https://docs.google.com/presentation/d/1M6zU7oJcxutlxPiBCvM1ULSsw9anUcM-43E6m0CBveo/edit?usp=sharing

https://docs.google.com/document/d/17F4O7WKpNpaome-yb2YiMmaynvhc6IvgnQRjpm5c9NQ/edit?usp=sharing

1) (Nels) Explain your question and provide some background information: why is the question important/interesting; institution background.

   Earning announcements can have a significant impact on the price of stocks and we aim to understand the relationship between earnings announcements and the stock's price. A thorough understanding of how earnings announcement release impact the price of a stock can help us to make informed investment decisions and minimize investment risk.

2) Your test design – discuss how you want to test your question, what empirical tests you will use and why you choose these tests.
   To test our research question, if earnings announcements can have a significant impact on the price of stocks, we decided to collect data from five different industries: Energy, Real

3) Get data – discuss sample selection – is the sample representative of the population. If not, discuss the limitations.

   The information about common stock prices was gathered from yahoo finance and the earnings announcement related information was collected from Alphavantage. We have chosen five industries to observe the relationship between stock prices and earning announcements since various industries often perform differently on the stock market. The industries are energy, real estate, technology, retail, and pharmaceuticals.

4) Calculate and discuss descriptive statistics, visualize the data – what do you learn about the property of the data from this exercise. Does it provide a preliminary answer to your question? (The answer will depend on your situation.)

5) Form and state test hypothesis.
   H_0 = Earnings surprise - Price Difference Before & After Announcement = 0
   (Earning Announcement has no effect on stock price)
   H_a = Earnings surprise - Price Difference Before & After Announcement ≠ 0
   (Earning Announcement has an effect on stock price)

6) Test your hypothesis – discuss the proper test statistic and significance level.
   Alpha = 0.05

7) Use regression analysis to test your hypothesis.

8) When applicable, use one of the methods in the last class (Monte Carlo Simulation, event study, portfolio sorts to test your hypothesis. You may want to include part 7 and 8 in part 5 and 6 because these are additional methods of hypothesis testing.

9) Interpret your findings and conclude – discuss both statistical and economic significance.

10) Prepare a 10-minute presentation (ppt will be helpful) of the final project. Each student should participate in the presentation

Effect of Earnings announcements:

Test Window: 10 years, only Q4 earnings, 1 week prior to announcement, 1 week after announcement

Data to gather: Expected EPS, Actual EPS, daily volatility change, percent they beat or lost earnings, average volatility, simple daily return, volume,

- Test different industries (3 individual stocks)
    - Energy (Daisy)
    - Real estate (Nels)
    - Tech (Terry)
    - Retail (Yiyang)
    - Pharmaceutical (Jesus)
- Does Q4 impact certain industries more than others?
- Economic cycle
- Potentially look into effects stock splits
- Study volatility and price change before and after announcement of earnings
- Use SPY as baseline for comparison

```
-   import requests
-
-   # replace YOUR_API_KEY with your actual API key from Alpha Vantage
-   api_key = 'YOUR_API_KEY'
-
-   # specify the stock symbol for which you want to retrieve earnings data
-   symbol = 'AAPL'
-
-   # make a GET request to retrieve the earnings report data for the specified stock
-   response = requests.get(f'https://www.alphavantage.co/query?function=EARNINGS&symbol={symbol}&apikey={api_key}')
-
-   # convert the response content to a JSON object
-   data = response.json()
-
-   # extract the earnings data from the JSON object
-   earnings = data['quarterlyEarnings']
-
-   # print the earnings report data between 2013 and 2022
-   for quarter in earnings:
-       reportedDate = quarter['reportedDate']
-       if '2013-01-01' <= reportedDate <= '2022-12-31':
-           # check if the fiscal date ending month is between 10 and 12 (inclusive) to filter to Q4 reports
-           fiscal_month = int(reportedDate.split('-')[1])
-           if fiscal_month >= 10 and fiscal_month <= 12:
-               print('reportedDate:', quarter['reportedDate'])
-               print('Reported EPS:', quarter['reportedEPS'])
-               if 'reportedRevenue' in quarter:
-                   print('Reported Revenue:', quarter['reportedRevenue'])
-               print('Estimated EPS:', quarter['estimatedEPS'])
-               if 'estimatedRevenue' in quarter:
-                   print('Estimated Revenue:', quarter['estimatedRevenue'])
-               if 'surprise' in quarter:
-                   print('Surprise:', quarter['surprise'])
-               if 'surprisePercentage' in quarter:
-                   print('SurprisePercentage:', quarter['surprisePercentage'])
-               print('---')
-
```

