In [1]:
```
#Imports
import panel as pn
pn.extension('plotly')
import plotly.express as px
import pandas as pd
import hvplot.pandas
import matplotlib.pyplot as plt
import os
from pathlib import Path
from dotenv import load_dotenv
import requests
import alpaca_trade_api as tradeapi
import numpy as np
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from datetime import datetime, timedelta
%matplotlib inline
```

In [2]:
```
# Initialize the Panel Extensions (for Plotly)
import panel as pn
pn.extension("plotly")
```

In [3]:
```
#Stock Annaylsis
# Reading nasdaq returns
stock_price_csv = Path("Data/StockPriceData.csv")
nasdaq_stock_price = pd.read_csv(stock_price_csv, index_col=["Date"], parse_dates=Tr
nasdaq_stock_price.drop("Unnamed: 0", axis=1, inplace=True)
```

In [4]:
```
# Stock Close Price change
stock_change_csv = Path("Data/StockPriceChange.csv")
nasdaq_stock_change = pd.read_csv(stock_change_csv, index_col=["Unnamed: 0"])
```

In [5]:
```
#Sentiment Annaylsis
# Reading nasdaq sentiment
stock_sentiment_csv = Path("Data\stock_tweet_sentiment.csv")
nasdaq_stock_sentiment = pd.read_csv(stock_sentiment_csv, index_col=["Date"], parse_
# whale_returns.describe
nasdaq_stock_sentiment.drop("Unnamed: 0", axis=1, inplace=True)
```

In [6]:
```
# Stock Sentiment Annalysis
# Join the sentiment Score and Ticker Pricing for cross analysis

# Set up unique Index
# Reset index
nasdaq_stock_price.reset_index(inplace = True)
nasdaq_stock_sentiment.reset_index(inplace = True)

# Set up Referance to be able to match the data sets
nasdaq_stock_price['Ref'] = nasdaq_stock_price['Ticker'].astype(str) + nasdaq_stock_
nasdaq_stock_sentiment['Ref'] = nasdaq_stock_sentiment['Ticker'].astype(str) + nasda
```

```
# set Ref as new index
nasdaq_stock_price.set_index('Ref', inplace = True)
nasdaq_stock_sentiment.set_index('Ref', inplace = True)

# concatinate Stock price to Sentiment score
cross_analysis = pd.concat([nasdaq_stock_price, nasdaq_stock_sentiment],
                            join = 'outer',
                            axis = 'columns')

cross_analysis.reset_index(inplace = True)
#uniquily identify each coloum, and mark columns for deletion
cross_analysis.columns = ['Ref-Del','Date','Close','Ticker','Date-Del','Ticker-Del',
cross_analysis.drop(labels = ['Ref-Del','Date-Del','Ticker-Del'] ,axis = 'columns',

cross_analysis['Sentiment_Score'] = cross_analysis['Sentiment_Score']*1000
```

In [7]:

```
# Correlation
cross_analysis_corr = cross_analysis.groupby('Ticker')[['Close','Sentiment_Score']].

cross_analysis_corr.reset_index(inplace = True)

cross_analysis_corr = cross_analysis_corr[cross_analysis_corr['level_1'].isin(['Clos

cross_analysis_corr.drop(labels = ['level_1', 'Close'], inplace = True, axis = 'colu

cross_analysis_corr.columns = ['Ticker', 'Correlation']

cross_analysis_corr.dropna(inplace = True)

cross_analysis_corr.sort_values(by = 'Correlation',inplace = True, ascending = False
```

In [8]:

```
# Standard Deviation

#Calculate Standard Deviation of Percentage Change per Ticker
df_daily_std_nasdaq_stock_price = nasdaq_stock_price
df_daily_std_nasdaq_stock_price['Close'].pct_change()
df_daily_std_nasdaq_stock_price = df_daily_std_nasdaq_stock_price.groupby(by = "Tick
df_daily_std_nasdaq_stock_price.columns = ["Standard_Deviation"]
#remove NaNs
df_daily_std_nasdaq_stock_price.dropna(inplace = True)
# Add Comparison to Market
df_daily_std_nasdaq_stock_price.loc['00_Market Mean'] = df_daily_std_nasdaq_stock_pr
# Check Output
df_daily_std_nasdaq_stock_price.sort_values(by = 'Ticker', inplace = True)
df_daily_std_nasdaq_stock_price.reset_index(inplace = True)
```

