DATA 228 BIG DATA

# Job Rate Analysis on US Market Influenced by Covid-19

**Big Cloud**
Yixin Huang
Chloe Ngo
Minh-Tam Pham
SeungMin Yoo

## Overview

Unemployment is generally regarded as an important indicator of the economy, which is currently a topic of great concern since Covid-19 led to a peak unemployment rate of 14.7% in April [1]. Our project's primary goal is to research and analyze the labor market's efforts to return to "normal", pre-Covid times and to predict when to return to pre- Covid19. Some questions of interest include: how are various industries' hiring patterns impacted, and which states are doing better or worse?

Our team will hope to explore this problem and analyze these questions by building an application that showcases visualizations and analyses done by the team. The insights provided by the application aim to help aid in decision-making (at the private and public level), general research, or for educational purposes.

This report will outline the application building process, featuring cloud technologies, and relay insights derived from the datasets.

## About the Datasets

Our team uses datasets provided by greenwich.hr that measure the rate (relative to pre-Covid times) of new, unique job postings. We have data sets that are segmented in 3 different ways: an overall dataset, that measures the overall new job rates (unsegmented), industries, which segments the new job rates by job industry, and geography, which segments the new job rates by state [2]. We also use a dataset that measures the covid mortality rates [4].

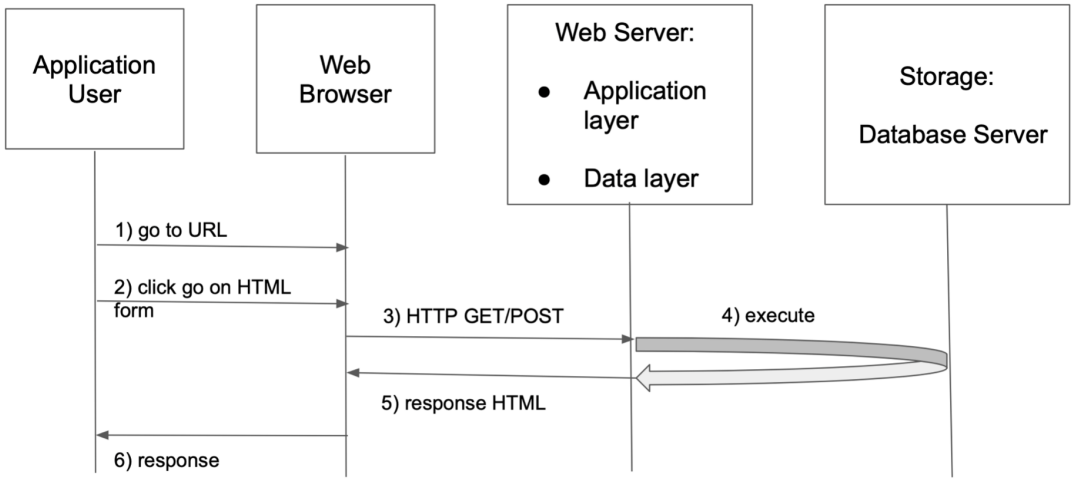
Additionally we predict the pattern of new cases of the Covid-19 with the datasets and statistical models and compare our results with experts' opinion to evaluate our models.

## Application Structure

### *1) Full Stack: HTML- Python - FLASK- MySQL*

Our team developed a web application that can demonstrate our data analysis, by interacting between among the front-end, middle-tier server, and back-end database:

- On the front-side, we chose HTML as the language to communicate through HTTP requests.
- On the server-side, we used python packages FLASK, Jinja2 to run the server and respond to the HTTP requests.
- For the back-end databases, we chose MySQL PHP myadmin as our database host since it is cost efficient (as long as the data is under a certain limited storage) and is easy to access to our front-end by using flask.

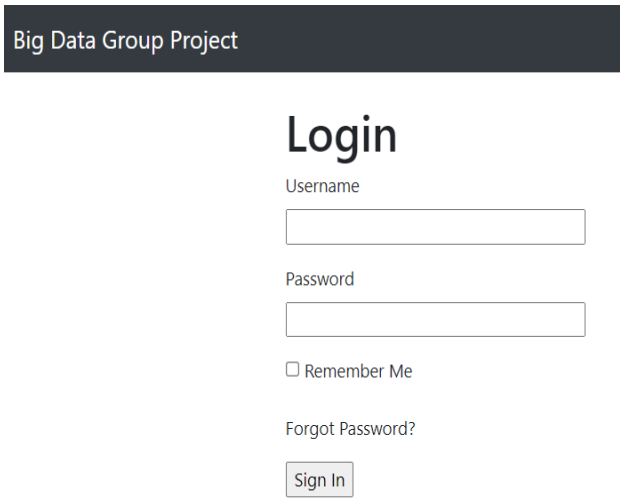



### *2) Front-end: HTML*

In order to create a user-friendly interface, we used HTML as the functional foundation of the front-end with an associative framework Bootstrap. For our web, we built a login page, a home page, and three analytical pages sharing jupyter analytics and tableau visualizations.

- *Login Page*

The ‘login’ page contains two text fields and one button. We are requesting the user for their credentials to verify that they are allowed to access the page. Hitting the button ‘Sign in’ will send their data to a web server and compare the user information with the data stored in our back-end database, a user table. If the personal information matches, the user will be allowed to access the home page, otherwise, the login page will demonstrate an error and request that the user to type in email and password again.




```
{% extends "layout.html" %}

{% block content %}
<div class="col-sm-2"></div>
    <div class="col-md-8">
<div class="center-block">
        <h1>Login</h1>
        <form action="" method="post" novalidate>
          <div class='form-group'>
              {{ form.csrf_token }}
              {{ form.hidden_tag() }}
              <p>
                  {{ form.username.label }}<br>
                  {{ form.username(size=32) }}
              </p>
          </div>
          <div class='form_group'>
              <p>
                  {{ form.password.label }}<br>
                  {{ form.password(size=32) }}
              </p>
              <p>{{ form.remember_me() }} {{ form.remember_me.label }}</p>
              <p id="forgotPW"> Forgot Password? </p>
              <p>{{ form.submit() }}</p>
```

