analysis

June 20, 2025

1 Stock Market Data Analysis

[ ]:
```
#imports
import pandas as pd
from pandas import Series,DataFrame
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline
from datetime import datetime
from __future__ import division
pd.core.common.is_list_like = pd.api.types.is_list_like
import pandas_datareader as web
import yfinance as yf
```

Obtain Data from Yahoo

[ ]:
```
tech_list = ['AAPL','GOOG','MSFT','AMZN']

end = datetime.now()
start = datetime(end.year - 1, end.month,end.day)
stock_data = {}

for stock in tech_list:
    data = yf.download(stock, start=start, end=end)
    globals()[stock] = data
    # print(f"{stock} data. Head:\n", data.head(), "\n")
```

[ ]:
```
AAPL.head()
```

[ ]:
```
AAPL.describe()
```

[ ]:
```
AAPL.info()
```

1.0.1 Closing Price

```
[ ]: AAPL['Close'].plot(legend=True, figsize = (10,6))
     plt.show()
```

1.0.2 Volume Traded

```
[ ]: AAPL["Volume"].plot(legend=True,figsize=(10,6))
     plt.show()
```

1.0.3 Moving Average (MA)

Moving Average(MA) is a widely used indicator in technical analysis that helps smooth out price action by filtering out "noise" from random price fluctuations. It is a trend-following, or lagging, indicator because it is based on past prices.

```
[ ]: ma_day = [10, 20, 30 , 50]

     for ma in ma_day:
         column_name = "MA for %s days" %(str(ma))
         AAPL[column_name] = pd.DataFrame.rolling(AAPL['Close'],ma).mean()


     AAPL[['Close', 'MA for 10 days','MA for 20 days','MA for 30 days','MA for 50␣
      ↪days']].plot(subplots=False,figsize=(12,6))
     plt.show()
```

1.0.4 Daily Return Analysis

Now, we will analyze the risk of stock and in order to do so, we need take a closer look at the daily changes of the stock and not just the absolute values.

```
[ ]: AAPL['Daily Return'] = AAPL['Close'].pct_change()
     AAPL['Daily Return'].plot(figsize=(10,5),legend=True, linestyle='--',marker='o')
     plt.show()
```

```
[ ]: sns.distplot(AAPL['Daily Return'].dropna(),bins=100,color="green")
     plt.show()
```

Let's analyze the returns of all the stocks in our `tech_list`.

```
[ ]: closing_df1 = pd.DataFrame(AAPL["Close"])
     Close1 = closing_df1.rename(columns={"Close": "AAPL_Close"})

     closing_df2 = pd.DataFrame(GOOG["Close"])
     Close2 = closing_df2.rename(columns={"Close": "GOOG_Close"})

     closing_df3 = pd.DataFrame(MSFT["Close"])
```

```python
Close3 = closing_df3.rename(columns={"Close": "MSFT_Close"})

closing_df4 = pd.DataFrame(AMZN["Close"])
Close4 = closing_df4.rename(columns={"Close": "AMZN_Close"})

closing_df = pd.concat([Close1, Close2, Close3, Close4], axis=1)
closing_df.head()
```

[ ]:
```python
tech_returns = closing_df.pct_change()
```

[ ]:
```python
sns.jointplot(
    x="GOOG", y="GOOG", data=tech_returns, kind="scatter", color="red",␣
 ↪legend=True
)
plt.show()
```

[ ]:
```python
sns.pairplot(tech_returns.dropna())
plt.show()
```

We obtain all the relationships on daily returns between all the stocks.

The following gives us a control over what plot we want in the diagonal, upper triangle and the lower triangle.

