropna(inplace=True)
```

In [ ]:
```
# final_df.tail()
```

Out[ ]:

	optiondate	maturity	strike_price	moneyness	impl_volatility	impl_volatility_1	impl_volatility_5	impl_vo
1723982	737758	357	2750	0.875523	0.207395	0.192834	0.198356	
1723983	737758	357	3000	0.955116	0.178169	0.192834	0.185275	
1723984	737758	357	3025	0.963075	0.174868	0.192834	0.185275	
1723985	737758	357	3075	0.978994	0.168095	0.166515	0.185275	
1723986	737758	357	3200	1.018790	0.150406	0.157116	0.147164	

Lets start training the NN

In [ ]:
```
from google.colab import drive
import pandas as pd
import numpy as np

drive.mount('/content/drive')

final_df = pd.read_csv('/content/drive/My Drive/Term 3/Financial Econ/final_df.csv')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

1 day lag

In [ ]:
```
bs_impl_vol = final_df['impl_volatility'].mean()
bs_impl_vol_1 = final_df['impl_volatility_1'].mean()
bs_impl_vol_5 = final_df['impl_volatility_5'].mean()
bs_impl_vol_21 = final_df['impl_volatility_21'].mean()

final_df['impl_volatility'] = final_df['impl_volatility']-bs_impl_vol
final_df['impl_volatility_1'] = final_df['impl_volatility_1']-bs_impl_vol_1
final_df['impl_volatility_5'] = final_df['impl_volatility_5']-bs_impl_vol_5
final_df['impl_volatility_21'] = final_df['impl_volatility_21']-bs_impl_vol_21
```

In [ ]:
```
final_df.head()
```

Out[ ]:

	optiondate	maturity	strike_price	moneyness	impl_volatility	impl_volatility_1	impl_volatility_5	impl_volatility_
0	736333	11	1710	0.849622	0.267865	0.141944	0.263138	0.1435
1	736333	11	1715	0.852106	0.260937	0.141944	0.256049	0.1435
2	736333	11	1725	0.857075	0.249456	0.141944	0.245855	0.1435
3	736333	11	1730	0.859559	0.242525	0.141944	0.245855	0.1435
4	736333	11	1745	0.867012	0.226053	0.141944	0.224481	0.1435

In [ ]:
```
# from sklearn.model_selection import train_test_split

# X = final_df[['moneyness', 'maturity', 'impl_volatility']]
# y = final_df['impl_volatility_1']

# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [ ]:
```
X_train = final_df[final_df['strike_price']%10 == 0][['moneyness', 'maturity', 'impl_volatility',
X_test = final_df[final_df['strike_price']%10 != 0][['moneyness', 'maturity', 'impl_volatility', '
y_train = final_df[final_df['strike_price']%10 == 0]['impl_volatility_1']
y_test = final_df[final_df['strike_price']%10 != 0]['impl_volatility_1']
```

In [ ]:
```
X_train_tensor = torch.tensor(X_train.values, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test.values, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32)

train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=32)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)
test_loader = DataLoader(test_dataset, batch_size=32)
```

In [ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
```

In [ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [ ]:
```
class MyModel(nn.Module):
    def __init__(self, input_size, layer_sizes):
        super(MyModel, self).__init__()
        layers = []
        layers.append(nn.Linear(input_size, layer_sizes[0]))
        layers.append(nn.ReLU())
        for i in range(len(layer_sizes) - 1):
            layers.append(nn.Linear(layer_sizes[i], layer_sizes[i+1]))
            layers.append(nn.ReLU())
        self.model = nn.Sequential(*layers)
        self.output_layer = nn.Linear(layer_sizes[-1], 1)

    def forward(self, x):
        x = self.model(x)
        x = self.output_layer(x)
        return x.squeeze(1)
```

In [ ]:
```
nn1 = MyModel(input_size=X_train.shape[1], layer_sizes=[32])
nn2 = MyModel(input_size=X_train.shape[1], layer_sizes=[32, 16, 1])
nn3 = MyModel(input_size=X_train.shape[1], layer_sizes=[32, 16, 8, 1])
nn4 = MyModel(input_size=X_train.shape[1], layer_sizes=[32, 16, 8, 4, 1])
nn5 = MyModel(input_size=X_train.shape[1], layer_sizes=[32, 16, 8, 4, 2, 1])

nn1.to(device)
nn2.to(device)
nn3.to(device)
```

```
nn4.to(device)
nn5.to(device)
```