- *Home Page*

The home page contains four buttons at the top of the page, which directs to other pages we designed. We used the HTML elements, such as <form>, <style>, <textarea>, <button> to format

the pages. All pages are decorated by extending to a file called ’layout.html’ which is a general html file to style all the pages’ format [4].


```
templates > Login.html > ...
1   {% extends "layout.html" %}
2
3   {% block content %}
4   <div class="col-sm-2"></div>
5       <div class="col-md-8">
6   <div class="center-block">
7           <h1>Login</h1>
8           <form action="" method="post" novalidate>
9             <div class='form-group'>
10                {{ form.csrf_token }}
11                {{ form.hidden_tag() }}
12                <p>
13                    {{ form.username.label }}<br>
14                    {{ form.username(size=32) }}
15                </p>
16            </div>
17            <div class='form_group'>
18                <p>
19                    {{ form.password.label }}<br>
20                    {{ form.password(size=32) }}
21                </p>
22                <p>{{ form.remember_me() }} {{ form.remember_me.label }}</p>
23                <p id="forgotPW"> Forgot Password? </p>
24                <p>{{ form.submit() }}</p>
```

### *3) Middle-tier: Python Flask*

In the main project folder, we created a file called app.py, which is the backend for our web application [4].

The Flask package is what we utilized to initiate the application, while the render_template package is used to connect the HTML we wrote to the web application [4].

- Create the app and store in a variable called app, app = Flask(__name__)
- Create route, which is a URL pattern. Our base URL is assumed to be the login page. And a decorator is used above the initialize function.
- Create a ‘session’ function to check if the user has logged in or not. Without this function, the user can go directly to the home page by just typing the URL address.
- For the login function, firstly it requests the users' email and password. Then it creates a cursor connected to our MySQL database. We request the user's email address and password, and compare the user information with the data stored in our database. If this personal information matches, the user will be allowed to access the home page, otherwise, the login page will demonstrate an error and redirect back to the login page.
- Create another four functions for the home page, python page, Tableau page and prediction page.

The code is shown below and can also be accessed in the github [4].

```python
import sys
import logging
import datetime
import numpy as np
from flask import Flask, request, jsonify, render_template, flash, redirect, session, url_for
from flaskext.mysql import MySQL
from urllib.parse import urlsplit
from login import LoginForm
from functools import wraps




secret_key = "secret123"

# create the app
app = Flask(__name__)
app.config['SECRET_KEY'] = secret_key
mysql = MySQL(app)


# connect to database
app.config['MYSQL_DATABASE_USER'] = 'sql3411464'
app.config['MYSQL_DATABASE_PASSWORD'] = 'ZVfm84xHeJ'
app.config['MYSQL_DATABASE_DB'] = 'sql3411464'
app.config['MYSQL_DATABASE_HOST'] = 'sql3.freemysqlhosting.net'

mysql.init_app(app)

# default route
@app.route('/')
def initlialize_app():
    return redirect('/login')
```

```python
# Check if user logged in
def is_logged_in(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        if session['logged_in']:
            return f(*args, **kwargs)
        else:
            flash('Unauthorized, please login first', 'danger')
            return redirect('/login')

    return wrap



# Login Page
@app.route('/login', methods=['GET', 'POST'])
def login():
    session['logged_in'] = False
    login = LoginForm()
    if request.method == 'POST':
        if login.validate_on_submit():
            # Get Form Fields
            email = login.username.data
            password_candidate = login.password.data
            # Create Cursor
            cur = mysql.get_db().cursor()
            # Get user by name
            result = cur.execute ("SELECT * FROM User WHERE email = '%s'" % email)
            if result > 0:
                # Get stored pin
                data = cur.fetchone()
                password = data[2]
                # Compare Passwords
                if password_candidate == password:
                    # Passed
                    session['logged_in'] = True
```

### *4) Back-end: MySQL database*

The sqlalchemy engine creates a dialect object tailored towards MySQL, as well as a pool object which will establish a DBAPI connection, MySQL-connector-python, at localhost when a connection request is first received.

The ‘create engine’ function of sqlalchemy establishes a connection to our MySQL database with server address, database name, and database password. We are now prepared to implement SQL queries to extract and manipulate data.

```
# Import df3_merge into MySQL database

# Use sqlalchemy to create the connection
database_username = 'sql13411464'
database_password = 'ZVfm84xHeJ'
database_ip       = 'sql3.freemysqlhosting.net'
database_name     = 'sql13411464'
database_connection = sqlalchemy.create_engine('mysql+mysqlconnector://{0}:{1}@{2}/{3}'.
                                               format(database_username, database_password,
                                                      database_ip, database_name))
```

### *1) ETL*

ETL is a process to extract the data from multiple sources, transform the data, and finally load the data into databases. In order to detect the change of job rates influenced by Covid-19, ETL should be a recurring activity of our database. We opt to update this information in batches, such as every month.

For our data, we extracted the data from two different data sources, and most of the data are CSV format. From observing the original data, we found that there are considerable amounts of NaN values. We proceed to investigate the meaning of these NaN values, to handle them accordingly: can we replace them with zeros or delete the data entirely? For the Covid-19 deaths table, we read that NaN means there weren't any new cases or deaths that day. Thusly, replacing the NaN values with 0 will be appropriate.

```
df_covid19_death_us.head()
```

| | iso_code | continent | location | date | total_cases | new_cases | new_cases_smoothed | total_deaths | new_deaths | new_deaths_smoothed | total_cases_per_m |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | USA | North America | United States | 2020/1/22 | 1 | NaN | NaN | NaN | NaN | NaN | |
| 1 | USA | North America | United States | 2020/1/23 | 1 | 0.0 | NaN | NaN | NaN | NaN | |
| 2 | USA | North America | United States | 2020/1/24 | 2 | 1.0 | NaN | NaN | NaN | NaN | |
| 3 | USA | North America | United States | 2020/1/25 | 2 | 0.0 | NaN | NaN | NaN | NaN | |
| 4 | USA | North America | United States | 2020/1/26 | 5 | 3.0 | NaN | NaN | NaN | NaN | |

We observed another problem when doing the ETL process: the date columns were inconsistent in format (some by year-month-day, some by year/month/day), and the original format of date is

string. Since our data is time-oriented, we want this format to be ‘date_time’ to perform meaningful analysis. It will enable statisticians to perform time series analysis.  So we use the 'to_datetime' function in pandas to change all the string format dates into a datetime format.