In [9]:

```
# Current Sentiment Tool
# In[2]:


#load environment Variables
load_dotenv()

## Set Alpaca and Twitter API keys and secret keys
alpaca_api_key = os.getenv("ALPACA_API_KEY")
alpaca_secret_key = os.getenv("ALPACA_SECRET_KEY")
twitter_bearer_token = os.getenv("TWITTER_BEARER_TOKEN")


# In[3]:
```

```
# Define functions to be used

## Function to read 100 recent tweets related to ticker and from the date time speci
def read_100_Tweets(ticker, tweet_date_time):
    ## Function to read 100 recent tweets from the specific dates
    ##  Input : ticker - Stock Ticker
    ##          tweet_date_time - UTC Date/Time Format YYYY-MM-DDTHH:mm:ssZ (ISO 860
    ##
    ##  Output: List of 100 tweets
    auth_token = "Bearer " + twitter_bearer_token
    headers = {"Authorization": auth_token}
    twitter_api_url  = f"https://api.twitter.com/2/tweets/search/recent?max_results=
    response = requests.get(twitter_api_url, headers=headers)
    ## Check for 200 status code which means it was successful
    tweets_list = [];
    if(response.status_code == 200):
        json_response = response.json()
        #Check if there are any tweets at all
        if('data' in json_response.keys()):
            all_tweets = response.json()["data"]
            for tweet in all_tweets:
                tweets_list.append(tweet["text"])

    else:
        print(f"Response code: {response.status_code}. Error in getting the tweet")
        print(response.text)
    return tweets_list


## Perform Vader Sentiment Analysis
## Define Sentiment Object for Sentiment Analysis
sentiment_obj = SentimentIntensityAnalyzer()
def perform_sentiment_analysis(tweets_list):
    ## Function to read 100 recent tweets from the specific dates
    ##  Input : tweets_list - List of 100 tweets
    ##
    ##
    ##  Output: sentiment score average

    ##Check if there are tweets to analyse
    if (len(tweets_list) > 0):
        sentiment_scores_all = []
        for tweet in tweets_list:
            sentiment_dict = sentiment_obj.polarity_scores(tweet)
            sentiment_scores_all.append(sentiment_dict["compound"])

        #Average the sentiment of all tweets
        average_sentiment = np.average(sentiment_scores_all)
        return average_sentiment
    else:
        return 0;


# In[4]:


# Input ticker to analyse using Select Widget

# Import Ticker list
ticker_list = pd.read_csv('Data/TickerList.csv',header=None)
ticker_list = list(ticker_list[0])
ticker_list.sort()
```

```
# Set up Select ticker widget
select_ticker = pn.widgets.Select(options = ticker_list, name = 'Choose Ticker')


def collect_clean_data_API(input_ticker):

    if type(input_ticker) == str:
        ticker = input_ticker
    else:
        ticker = input_ticker.value

#     ticker = input_ticker.value


    ## Set time Variables
    # Set adjustable variables
    seven_day_delta = timedelta(days=6)
    one_day_delta = timedelta(days=1)
    #Set the end date as yesterday
    end_date_time = datetime.today() - one_day_delta
    end_date = end_date_time #.date()
    #Set the start date as end date - 7 days
    start_date_time = end_date_time - seven_day_delta
    start_date = start_date_time #.date()
    #Set the time to 1PM for tweet retrieval
    start_date_time = start_date_time.replace(minute=0, hour=13, second=0)
    end_date_time = end_date_time.replace(minute=0, hour=13, second=0)



    ## Fetch and capture Ticker price data
    # Create the Alpaca API object
    alpaca = tradeapi.REST(
    alpaca_api_key,
    alpaca_secret_key,
    api_version="v2")
    # Format current date as ISO format
    start_date = pd.Timestamp(start_date, tz="America/New_York").isoformat()
    end_date = pd.Timestamp(end_date, tz="America/New_York").isoformat()
    # Set timeframe of stock bars to collect
    timeframe = "4Hour"
    # make API call to Alpaca to receive a data frame of selected ticker stock data
    df_stock_data = alpaca.get_bars(
        ticker,
        timeframe,
        end = end_date,
        start = start_date
    ).df
    # clean df_stock_data
    # remove unneeded columns
    df_stock_data.drop(['open','high','low','volume','trade_count','vwap'], axis='co
    # reset index
    df_stock_data.reset_index(inplace = True)
    # change timestamp to date only
#     df_stock_data.loc[:,'timestamp'] = df_stock_data.loc[:,'timestamp'].dt.date
    # change column names to more suitable names
    df_stock_data.columns = ['Date', 'Close']



    ## Fetch and capture Ticker sentiments
    tweet_sentiments = []
    analysis_date_time = start_date_time
    while analysis_date_time <= end_date_time:
    #    print(f"Executing Tweet Analysis for {ticker} on {analysis_date_time.isofor
        tweets_list = read_100_Tweets(ticker, analysis_date_time.isoformat() + "Z")
        sentiment_score = perform_sentiment_analysis(tweets_list)
        tweet_sentiment = {}
        tweet_sentiment["Ticker"] = ticker
        tweet_sentiment["Date"] = analysis_date_time
        tweet_sentiment["Sentiment_Score"] = sentiment_score
        tweet_sentiments.append(tweet_sentiment)
            #print("Ticker: " + tweet_sentiment["ticker"] + ", Date : " + str(tweet_
        analysis_date_time += one_day_delta

    # convert Tweet sentiment data to Data Frame
    stock_tweet_sentiment_df = pd.DataFrame(tweet_sentiments)

    return stock_tweet_sentiment_df, df_stock_data





# Plot the Data from collect_clean_data_API()
@pn.depends(select_ticker) # this will automattically push plot_recent_sentiment_vs_
def plot_recent_sentiment_vs_close(input_ticker):
    stock_tweet_sentiment_df, df_stock_data = collect_clean_data_API(input_ticker)


    ##Create Line plot output
    stock_plot = df_stock_data.hvplot.line(x = 'Date',
                                                         y = 'Close',
                                                         height = 400,
                                                         width = 600,
                                                         title = 'Stock Close Pr
                                                         ylabel = 'Closeing Pric
                                                         )

    sent_plot = stock_tweet_sentiment_df.hvplot.line(x = 'Date',
                                                        y = 'Sentiment_Score',
                                                        height = 400,
                                                        width = 600,
                                                        title = 'Sentiment Score
                                                        ylabel = 'Sentiment Scor
                                                        color = 'red')


    line_plot = (stock_plot + sent_plot)

    return line_plot
```