Out[ ]:
```
MyModel(
  (model): Sequential(
    (0): Linear(in_features=3, out_features=32, bias=True)
    (1): ReLU()
    (2): Linear(in_features=32, out_features=16, bias=True)
    (3): ReLU()
    (4): Linear(in_features=16, out_features=8, bias=True)
    (5): ReLU()
    (6): Linear(in_features=8, out_features=4, bias=True)
    (7): ReLU()
    (8): Linear(in_features=4, out_features=2, bias=True)
    (9): ReLU()
    (10): Linear(in_features=2, out_features=1, bias=True)
    (11): ReLU()
  )
  (output_layer): Linear(in_features=1, out_features=1, bias=True)
)
```

In [ ]:
```
criterion = nn.MSELoss()
optimizer = optim.Adam(nn1.parameters())

epochs = 10

for model in [nn1, nn2, nn3, nn4, nn5]:
    optimizer = optim.Adam(model.parameters())
    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * inputs.size(0)
        print(f"Epoch {epoch+1}/{10}, Loss: {running_loss / len(train_loader.dataset)}")
```

```
Epoch 1/10, Loss: 0.04804271825503772
Epoch 2/10, Loss: 0.0013056405913988813
Epoch 3/10, Loss: 0.0009851565161401719
Epoch 4/10, Loss: 0.0007829326426013592
Epoch 5/10, Loss: 0.0006893031605498265
Epoch 6/10, Loss: 0.0005766217889182728
Epoch 7/10, Loss: 0.0005149197022061941
Epoch 8/10, Loss: 0.0004967956196391878
Epoch 9/10, Loss: 0.0004946662046923782
Epoch 10/10, Loss: 0.0004937866821370214
Epoch 1/10, Loss: 0.0008137925679275323
Epoch 2/10, Loss: 0.0005054238997984343
Epoch 3/10, Loss: 0.000493009367358792
Epoch 4/10, Loss: 0.00048736024049840775
Epoch 5/10, Loss: 0.0004835682189113477
Epoch 6/10, Loss: 0.0004806092057155872
Epoch 7/10, Loss: 0.00047797153555427117
Epoch 8/10, Loss: 0.0004762681592276546
Epoch 9/10, Loss: 0.00047458098325560286
Epoch 10/10, Loss: 0.0004727823247375132
Epoch 1/10, Loss: 0.0065131641820095855
Epoch 2/10, Loss: 0.0065096065918018205
Epoch 3/10, Loss: 0.0065096065918018205
Epoch 4/10, Loss: 0.0065096065918018205
Epoch 5/10, Loss: 0.0065096065918018205
Epoch 6/10, Loss: 0.0065096065918018205
Epoch 7/10, Loss: 0.0065096065918018205
Epoch 8/10, Loss: 0.0065096065918018205
Epoch 9/10, Loss: 0.0065096065918018205
Epoch 10/10, Loss: 0.0065096065918018205
Epoch 1/10, Loss: 0.009144853793657941
Epoch 2/10, Loss: 0.0065096065918018205
Epoch 3/10, Loss: 0.0065096065918018205
Epoch 4/10, Loss: 0.0065096065918018205
Epoch 5/10, Loss: 0.0065096065918018205
Epoch 6/10, Loss: 0.0065096065918018205
Epoch 7/10, Loss: 0.0065096065918018205
Epoch 8/10, Loss: 0.0065096065918018205
Epoch 9/10, Loss: 0.0065096065918018205
Epoch 10/10, Loss: 0.0065096065918018205
Epoch 1/10, Loss: 0.007625703463499614
Epoch 2/10, Loss: 0.0065136114532949995
Epoch 3/10, Loss: 0.006512269471673049
Epoch 4/10, Loss: 0.006510090538174745
Epoch 5/10, Loss: 0.0065096065918018205
Epoch 6/10, Loss: 0.0065096065918018205
Epoch 7/10, Loss: 0.0065096065918018205
Epoch 8/10, Loss: 0.0065096065918018205
Epoch 9/10, Loss: 0.0065096065918018205
Epoch 10/10, Loss: 0.0065096065918018205
```

In [ ]:
```
torch.save(nn1.state_dict(), 'nn1_1.pt')
torch.save(nn2.state_dict(), 'nn2_1.pt')
torch.save(nn3.state_dict(), 'nn3_1.pt')
torch.save(nn4.state_dict(), 'nn4_1.pt')
torch.save(nn4.state_dict(), 'nn5_1.pt')
```

In [ ]:
```
for model in [nn1, nn2, nn3, nn4, nn4]:
    model.eval()
    test_running_loss = 0.0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            test_running_loss += loss.item() * inputs.size(0)
    test_loss = test_running_loss / len(test_loader.dataset)
    print(f'Test loss for {model.__class__.__name__} is {test_loss}')
```

```
Test loss for MyModel is 0.00046776193879443094
Test loss for MyModel is 0.00046242887835243524
Test loss for MyModel is 0.006490881921036101
Test loss for MyModel is 0.006490881921036101
Test loss for MyModel is 0.006490881921036101
```

In [ ]: