Worksheet 8

You are encouraged to work in groups of up to 3 total students, but each student should make their own submission on Canvas. (It's fine for everyone in the group to have the same upload.)

```
In [1]: import pandas as pd
        import altair as alt
```

Overview

This project includes a csv file `unemployment.csv` which includes US unemployment rates for various industries between 2000 and 2010.

- Load that csv file using `pd.read_csv` and save the resulting DataFrame as `df`.

The background question for this homework is,

> How does the time of year affect unemployment in different industries?

```
In [2]: df = pd.read_csv("../Data/unemployment.csv")
        df.head()
```

Out[2]:

	industry	date	rate
0	Government	2000-01-01 08:00:00+00:00	2.1
1	Government	2000-02-01 08:00:00+00:00	2.0
2	Government	2000-03-01 08:00:00+00:00	1.5
3	Government	2000-04-01 08:00:00+00:00	1.3
4	Government	2000-05-01 07:00:00+00:00	1.9

Cleaning the data

- Evaluate `df.dtypes`. Notice that this "date" column is not being recognized as containing datetime values.

```
In [3]: df.dtypes
```

```
Out[3]: industry     object
        date         object
        rate        float64
        dtype: object
```

- Convert this column to datetime format using `pd.to_datetime`.

```
In [4]: df["date"] = pd.to_datetime(df["date"])
```

- Evaluate `df.dtypes` and `df.dtypes["date"]` to make sure the change has actually occurred within `df`.

```
In [5]: print(df.dtypes)
        print(df.dtypes["date"])
```

```
industry                  object
date         datetime64[ns, UTC]
rate                     float64
dtype: object
datetime64[ns, UTC]
```

Less good strategy, just for practice with `map` and lambda functions.

- Using the pandas Series method `map` and a lambda function, make a new column `"month0"` which contains the numerical month for each date. (For example, `4` if the date is in April.)

Side question: why don't you need to use the `dt` accessor here?

Since x is already a datetime object, we can just access the month directly

```
In [6]: df["month0"] = df["date"].map(lambda x: x.month)
        df["month0"]
```

```
Out[6]: 0        1
        1        2
        2        3
        3        4
        4        5
                ..
        1703    10
        1704    11
        1705    12
        1706     1
        1707     2
        Name: month0, Length: 1708, dtype: int64
```

Better strategy.

- Make a new column `"month"` in the DataFrame which contains the numerical month for each date. Use the `dt` accessor but not `map`.

In [7]:
```
df["month"] = df["date"].dt.month
df["month"]
```

Out[7]:
```
0        1
1        2
2        3
3        4
4        5
        ..
1703    10
1704    11
1705    12
1706     1
1707     2
Name: month, Length: 1708, dtype: int32
```

- Verify that the "month" and "month0" columns contain the same values. First create a Boolean Series, then call the `all` method.

In [8]:
```
print(any(df["month"] != df["month0"]))
```

```
False
```

- Evaluate the following. What is it telling us about the presence of missing values in `df` ?

  ```
  df.isna().any(axis=1).any()
  ```

In [9]:
```
df.isna().any(axis=1).any()
```

Out[9]:
```
np.False_
```

This is telling us there are no missing values along the columns (axis = 1) of df

Normalizing the data

- Make a pandas Series `mean_ser` containing the average unemployment rate for each industry, using the following code.

  ```
  mean_ser = df.groupby("industry")["rate"].mean()
  ```

In [10]:
```
mean_ser = df.groupby("industry")["rate"].mean()
```

- Make the analogous pandas Series for standard deviation, and name it `std_ser` .

In [11]:
```
std_ser = df.groupby("industry")["rate"].std()
```

- Write a function `make_norm` which takes as input a row of `df` (not a row label but the whole row as a pandas Series) and as output returns the normalized unemployment rate, where by "normalized", mean that you should subtract the mean for that industry and divide by the standard deviation for that industry.