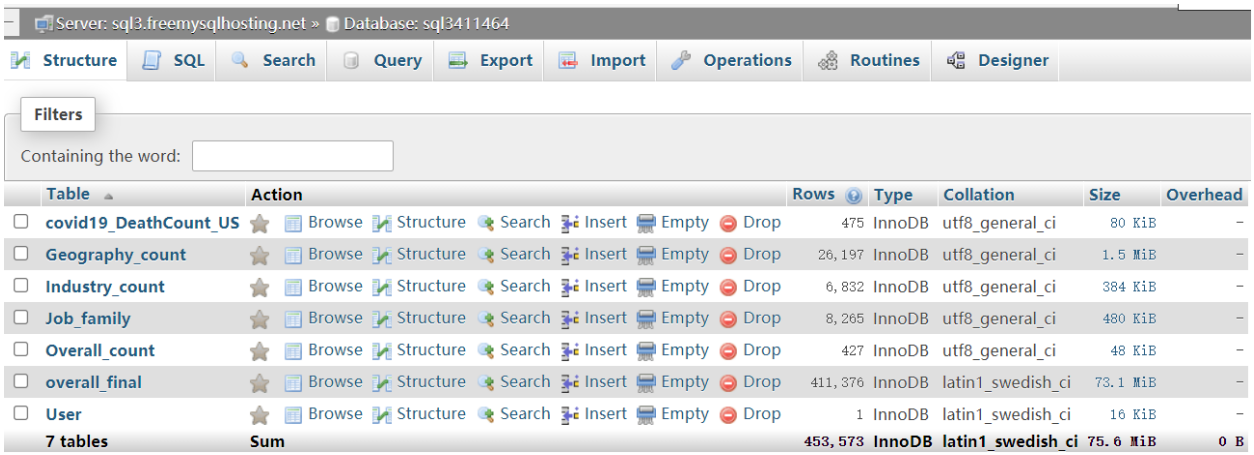
| | iso_code | continent | location | date |
|---|---|---|---|---|
| 0 | USA | North America | United States | 2020/1/22 |
| 1 | USA | North America | United States | 2020/1/23 |
| 2 | USA | North America | United States | 2020/1/24 |
| 3 | USA | North America | United States | 2020/1/25 |
| 4 | USA | North America | United States | 2020/1/26 |

| | iso_code | continent | location | date |
|---|---|---|---|---|
| 0 | USA | North America | United States | 2020-01-22 |
| 1 | USA | North America | United States | 2020-01-23 |
| 2 | USA | North America | United States | 2020-01-24 |
| 3 | USA | North America | United States | 2020-01-25 |
| 4 | USA | North America | United States | 2020-01-26 |

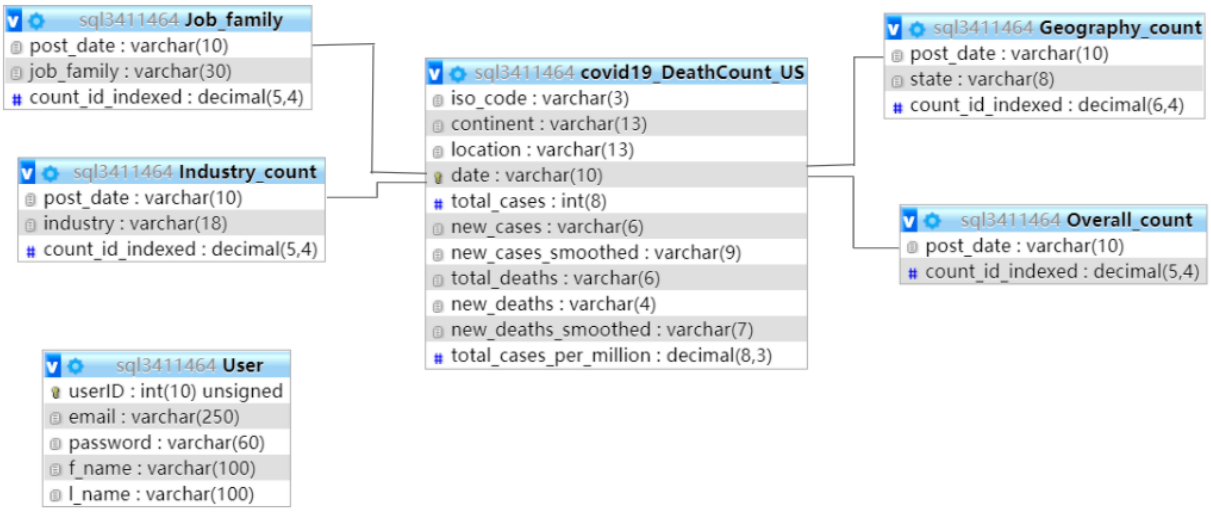
```
# change 'date' format from 2020/1/22 to 2020-01-22

df_covid19_death_us['date'] = pd.to_datetime(df_covid19_death_us['date'],format='%Y-%m-%d')
df_overall['post_date'] = pd.to_datetime(df_overall['post_date'],format='%Y-%m-%d')
df_geo['post_date'] = pd.to_datetime(df_geo['post_date'],format='%Y-%m-%d')
df_ind['post_date'] = pd.to_datetime(df_ind['post_date'],format='%Y-%m-%d')
```

### *2) Data modeling*

Based on the data loaded into MySQL database, we defined the relational tables and keys by using the ER diagram. There are 6  tables, 27 columns, 420,000 rows in all.