To make the data to be Data Frame

```
import requests
# sign in to alpha vantage to get your own key
# replace YOUR_API_KEY with your actual API key from Alpha Vantage
api_key = 'API_KEY'

# specify the stock symbol for which you want to retrieve earnings data
symbol = 'NEE'

# make a GET request to retrieve the earnings report data for the specified stock
response =
requests.get(f'https://www.alphavantage.co/query?function=EARNINGS&symbol={symbol}&apik
ey={api_key}')


# convert the response content to a JSON object
data = response.json()


# extract the earnings data from the JSON object
earnings = data['quarterlyEarnings']

Date_list=[]
earning_diff=[]
Expected_EPS= []
Actual_EPS= []

# print the earnings report data between 2013 and 2022
for quarter in earnings:
    reportedDate = quarter['reportedDate']

    if '2014-01-01' <= reportedDate <= '2023-03-01':

        fiscal_month = int(reportedDate.split('-')[1])

        if fiscal_month >= 1 and fiscal_month <= 3:
            Date_list.append(quarter['reportedDate'])
            print('reportedDate:', quarter['reportedDate'])

            Actual_EPS.append(quarter['reportedEPS'])
            print('Reported EPS:', quarter['reportedEPS'])

            if 'reportedRevenue' in quarter:
                print('Reported Revenue:', quarter['reportedRevenue'])
```

```
        Expected_EPS.append(quarter['estimatedEPS'])
        print('Estimated EPS:', quarter['estimatedEPS'])

        if 'estimatedRevenue' in quarter:
            print('Estimated Revenue:', quarter['estimatedRevenue'])

        if 'surprise' in quarter:
            earning_diff.append(quarter['surprise'])
            print('Surprise:', quarter['surprise'])

        if 'surprisePercentage' in quarter:
            print('SurprisePercentage:', quarter['surprisePercentage'])

        print('---')


        df = pd.DataFrame(Expected_EPS,
                          index = Date_list,
                           columns = ['Expected EPS'])

        df1 = pd.DataFrame(Actual_EPS,
                           index = Date_list,
                           columns = ['Actual EPS'])

        df2 = pd.DataFrame(earning_diff,
                           index = Date_list,
                           columns = ['Actual - Expected EPS'])


earning_table = pd.merge(pd.merge(df,df1,left_index=True,right_index=True),
                         df2,left_index=True,right_index=True)

earning_table
```

```
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', -1)
jan_feb_data = sec_data_daily[(sec_data_daily['ABBV'].index.month == 1) |
(sec_data_daily['ABBV'].index.month == 2)]
jan_feb_data = jan_feb_data.drop(index=jan_feb_data.index[(jan_feb_data.index.weekofyear
>= 26) & (jan_feb_data.index.weekofyear <= 35)])
jan_feb_data['ABBV']
```

```
import yfinance as yf
import pandas as pd
import datetime
import finnhub

API_KEY = "cg989a9r01qk68o80ft0cg989a9r01qk68o80ftg"

def get_earnings_dates(stock, start_year, years=10):
    finnhub_client = finnhub.Client(api_key=API_KEY)
    earnings_data = finnhub_client.company_earnings(stock, limit=years*4)

    print(f"Earnings data for {stock}:")
    print(earnings_data)

    earnings_dates = [
        datetime.datetime.strptime(ed["date"], "%Y-%m-%d").date()
        for ed in earnings_data
        if ed["period"].endswith("Q4") and start_year <= int(ed["period"][:4]) <
start_year + years
    ]

    return earnings_dates


def analyze_stock(stock, earnings_dates):
    start_date = datetime.date.today() - datetime.timedelta(days=365 * 10)
    end_date = datetime.date.today()

    data = yf.download(stock, start=start_date, end=end_date)
    results = []

    for date in earnings_dates:
        week_before = date - pd.DateOffset(days=7)
        week_after = date + pd.DateOffset(days=7)
        week_data = data.loc[week_before:week_after]
```

```python
        week_data['daily_return'] = week_data["Adj Close"] / week_data["Adj Close"].shift()
        simple_daily_change = week_data['daily_return'].dropna()
        results.append(simple_daily_change)

    if results:
        return pd.concat(results)
    else:
        print(f"No earnings dates found for {stock}")
        return None

def main():
    stock_tickers = ["MSFT", "AAPL", "GOOGL", "AMZN", "Z"]
    start_year = 2011
    years = 10
    all_results = []

    for stock in stock_tickers:
        earnings_dates = get_earnings_dates(stock, start_year, years)
        simple_daily_changes = analyze_stock(stock, earnings_dates)

        if simple_daily_changes is not None:
            all_results.append((stock, simple_daily_changes))

    results_df = pd.DataFrame(all_results, columns=["Ticker", "Earnings Data"])
    print(results_df)


if __name__ == "__main__":
    main()
```