For example, if `rate` is `7`, the `mean` for the industry is `1.2` and the standard deviation for the industry is `4.3`, then the function should return `(7 - 1.2)/4.3`.

In [12]:
```python
def make_norm(row):
    return (row["rate"] - mean_ser[row["industry"]]) / std_ser[row["industry"]]
```

- Using `apply`, the above function `make_norm`, and a suitable `axis` argument, for each row in `df`, normalize the unemployment rate (so that the mean becomes 1 and the standard deviation becomes 1). We need to use `apply` and not `map` here, because we need to know the industry. Put the result in a new column in `df` called `"norm_rate"`.

In [13]:
```python
df.head()
```

Out[13]:

	industry	date	rate	month0	month
0	Government	2000-01-01 08:00:00+00:00	2.1	1	1
1	Government	2000-02-01 08:00:00+00:00	2.0	2	2
2	Government	2000-03-01 08:00:00+00:00	1.5	3	3
3	Government	2000-04-01 08:00:00+00:00	1.3	4	4
4	Government	2000-05-01 07:00:00+00:00	1.9	5	5

In [14]:
```python
df["norm_rate"] = df.apply(make_norm, axis=1)
```

- Using `groupby`, check that the means for the various industries of this new `"norm_rate"` column are all very close to 0, and the standard deviations are all close to 1.

In [15]:
```python
# Take the mean of the rates by industry, then take the mean of the means to just g
# Sae thing with the std
print(df.groupby("industry")["norm_rate"].mean().round())
print(df.groupby("industry")["norm_rate"].std().round())

print(f'Mean of the means: {df.groupby("industry")["norm_rate"].mean().mean().round
print(f'Mean of the stds:  {df.groupby("industry")["norm_rate"].std().mean().round(
industry
Agriculture                     0.0
Business services              -0.0
Construction                   -0.0
Education and Health           -0.0
Finance                         0.0
Government                      0.0
Information                    -0.0
Leisure and hospitality         0.0
Manufacturing                   0.0
Mining and Extraction           0.0
Other                           0.0
Self-employed                   0.0
Transportation and Utilities   -0.0
Wholesale and Retail Trade      0.0
Name: norm_rate, dtype: float64
industry
Agriculture                     1.0
Business services               1.0
Construction                    1.0
Education and Health            1.0
Finance                         1.0
Government                      1.0
Information                     1.0
Leisure and hospitality         1.0
Manufacturing                   1.0
Mining and Extraction           1.0
Other                           1.0
Self-employed                   1.0
Transportation and Utilities    1.0
Wholesale and Retail Trade      1.0
Name: norm_rate, dtype: float64
Mean of the means: 0.0
Mean of the stds:  1.0
```