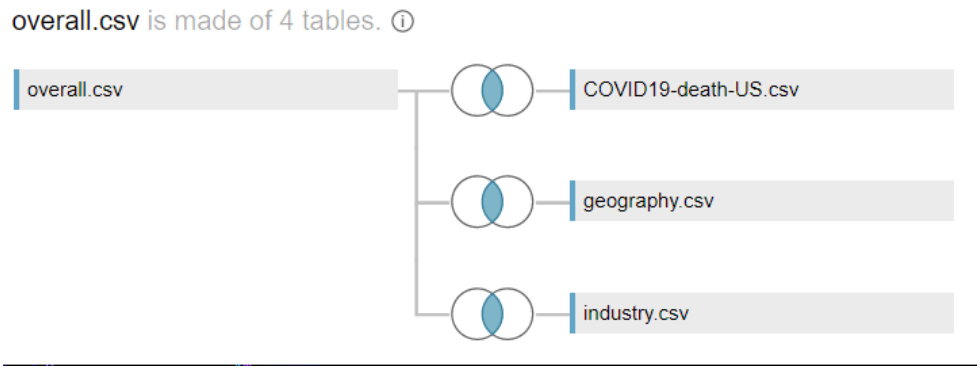


- Entity: Covid19_DeathCount_US, Overall_count, Geography_count, Job_family, Industry_count, User
- Attribute: date is the primary key of Covid19_DeathCount_US, post_date is the primary key of Overall_count, post_date is the primary key of Geography_count, post_date is the primary key of Job_family, post_date is the primary key of Industry_count. User table is a separated table to store all the information of users.




### *3) Join Method*

After checking the data structure and cleansing the datasets, we picked the inner join method to link those tables. As long as the keywords match, the joined table will combine all the rows from original tables.




#### *4) Single-sign-on (SSO) authentication*

The single-sign-on function is an authentication feature to validate users' credentials and establish the identity of the users, then share the information with the subsystems that require the data. We developed the application so that it requires the user to log in at domain http://127.0.0.1:5000/', and allows the same user to have access to other domains without logging in again. Our way of SSO is to create a central domain and then to share the session information across different subdomains.

## Big Data Analytics and Visualization

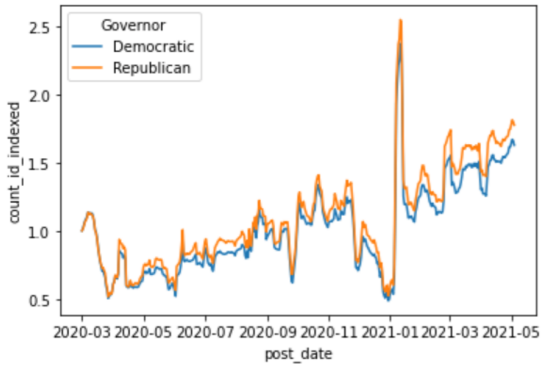
#### *1) Jupyter Analytics*

We wanted to run some exploratory analysis using Python and Jupyter notebook along with segmenting the data by political party affiliation. The code for this portion can be viewed in the analysis file in github [5]. Based on our results we can conclude the following:

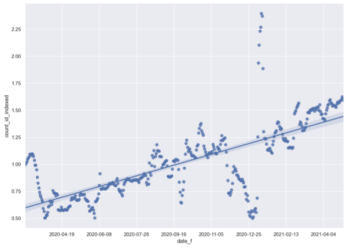
Republican states (defined by the political party of their governor) have a higher average rate of new job listings compared to Democratic states.

```
In [14]: #plot comparing new job listing averages of Democratic and Republican states.

         import seaborn as sns
         sns.lineplot(x='post_date', y='count_id_indexed', hue='Governor', data=groups)
```

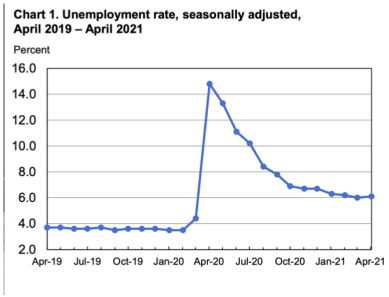
```
Out[14]: <AxesSubplot:xlabel='post_date', ylabel='count_id_indexed'>
```




There is an overall linear trend of increasing new job rates (higher than pre-covid times) and we postulate that this increasing rate of new jobs is to close the huge gap in unemployment that stemmed from Covid.



Linear regression of new unique job postings over time (03/2020-5/20201) done using python (seaborn package)




Unemployment rates from bls.gov.

- The first conclusion took extra steps, specifically when creating a new table that identifies the States' parties. We first created a data frame of states and their parties and we opted to web scrape from Wikipedia, as shown in the image below [5].

7830 rows × 3 columns

In [3]:
```
url = "https://en.wikipedia.org/wiki/Political_party_strength_in_U.S._states"

page = urllib.request.urlopen(url)
soup = BeautifulSoup(page, "lxml")
all_tables = soup.find_all("table")
right_table = soup.find_all('table', class_ = "sortable wikitable")
```

In [4]:
```
A = []
B = []
C = []
D = []
E = []
F = []
G = []
H = []
I = []

for row in right_table[0].findAll('tr'):
    cells = row.findAll('td')
    if len(cells) == 9:
        A.append(cells[0].find(text = True))
        B.append(cells[1].find(text = True))
        C.append(cells[2].find(text = True))
        D.append(cells[3].find(text = True))
        E.append(cells[4].find(text = True))
        F.append(cells[5].find(text = True))
        G.append(cells[6].find(text = True))
        H.append(cells[7].find(text = True))
        I.append(cells[8].find(text = True))
    elif len(cells) == 8:
        A.append(cells[0].find(text = True))
        B.append(cells[1].find(text = True))
        C.append(cells[2].find(text = True))
        D.append(cells[3].find(text = True))
        E.append(cells[3].find(text = True))
        F.append(cells[4].find(text = True))
        G.append(cells[5].find(text = True))
        H.append(cells[6].find(text = True))
        I.append(cells[7].find(text = True))
```