In [10]:

```
# Define Panel visualization functions

# nasdaq plot to show returns
def nasdaq_stock_plot(nasdaq_stock_price):
    plot = nasdaq_stock_price.hvplot.line(x='Date',
                                          y='Close',
                                          rot=90,
                                          width=800,
                                          groupby='Ticker',
                                          widget_location = 'left_top',
                                          title = "Market Returns",
                                          )
    return plot
```

```python
# nasdaq plot to show % change
def nasdaq_stock_change_plot(nasdaq_stock_change, nasdaq_stock_price):
    ticker_list = list(set(nasdaq_stock_price['Ticker']))
    plot = nasdaq_stock_change[nasdaq_stock_change['Ticker'].isin(ticker_list)].hvpl







    return plot

# nasdaq plot to show returns
def nasdaq_stock_sentiment_plot(nasdaq_stock_sentiment):
    plot = nasdaq_stock_sentiment.hvplot.line(x='Date',
                                              y='Sentiment_Score',
                                              rot=90,
                                              width=800,
                                              groupby='Ticker',
                                              title = "Tickers by Twitter Sentiment"
                                              widget_location = 'left_top',
                                              color = 'red')
    return plot


# nasdaq cross analysis (stock vs sentiment)
def nasdaq_cross_annalysis_plot(cross_analysis):
    plot = cross_analysis.hvplot.scatter(x = 'Date',
                                y = 'Close',
                                c = 'Sentiment_Score',
                                size = 'Sentiment_Score',
                                rot = 90,
                                width = 800,
                                groupby = 'Ticker',
                                widget_location = 'left_top',
                                title = "Plot to Show Stock Price Against Sentiment Sc
    return plot


# Correlation of stock sentiment
def plot_cross_analysis_corr(cross_analysis_corr):
    plot = cross_analysis_corr.hvplot.bar(x='Ticker',
                                          y='Correlation',
                                          c='Correlation',
                                          rot=90,
                                          width=800,
                                          height = 300,
                                          title = "Stock Price vs Correlation")
    return plot


def def_daily_std_nasdaq_stock_price(df_daily_std_nasdaq_stock_price):

    plot = df_daily_std_nasdaq_stock_price.hvplot.bar(
        x='Ticker',
        y='Standard_Deviation',
        rot=90,
        width=800,
        height = 300,
        title = "Standard Deviation",
        color = 'pink')

    return plot
```

```
# Define a welcome text
welcome_text = pn.pane.Markdown('''
###This is the most awesome dashboard EVER!!!!

This is the outputs of all Sentiment Analysis completed for the Fintech Project 1
''')

# Define a welcome image
# Image source https://www.britannica.com/place/Toronto
welcome_image = pn.pane.Markdown('''
<img width="859" alt="Welcome" src="https://user-images.githubusercontent.com/979963
''')
```