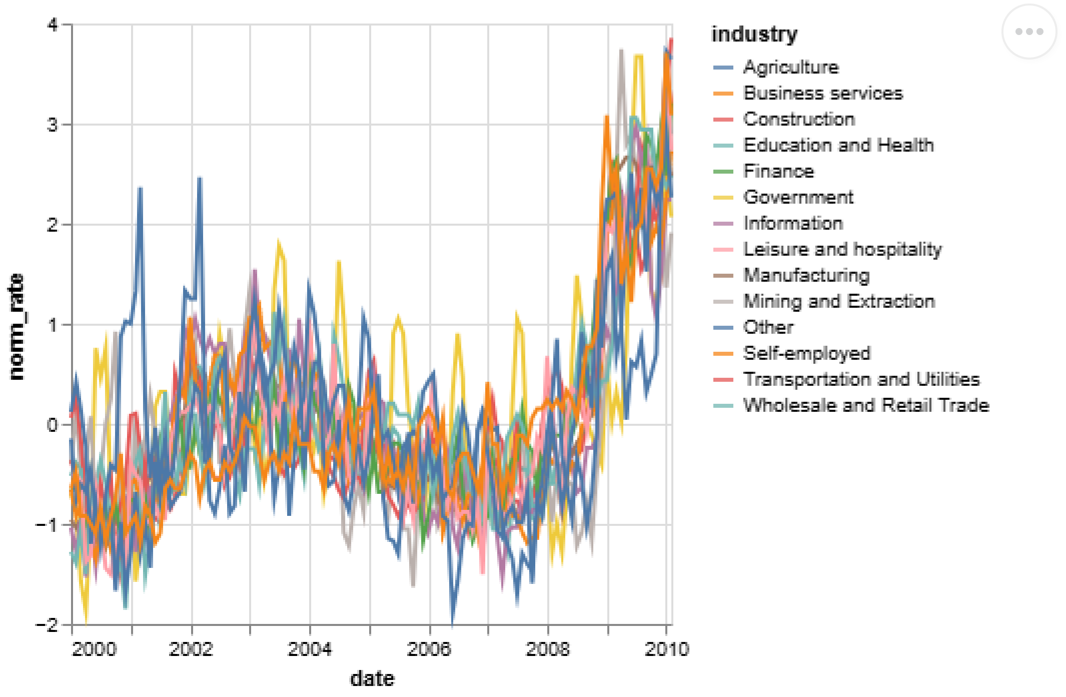
Plotting the data

- Make an Altair chart of this data, using `mark_line`, using "date" for the x-channel, using "norm_rate" for the y-channel, and using "industry" for the color.

Side question: can you recognize the impact of the 2008 financial crisis?

```
In [16]: alt.Chart(df).mark_line().encode(x = "date", y = "norm_rate", color = "industry")
```

Out[16]:



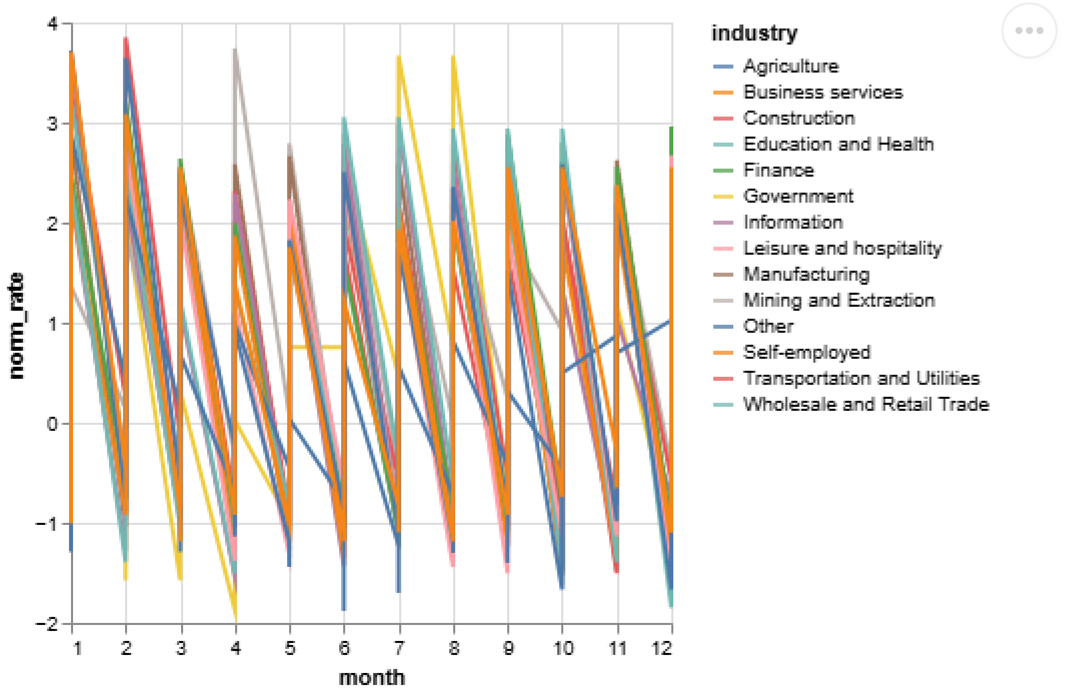
We can see a large spike in the unemployment rate after 2008

- Make the same Altair chart, but change from "date" to "month" for the x-channel.