In [5]:
```
df = pd.DataFrame(A, columns = ['State'])
df['2020 presidential election'] = list(map(str.strip, B))
df['Governor'] = list(map(str.strip, C))
df['State Senate'] = list(map(str.strip, D))
df['State House'] = list(map(str.strip, E))
df['Senior U.S. Senator'] = list(map(str.strip, F))
df['Junior U.S. Senator'] = list(map(str.strip, G))
df['U.S. House of Reprsentatives'] = list(map(str.strip, H))
df['party registration'] = list(map(str.strip, I))
```

In [6]:
```
states = ["AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA",
          "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD",
          "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ",
          "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC",
          "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"]

df['State Abbreviation'] = states
```

In [7]:
```
df.head()
```

Out[7]:

| | State | 2020 presidential election | Governor | State Senate | State House | Senior U.S. Senator | Junior U.S. Senator | U.S. House of Reprsentatives | party registration | State Abbreviation |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Alabama | Republican | Republican | Republican 27–8 | Republican 77–28 | Republican | Republican | Republican 6–1 | Republican | AL |
| 1 | Alaska | Republican | Republican | Republican 13–7 | Coalition 23–17 | Republican | Republican | Republican | Republican | AK |
| 2 | Arizona | Democratic | Republican | Republican 16–14 | Republican 31–29 | Democratic | Democratic | Democratic 5–4 | Republican | AZ |
| 3 | Arkansas | Republican | Republican | Republican 28–7 | Republican 77–23 | Republican | Republican | Republican 4 | Republican | AR |

After obtaining the data frame of State parties, we joined it with our geography table, as shown below in line 12. We then took the daily average of the data, grouping by the political parties to obtain a chart that compared the two parties (as opposed to states).

In [8]:
```
state_party = df[['State Abbreviation', 'Governor']]
state_party.head()
```

Out[8]:

| | State Abbreviation | Governor |
|---|---|---|
| 0 | AL | Republican |
| 1 | AK | Republican |
| 2 | AZ | Republican |
| 3 | AR | Republican |
| 4 | CA | Democratic |

In [9]:
```
geo.head()
```

Out[9]:

| | post_date | state | count_id_indexed |
|---|---|---|---|
| 0 | 2020-03-01 | NaN | 1.0 |
| 1 | 2020-03-01 | AK | 1.0 |
| 2 | 2020-03-01 | AL | 1.0 |
| 3 | 2020-03-01 | AR | 1.0 |
| 4 | 2020-03-01 | AZ | 1.0 |

In [10]:
```
geo_party = state_party.merge(geo, right_on = 'state', left_on = 'State Abbreviation')
```

In [11]:
```
clean_geo = geo_party.drop(columns = ['State Abbreviation'])
```

In [12]:
```
clean_geo.head()
```

Out[12]:

| | Governor | post_date | state | count_id_indexed |
|---|---|---|---|---|
| 0 | Republican | 2020-03-01 | AL | 1.0000 |
| 1 | Republican | 2020-03-02 | AL | 1.0065 |
| 2 | Republican | 2020-03-03 | AL | 1.0451 |
| 3 | Republican | 2020-03-04 | AL | 1.0885 |
| 4 | Republican | 2020-03-05 | AL | 1.0865 |

In [13]:
```
#taking the average count_id_indexed for democratic and republican states
groups = clean_geo.drop(columns=['state']).groupby(['post_date', 'Governor']).mean()
groups.head()
```

Out[13]:

| | | count_id_indexed |
|---|---|---|
| post_date | Governor | |
| 2020-03-01 | Democratic | 1.000000 |
| | Republican | 1.000000 |
| 2020-03-02 | Democratic | 1.017613 |
| | Republican | 1.012985 |
| 2020-03-03 | Democratic | 1.043513 |

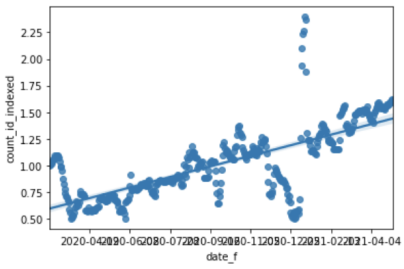
- The second conclusion was reached by implementing the following codes, where we ran a linear regression model against the data. The outliers, the massive drop and spike, happened around the holiday season following before the winter holidays, and after New Years. We conclude that companies typically do not look to hire around this time.

date

In [16]:
```
%matplotlib inline
temp = result2.sort_values('date')
temp['date_f'] = pd.factorize(temp['date'])[0] + 1
mapping = dict(zip(temp['date_f'], temp['date'].dt.date))

ax = sns.regplot('date_f', 'count_id_indexed', data=temp)
labels = pd.Series(ax.get_xticks()).map(mapping).fillna('')
ax.set_xticklabels(labels)
```

```
/Users/minh-tampham/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the followi
ng variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
<ipython-input-16-ad79872ffb42>:8: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels(labels)
```

Out[16]:
```
[Text(0.0, 0, ''),
 Text(50.0, 0, '2020-04-19'),
 Text(100.0, 0, '2020-06-08'),
 Text(150.0, 0, '2020-07-28'),
 Text(200.0, 0, '2020-09-16'),
 Text(250.0, 0, '2020-11-05'),
 Text(300.0, 0, '2020-12-25'),
 Text(350.0, 0, '2021-02-13'),
 Text(400.0, 0, '2021-04-04'),
 Text(450.0, 0, '')]
```