In [11]:

```
#nasdaq_stock_plot(nasdaq_stock_price)
```

In [12]:

```
# Panel DashBoard

# Welcome Text
welcometab = pn.Column(welcome_text,welcome_image)

# Create a tab layout for the dashboard

nasdaq_stock_price_column = pn.Column(
    "#Nasdaq Stock Price and Sentiment", nasdaq_stock_plot(nasdaq_stock_price),nasda

nasdaq_price_change_percent_column = pn.Column(
    "#Market Selection Sample Nasdaq Price Change Percentage", nasdaq_stock_change_p

def_daily_std_nasdaq_stock_price_tab = pn.Column(
    "#Market Standard Deviation", def_daily_std_nasdaq_stock_price(df_daily_std_nasd


nasdaq_stock_sentiment_column = pn.Column(
     "#Nasdaq Stock Sentiment", nasdaq_stock_sentiment_plot(nasdaq_stock_sentiment))

stock_sentiment_analysis_column = pn.Column(
    "#Nasdaq Stock Cross Annalysis", nasdaq_cross_annalysis_plot(cross_analysis))

stock_sentiment_analysis_corr_column = pn.Column(
    "#Nasdaq Stock Sentiment Correlation", plot_cross_analysis_corr(cross_analysis_c

current_sentiment_tool = pn.Column(
    '#Current Sentiment Tool',select_ticker, plot_recent_sentiment_vs_close)

# Create Main Dashboard
stock_sentiment_dashboard = pn.Tabs(
                                 ("Welcome", welcometab),
                                 ("Nasdaq Stock Price and Sentiment", nasdaq_stock_p
                                 ("Market Standard Deviation", def_daily_std_nasdaq_
                                 ("Nasdaq Price Change %", nasdaq_price_change_perce
                                 ("Nasdaq Cross Analysis", stock_sentiment_analysis_
                                 ("Nasdaq Stock Sentiment Correlation", stock_sentim
                                 ("Current Sentiment Tool", current_sentiment_tool)
                                  )
```

In [13]:

```
# Show Dashboard
# dashboard.servable(location=True, area='main')
stock_sentiment_dashboard.show()
```

```
        Launching server at http://localhost:57904
Out[13]: <bokeh.server.server.Server at 0x230f4305ac8>
```