[ ]:
```python
# to ignore warnings
import warnings

warnings.filterwarnings("ignore")
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
returns_fig = sns.PairGrid(tech_returns.dropna())

# Using map_upper we can specify what the upper triangle will look like.
returns_fig.map_upper(plt.scatter, color="purple")

# We can also define the lower triangle in the figure, inclufing the plot type␣
 ↪(kde) or the color map (BluePurple)
returns_fig.map_lower(sns.kdeplot, cmap="cool_d")

# Finally we'll define the diagonal as a series of histogram plots of the daily␣
 ↪return
returns_fig.map_diag(plt.hist, bins=30)


plt.show()
```

[ ]:
```python
returns_fig = sns.PairGrid(closing_df)
returns_fig.map_upper(plt.scatter,color='purple')
returns_fig.map_lower(sns.kdeplot,cmap="cool_d")

returns_fig.map_diag(plt.hist,bins=30)

plt.show()
```

1.0.5 Correlation Plot

[ ]:
```
corr = tech_returns.dropna().corr()

mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True

fig,ax = plt.subplots(figsize=(10,8))

cmap = sns.diverging_palette(220,10,as_cmap=True)

sns.heatmap(corr,mask=mask,cmap=cmap,annot=True)

plt.show()
```

1.0.6 Risk Analysis

There are many ways we can quantify risk, one of the most basic ways using the information we've gathered on daily percentage returns is by comparing the expected return with the standard deviation of the daily returns.

[ ]:
```
rets = tech_returns.dropna()

area = np.pi*20

plt.scatter(rets.mean(), rets.std(), alpha=0.5, s=area)

plt.xlabel('Expected returns')
plt.ylabel('Risk')

plt.ylim([0.01,0.025])

plt.xlim([-0.003,0.005])

for label,x,y in zip(rets.columns,rets.mean(),rets.std()):
    plt.annotate(
        label,
        xy = (x,y),xytext=(x + 0.0005, y + 0.0005),
        textcoords= 'offset points' , ha= 'right', va='bottom',
        arrowprops=dict(arrowstyle='-', connectionstyle='arc3,rad=-0.3'))
```

Value at risk using the "bootstrap" method

[ ]:
```
sns.distplot(AAPL["Daily Return"].dropna(), bins=100)
plt.show()
```

[ ]:
```
rets.head()
```

[ ]:
```
emp = rets["AAPL"].quantile(0.05)
print("The 0.05 empirical quantile of daily returns is at", emp)
```

1.0.7 Value at Risk using the Monte Carlo Method

[ ]:
```
days = 365

deltaT = 1/days

mu = rets.mean()['GOOG']

sigma = rets = rets.std()['GOOG']
```

[ ]:
```
def monte_carlo_simulation(start_price,days,mu,sigma):

    price = np.zeros(days)
    price[0]= start_price

    shock = np.zeros(days)
    drift = np.zeros(days)

    for x in range(1,days):
        shock[x] = np.random.normal(loc=mu*deltaT,scale=sigma*np.sqrt(deltaT))
        drift[x] = mu*deltaT
        price[x] = price[x-1] + (price[x-1]*(drift[x]+shock[x]))

    return price
```

[ ]:
```
GOOG.head()
```

[ ]:
```
start_price = 176.860947

for run in range(100):
    plt.plot(monte_carlo_simulation(start_price,days,mu,sigma))

plt.xlabel("Days")
plt.ylabel("Price")
plt.title("Monte Carlo Analysis for Google Stock\n", fontsize=14)
plt.show()
```

```
[ ]: runs = 10000

     simulations = np.zeros(runs)
     np.set_printoptions(threshold=5)

     for run in range(runs):
         simulations[run] = monte_carlo_simulation(start_price, days, mu,␣
      ↪sigma)[days - 1]
```

```
[ ]: q = np.percentile(simulations, 1)

     plt.hist(simulations, bins=200)

     plt.figtext(0.6, 0.8, s="Start price: $%.2f" % start_price)

     plt.figtext(0.6, 0.7, "Mean final price: $%.2f" % simulations.mean())

     plt.figtext(0.6, 0.6, "VaR(0.99): $%.2f" % (start_price - q,))

     plt.figtext(0.15, 0.6, "q(0.99): $%.2f" % q)

     plt.axvline(x=q, linewidth=4, color="r")

     plt.title(
         "Final price distribution for Google Stock after %s days\n" % days,␣
      ↪weight="bold"
     )
```