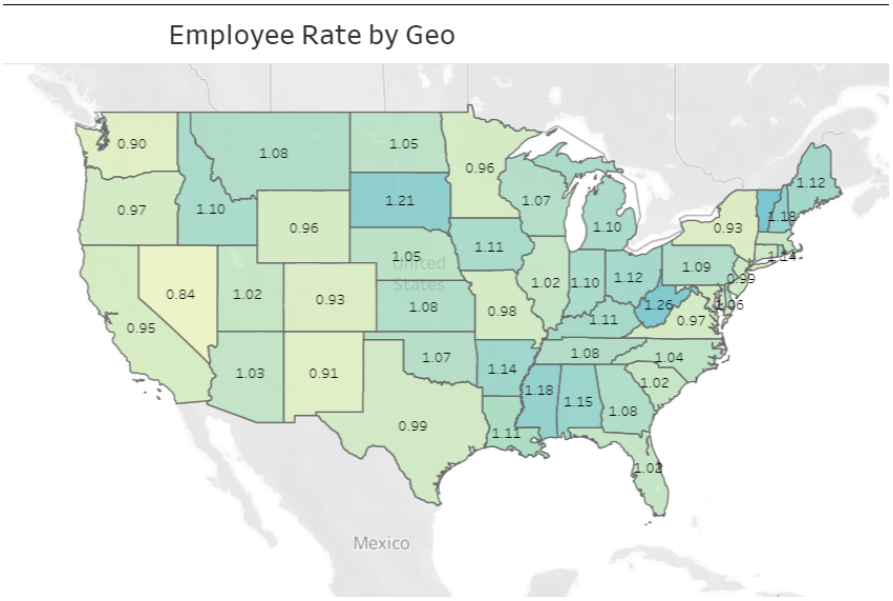



We notice that there is an overall linearly positive trend, except for the drop in January. We see that this rapid drop happens during the week before December 25 and 1 week after New Years.

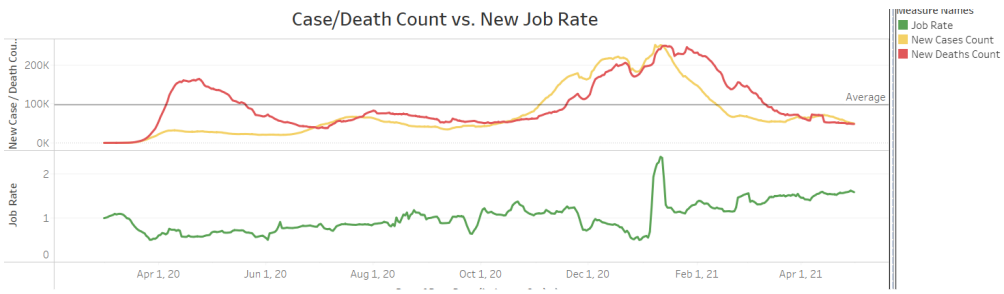
### *2) Tableau Visualization*

We created a Tableau Extension that integrates the web application written in Flask. Based on the visualization graphs, we get three conclusions.

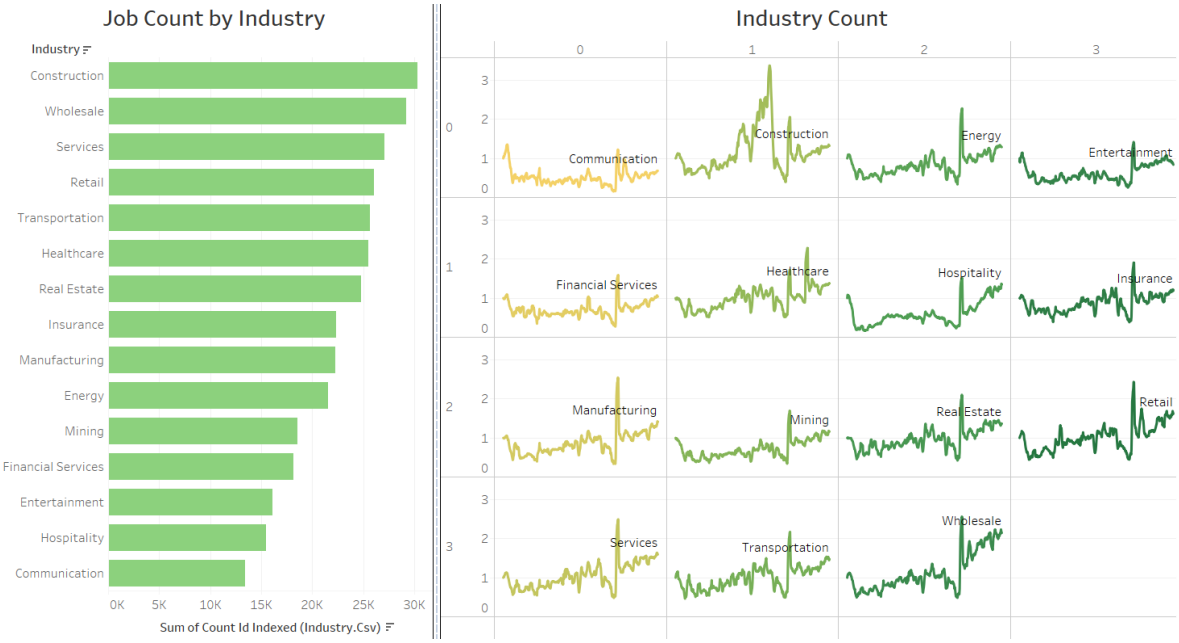
- The first observation is more jobs are released in the middle US since the job rates in that region are the highest among the whole country. Especially at WV the job rate is 1.26 by average from March 2020 to May 2021. To our surprise, CA and LA and other west states did not show a good employment trend with the job rate at 0.97 and 0.95. Additionally, NV state has the lowest rate at 0.84, we guess the reason for such a slumped number is that the state's primary attraction is tourism, which we know has been greatly affected Covid-19.