USE THIS TO REPLACE TEXT: https://www.browserling.com/tools/text-replace

```
Energy_Data['Actual - Expected EPS'] = Energy_Data['Actual - Expected EPS'].astype(float)

Alpha = 0.05

sst.ttest_ind(a=Energy_Data['Actual - Expected EPS'],
          b=Energy_Data['Price Diff Before & After Ann'])

plt.scatter(Energy_Data['Actual - Expected EPS'],
         Energy_Data['Price Diff Before & After Ann'],
         color='blue')

m, b = np.polyfit(Energy_Data['Actual - Expected EPS'],
            Energy_Data['Price Diff Before & After Ann'], 1)

plt.plot(Energy_Data['Actual - Expected EPS'],
       m*Energy_Data['Actual - Expected EPS']+b, color='red')

plt.title('Earning Surprise Vs Price Diff', fontsize=14)
plt.xlabel('Earning Surprise', fontsize=12)
plt.ylabel('Stock Price Diff', fontsize=14)
plt.grid(True)
plt.show()
```

Correlation test:

```
from scipy.stats import pearsonr
full_correlation = pearsonr(retail_data['Actual - Expected EPS'],retail_data['Price Diff Before & After Ann'])
print('Correlation coefficient:',full_correlation[0])
print('P-value:',full_correlation[1])
```

Regression test:

```
x = retail_data["Actual - Expected EPS"]
y = retail_data['Price Diff Before & After Ann']

x = sm.add_constant(x)

model = sm.OLS(y, x).fit()
model.summary()
```

#####Terry regression

Dep. Variable:	Price Diff Before & After Ann	**R-squared:**	0.009

Model:	OLS	**Adj. R-squared:**	-0.026
Method:	Least Squares	**F-statistic:**	0.2537
Date:	Thu, 16 Mar 2023	**Prob (F-statistic):**	0.618
Time:	18:08:51	**Log-Likelihood:**	41.040
No. Observations:	30	**AIC:**	-78.08
Df Residuals:	28	**BIC:**	-75.28
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.0140	0.013	1.056	0.300	-0.013	0.041
Actual - Expected EPS	0.0519	0.103	0.504	0.618	-0.159	0.263

Omnibus:	23.075	**Durbin-Watson:**	1.365
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	39.789
Skew:	1.730	**Prob(JB):**	2.29e-09
Kurtosis:	7.457	**Cond. No.**	8.89

Daisy Regression:

Dep. Variable:	Price Diff Before & After Ann	**R-squared:**	0.045
Model:	OLS	**Adj. R-squared:**	0.011
Method:	Least Squares	**F-statistic:**	1.331
Date:	Thu, 16 Mar 2023	**Prob (F-statistic):**	0.258
Time:	18:01:17	**Log-Likelihood:**	-89.137
No. Observations:	30	**AIC:**	182.3
Df Residuals:	28	**BIC:**	185.1
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-0.7575	0.912	-0.830	0.413	-2.627	1.112
Actual - Expected EPS	-5.3518	4.639	-1.154	0.258	-14.854	4.151