(It will look like a mess because each industry has the same month repeated many times, corresponding to different years.)

In [17]:
```
alt.Chart(df).mark_line().encode(x = "month", y = "norm_rate", color = "industry")
```

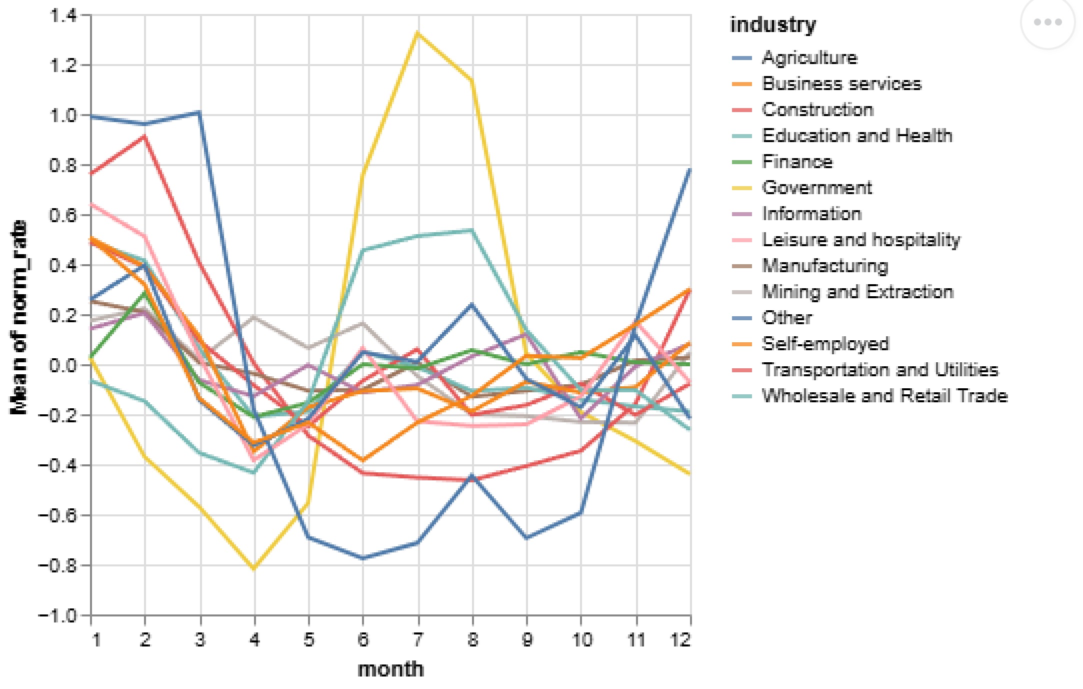
Out[17]:



- Make the same chart as in the previous cell, but change from `y="norm_rate"` to `y="mean(norm_rate)"`. This will replace the many-points-per-industry with a single point per industry. Store the chart with the variable name `c`, and display this chart.

```
In [18]: c = alt.Chart(df).mark_line().encode(x = "month", y = "mean(norm_rate)", color = "i
         c
```

Out[18]:



- Notice that "Government" appears to have the highest average normalized unemployment in July. Using Boolean indexing and `mean`, compute this average directly using pandas, and make sure it matches the value you see in Altair. (It should be approximately 1.3.)

```
In [19]: df[(df["month"] == 7) & (df["industry"] == "Government")]["norm_rate"].mean()

Out[19]: np.float64(1.3234115570133693)
```

Submission

- Save the chart as a json file using the following code and upload that json file to Canvas.

```
with open("chart.json", "w") as f:
    f.write(c.to_json())
```