- The second observation is that the new job rate is related to Covid-19 cases this year, but not in 2020. In May 2020, the death rate increased sharply with no obvious new cases. At the beginning of 2021, the new case count increased and dropped together with the death count. Surprisingly, the new job count is influenced by Covi-19 at the breakout of 2021, but not changed a lot last year. Mostly because the job rate is a slag indicator of the economy, it will not change immediately with the shortage of supply or demand in the market.



- The third observation is that Covid-19 has more effect on certain industries(construction, wholesale). The left bar chart shows the sum of  job count by industry, construction has the most new job count in the latest one year, followed by wholesale and services. From the right line charts, we can see that the increasing trend of the construction industry is the most obvious. We guess the main reason is the policy to stimulate basic construction in the US. While the entertainment industry shows the weakest trend the whole year, which is definitely influenced by Covid-19. Fewer people went to the cinema or restaurants.




### *3) Prediction*

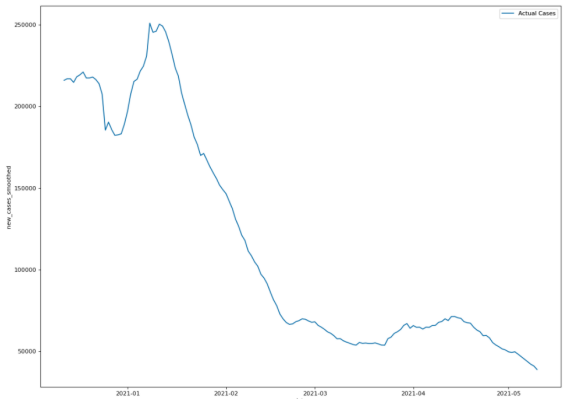
We trained several models and tested it with the Covid-19 CSV files, however, the accuracy of every model was less than fifty percent. Because the '.csv' files that we selected do not contain the information about vaccination, the most important feature.  So, without this information, any

machine learning algorithms were not able to make accurate predictions. Rather than finding and inserting a new dataset about vaccination, we decided to extract from the existing dataset. The extracted dataset starts December 11, 2020 which is the date of vaccine approved from FDA(Food and Drug Administration). When extracting a dataset in this way, the characteristics of the most important feature, vaccination, is reflected in the model to some extent.

```
#cleaning data and merging
covid = pd.read_csv("COVID19-death-US.csv", parse_dates = ['date'])
overall = pd.read_csv('overall.csv', parse_dates = ['post_date'])
result = covid.merge(overall, how='left', left_on='date', right_on='post_date')
result2 = covid.merge(overall, left_on='date', right_on='post_date')
industry = pd.read_csv('industry.csv', parse_dates = ['post_date'],encoding='ISO-8859-1')
jobfam = pd.read_csv('job_family.csv', parse_dates = ['post_date'],encoding='ISO-8859-1')
geo = pd.read_csv('geography.csv', parse_dates = ['post_date'],encoding='ISO-8859-1')
```

```
result_after =result[result["date"]>'2020-12-11']
train=result_after
```

| | iso_code | continent | location | date | total_cases | new_cases | new_cases_smoothed | total_deaths | new_deaths | new_deaths_smoothed | total_cases_per_milli |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 325 | USA | North America | United States | 2020-12-12 | 16245026 | 217585.0 | 215996.857 | 302002.0 | 2459.0 | 2517.857 | 49078.2 |
| 326 | USA | North America | United States | 2020-12-13 | 16432729 | 187703.0 | 216952.714 | 303629.0 | 1627.0 | 2558.000 | 49645.3 |
| 327 | USA | North America | United States | 2020-12-14 | 16627550 | 194821.0 | 216947.429 | 305279.0 | 1650.0 | 2564.429 | 50233.8 |
| 328 | USA | North America | United States | 2020-12-15 | 16836556 | 209006.0 | 214735.143 | 308398.0 | 3119.0 | 2635.571 | 50865.3 |
| 329 | USA | North America | United States | 2020-12-16 | 17083256 | 246700.0 | 218186.714 | 312062.0 | 3664.0 | 2704.429 | 51610.6 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 470 | USA | North America | United States | 2021-05-06 | 32604810 | 47366.0 | 45108.714 | 580148.0 | 789.0 | 668.714 | 98503.1 |
| 471 | USA | North America | United States | 2021-05-07 | 32651865 | 47055.0 | 43556.286 | 580901.0 | 753.0 | 671.857 | 98645.3 |
| 472 | USA | North America | United States | 2021-05-08 | 32686358 | 34493.0 | 42012.000 | 581516.0 | 615.0 | 659.143 | 98749.5 |
| 473 | USA | North America | United States | 2021-05-09 | 32707750 | 21392.0 | 40872.714 | 581754.0 | 238.0 | 646.857 | 98814.1 |
| 474 | USA | North America | United States | 2021-05-10 | 32743300 | 35550.0 | 38728.429 | 582150.0 | 396.0 | 634.286 | 98921.5 |




The number of features for machine learning algorithms was not enough in the dataset, even the accuracy was increased compared to before the dataset was divided, but the accuracy of the test was still lower than expected. So, we decided to use the logistic curve from several statistical models, which is widely used to predict infectious diseases. It was tuned in conjunction with our dataset and showed much higher accuracy compared to other machine learning

algorithms. Even though the logistic curve showed good results in the test, we wanted to further validate our model, so we compared our predictions with expert opinions. Some experts in CDC(Centers for Disease Control and Prevention) said, “by the end of July national weekly new cases could drop below 50,000”[6], which means about 7000 per a day. Our model predicts about 6,200 new cases occur a day around the end of July. Based on the CDC announcement, our model shows over 85 percent of accuracy.

```
(datetime.date(2021, 7, 31), 6224.855712136843)
```