Omnibus:	2.053	**Durbin-Watson:**	1.677
Prob(Omnibus):	0.358	**Jarque-Bera (JB):**	1.185
Skew:	-0.109	**Prob(JB):**	0.553
Kurtosis:	2.051	**Cond. No.**	5.21

Here is a condensed version of the regression report analysis:

R-squared: 0.045 - Indicates a weak relationship between the variables, as only 4.5% of the variation in the Price Diff Before & After Ann can be explained by the Actual - Expected EPS.
Prob (F-statistic): 0.258 - Suggests that the model is not statistically significant since the p-value is greater than 0.05.
Actual - Expected EPS coefficient: -5.3518 - Implies an inverse relationship between the variables, but the relationship is not statistically significant (P>|t| = 0.258).
Durbin-Watson: 1.677 - Indicates that autocorrelation is not a significant concern in the model.
In summary, the regression analysis shows a weak and statistically non-significant relationship between the "Actual - Expected EPS" and the "Price Diff Before & After Ann". The model may not be reliable for predicting the impact of earnings surprises on stock returns.

```
####CODE FOR STD###

#GOOGL 2013(Q4) Before Earning Announcement
GOOGL_13_b4 = pdr.get_data_yahoo("GOOGL",start = "2014-01-20", end = "2014-01-27")

#GOOGL 2014(Q4) Before Earning Announcement
GOOGL_14_b4 = pdr.get_data_yahoo("GOOGL",start = "2015-01-20", end = "2015-01-27")

#GOOGL 2015(Q4) Before Earning Announcement
GOOGL_15_b4 = pdr.get_data_yahoo("GOOGL",start = "2016-01-19", end = "2016-01-26")

#GOOGL 2016(Q4) Before Earning Announcement
GOOGL_16_b4 = pdr.get_data_yahoo("GOOGL",start = "2017-01-24", end = "2017-01-31")

#GOOGL 2017(Q4) Before Earning Announcement
GOOGL_17_b4 = pdr.get_data_yahoo("GOOGL",start = "2018-01-25", end = "2018-02-01")

#GOOGL 2018(Q4) Before Earning Announcement
GOOGL_18_b4 = pdr.get_data_yahoo("GOOGL",start = "2019-01-22", end = "2019-01-29")

#GOOGL 2019(Q4) Before Earning Announcement
GOOGL_19_b4 = pdr.get_data_yahoo("GOOGL",start = "2020-01-21", end = "2020-01-28")

#GOOGL 2020(Q4) Before Earning Announcement
GOOGL_20_b4 = pdr.get_data_yahoo("GOOGL",start = "2021-01-20", end = "2021-01-27")

#GOOGL 2021(Q4) Before Earning Announcement
GOOGL_21_b4 = pdr.get_data_yahoo("GOOGL",start = "2022-01-20", end = "2022-01-27")

#GOOGL 2022(Q4) Before Earning Announcement
GOOGL_22_b4 = pdr.get_data_yahoo("GOOGL",start = "2023-01-27", end = "2023-02-02")
```

```
GOOGL_13_b4mean = GOOGL_13_b4['Adj Close'].mean()

GOOGL_14_b4mean = GOOGL_14_b4['Adj Close'].mean()

GOOGL_15_b4mean = GOOGL_15_b4['Adj Close'].mean()

GOOGL_16_b4mean = GOOGL_16_b4['Adj Close'].mean()

GOOGL_17_b4mean = GOOGL_17_b4['Adj Close'].mean()

GOOGL_18_b4mean = GOOGL_18_b4['Adj Close'].mean()

GOOGL_19_b4mean = GOOGL_19_b4['Adj Close'].mean()

GOOGL_20_b4mean = GOOGL_20_b4['Adj Close'].mean()

GOOGL_21_b4mean = GOOGL_21_b4['Adj Close'].mean()

GOOGL_22_b4mean = GOOGL_22_b4['Adj Close'].mean()

GOOGL_13_b4std = GOOGL_13_b4['Adj Close'].std()

GOOGL_14_b4std = GOOGL_14_b4['Adj Close'].std()

GOOGL_15_b4std = GOOGL_15_b4['Adj Close'].std()

GOOGL_16_b4std = GOOGL_16_b4['Adj Close'].std()

GOOGL_17_b4std = GOOGL_17_b4['Adj Close'].std()

GOOGL_18_b4std = GOOGL_18_b4['Adj Close'].std()

GOOGL_19_b4std = GOOGL_19_b4['Adj Close'].std()

GOOGL_20_b4std = GOOGL_20_b4['Adj Close'].std()

GOOGL_21_b4std = GOOGL_21_b4['Adj Close'].std()

GOOGL_22_b4std = GOOGL_22_b4['Adj Close'].std()

#GOOGL 2013(Q4) After Earning Announcement
GOOGL_13_after = pdr.get_data_yahoo("GOOGL",start = "2014-01-27", end = "2014-02-03")
```

```
#GOOGL 2014(Q4) After Earning Announcement
GOOGL_14_after = pdr.get_data_yahoo("GOOGL",start = "2015-01-27", end = "2015-02-03")

#GOOGL 2015(Q4) After Earning Announcement
GOOGL_15_after = pdr.get_data_yahoo("GOOGL",start = "2016-01-26", end = "2016-02-04")

#GOOGL 2016(Q4) After Earning Announcement
GOOGL_16_after = pdr.get_data_yahoo("GOOGL",start = "2017-01-31", end = "2017-02-07")

#GOOGL 2017(Q4) After Earning Announcement
GOOGL_17_after = pdr.get_data_yahoo("GOOGL",start = "2018-02-01", end = "2018-02-08")

#GOOGL 2018(Q4) After Earning Announcement
GOOGL_18_after = pdr.get_data_yahoo("GOOGL",start = "2019-01-29", end = "2019-02-05")

#GOOGL 2019(Q4) After Earning Announcement
GOOGL_19_after = pdr.get_data_yahoo("GOOGL",start = "2020-01-28", end = "2020-02-02")

#GOOGL 2020(Q4) After Earning Announcement
GOOGL_20_after = pdr.get_data_yahoo("GOOGL",start = "2021-01-27", end = "2021-02-03")

#GOOGL 2021(Q4) After Earning Announcement
GOOGL_21_after = pdr.get_data_yahoo("GOOGL",start = "2022-01-27", end = "2022-02-03")

#GOOGL 2022(Q4) After Earning Announcement
GOOGL_22_after = pdr.get_data_yahoo("GOOGL",start = "2023-02-02", end = "2023-02-09")

GOOGL_13_aftermean = GOOGL_13_after['Adj Close'].mean()

GOOGL_14_aftermean = GOOGL_14_after['Adj Close'].mean()

GOOGL_15_aftermean = GOOGL_15_after['Adj Close'].mean()

GOOGL_16_aftermean = GOOGL_16_after['Adj Close'].mean()

GOOGL_17_aftermean = GOOGL_17_after['Adj Close'].mean()

GOOGL_18_aftermean = GOOGL_18_after['Adj Close'].mean()

GOOGL_19_aftermean = GOOGL_19_after['Adj Close'].mean()

GOOGL_20_aftermean = GOOGL_20_after['Adj Close'].mean()

GOOGL_21_aftermean = GOOGL_21_after['Adj Close'].mean()
```

```
GOOGL_22_aftermean = GOOGL_22_after['Adj Close'].mean()

GOOGL_13_afterstd = GOOGL_13_after['Adj Close'].std()

GOOGL_14_afterstd = GOOGL_14_after['Adj Close'].std()

GOOGL_15_afterstd = GOOGL_15_after['Adj Close'].std()

GOOGL_16_afterstd = GOOGL_16_after['Adj Close'].std()

GOOGL_17_afterstd = GOOGL_17_after['Adj Close'].std()

GOOGL_18_afterstd = GOOGL_18_after['Adj Close'].std()

GOOGL_19_afterstd = GOOGL_19_after['Adj Close'].std()

GOOGL_20_afterstd = GOOGL_20_after['Adj Close'].std()

GOOGL_21_afterstd = GOOGL_21_after['Adj Close'].std()

GOOGL_22_afterstd = GOOGL_22_after['Adj Close'].std()


GOOGL_13 = (GOOGL_13_aftermean - GOOGL_13_b4mean)/ GOOGL_13_b4mean

GOOGL_14 = (GOOGL_14_aftermean - GOOGL_14_b4mean)/ GOOGL_14_b4mean

GOOGL_15 = (GOOGL_15_aftermean - GOOGL_15_b4mean)/ GOOGL_15_b4mean

GOOGL_16 = (GOOGL_16_aftermean - GOOGL_16_b4mean)/ GOOGL_16_b4mean

GOOGL_17 = (GOOGL_17_aftermean - GOOGL_17_b4mean)/ GOOGL_17_b4mean

GOOGL_18 = (GOOGL_18_aftermean - GOOGL_18_b4mean)/ GOOGL_18_b4mean

GOOGL_19 = (GOOGL_19_aftermean - GOOGL_19_b4mean)/ GOOGL_19_b4mean

GOOGL_20 = (GOOGL_20_aftermean - GOOGL_20_b4mean)/ GOOGL_20_b4mean

GOOGL_21 = (GOOGL_21_aftermean - GOOGL_21_b4mean)/ GOOGL_21_b4mean

GOOGL_22 = (GOOGL_22_aftermean - GOOGL_22_b4mean)/ GOOGL_22_b4mean
```

```
GOOGL_13STD = (GOOGL_13_afterstd - GOOGL_13_b4std)/ GOOGL_13_b4std

GOOGL_14STD = (GOOGL_14_afterstd - GOOGL_14_b4std)/ GOOGL_14_b4std

GOOGL_15STD = (GOOGL_15_afterstd - GOOGL_15_b4std)/ GOOGL_15_b4std

GOOGL_16STD = (GOOGL_16_afterstd - GOOGL_16_b4std)/ GOOGL_16_b4std

GOOGL_17STD = (GOOGL_17_afterstd - GOOGL_17_b4std)/ GOOGL_17_b4std

GOOGL_18STD = (GOOGL_18_afterstd - GOOGL_18_b4std)/ GOOGL_18_b4std

GOOGL_19STD = (GOOGL_19_afterstd - GOOGL_19_b4std)/ GOOGL_19_b4std

GOOGL_20STD = (GOOGL_20_afterstd - GOOGL_20_b4std)/ GOOGL_20_b4std

GOOGL_21STD = (GOOGL_21_afterstd - GOOGL_21_b4std)/ GOOGL_21_b4std

GOOGL_22STD = (GOOGL_22_afterstd - GOOGL_22_b4std)/ GOOGL_22_b4std

Date_list = earning_table.index.array

Date_list

#################
GOOGL=[GOOGL_13STD, GOOGL_14STD, GOOGL_15STD, GOOGL_16STD,
GOOGL_17STD,
   GOOGL_18STD, GOOGL_19STD, GOOGL_20STD, GOOGL_21STD, GOOGL_22STD]

df_GOOGLSTD = pd.DataFrame(GOOGL,
                  index = earning_table.index,
                  columns = ['STD Diff Before & After Ann'])

df_GOOGLSTD
################

GOOGL_data = pd.merge(earning_table, pd.concat([df_GOOGL, df_GOOGLSTD], axis=1),
left_index=True, right_index=True)
GOOGL_data
```

Dep. Variable:	Price Diff Before & After Ann	**R-squared:**	0.185
Model:	OLS	**Adj. R-squared:**	0.156
Method:	Least Squares	**F-statistic:**	6.343
Date:	Thu, 16 Mar 2023	**Prob (F-statistic):**	0.0178
Time:	18:37:19	**Log-Likelihood:**	51.197
No. Observations:	30	**AIC:**	-98.39
Df Residuals:	28	**BIC:**	-95.59
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.0145	0.010	1.456	0.157	-0.006	0.035
Actual - Expected EPS	-0.3164	0.126	-2.518	0.018	-0.574	-0.059

Omnibus:	2.458	**Durbin-Watson:**	1.788
Prob(Omnibus):	0.293	**Jarque-Bera (JB):**	1.194
Skew:	0.337	**Prob(JB):**	0.550
Kurtosis:	3.708	**Cond. No.**	15.2