According to our model, in late October, 2021, national weekly new cases will drop below 1000 a day, which means the covid-19 is over in the United States. As the Covid-19 continues to decrease, the job market would be back to before the occurrence of the Covid-19.

```
def logistic_c(x1, L, k, x_0):
    lists = []
    for x in x1:
        xp = k*(x-x_0)
        if xp >= 0:
            lists.append(L / ( 1. + np.exp(-xp) ) )
        else:
            lists.append(L * np.exp(xp) / ( 1. + np.exp(xp) ) )
    return lists
try:
    popt, pcov = opt.curve_fit(logistic_c, x, y,p0)
    yfit = logistic(x_, *popt)
    popt_, pcov_ = opt.curve_fit(logistic_c, x, y_,p0_)
    yfit_ = logistic(x_, *popt_)
except:
    popt, pcov = opt.curve_fit(f, x, y, method="lm", maxfev=5000)
    yfit = f(x_, *popt)
    popt_, pcov_ = opt.curve_fit(f, x, y_, method="lm", maxfev=5000)
    yfit_ = f(x_, *popt_)
```

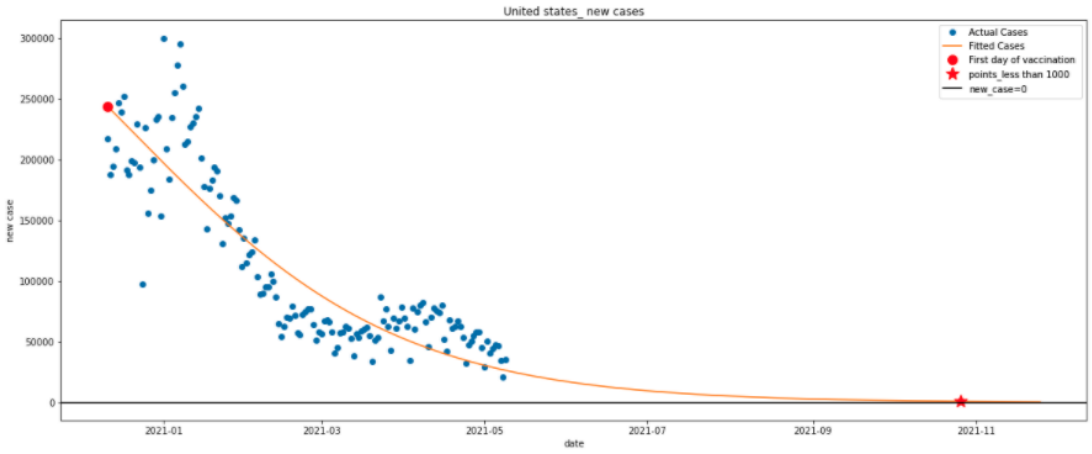
```
date1 = [datetime.date(2020, 12, 11)+ datetime.timedelta(days=i) for i in range(len(x))]
date2=[datetime.date(2020, 12, 11)+ datetime.timedelta(days=i) for i in range(350)]
```

```
fig, ax = plt.subplots(1, 1, figsize=(20, 8))
ax.plot(date1 , y, 'o', label ='Actual Cases')
ax.plot(date2, yfit, '-', label ='Fitted Cases')
j=319
j1=0
ax.plot(date2[j1],yfit[j1]  , "o",color='red',markersize=10, label="First day of vaccination ")
ax.plot(date2[j],yfit[j]  , "*",color='red',markersize=14, label="points_less than 1000 ")
ax.title.set_text('United states_ new cases')
plt.axhline(y=0, color='black', linestyle='-', label='new_case=0')
plt.ylabel("new case")
plt.xlabel("date")
plt.legend(loc="upper right")
plt.show()
```

```
def predict_day(x):
    for i in range(len(date2)):
        if yfit[i]<x:
            print(date2[i])
            break
```

```
predict_day(1000)
```

```
2021-10-26
```




**Summary**

Our project aimed to build a web application that allows users to sign in and view our analyzed data. In undertaking this project, we learned some valuable skills and lessons:

- The cloud technique simplifies many steps, we were able to build the whole web application on cloud by storing data in a cloud database.
- The stability of the database is important to a web application if you want the server to repost and request data from the database smoothly.
- We can use the Flask package from Python to link the front-end and the back-end by using HTML, Python code and SQL query in one platform.
- ETL process is a crucial precondition before getting reasonable analytics and conclusions.
- When we have an insufficient number of features, existing statistical models like the logistic curve or Poisson distribution can be an alternative for prediction.

**Future Work**

- By integrating the vaccine-related dataset recently released by CDC with our dataset, the performance of our predictive model can be evaluated and analyzed.
- It is necessary to study the complementary relationship between machine learning and existing statistical models to increase the accuracy.
- When the Covid-19 is over, it is necessary to evaluate the various analyzes and predictions we have actually made.
- Analysis comparing the political parties, may be limited, as we utilized Wikipedia to web scrape (we rationalized that it would be good coding practice) but for future work, we may opt to build a data frame manually in favor of accuracy.

**References**


[1]Heather Long, Andrew Van Dam. "U.S. Unemployment Rate Soars to 14.7 Percent, the Worst since the Depression Era." *The Washington Post*, WP Company, 9 May 2020, https://www.washingtonpost.com/business/2020/05/08/april-2020-jobs-report/

[2] "Covid Job Impacts - US Hiring Data Since March 1 2020." *Covid Job Impacts - US Hiring Data Since March 1 2020 - Registry of Open Data on AWS*, registry.opendata.aws/us-hiring-rates-pandemic/

[3] Centraal Bureau voor de Statistiek. "3,9 Duizend Mensen Overleden Aan COVID-19 in December 2020." *Centraal Bureau Voor De Statistiek*, Centraal Bureau Voor De Statistiek, 6 Apr. 2021,

https://www.cbs.nl/nl-nl/nieuws/2021/14/3-9-duizend-mensen-overleden-aan-covid-19-in-december-2020

[4] https://github.com/chloe-chau-ngo/BigCloud

[5] https://github.com/chloe-chau-ngo/BigCloud/blob/main/Analysis/analysis.ipynb

[6]AP / Updated: May 5, 2021. “Covid's US Toll Projected to Drop Sharply by the End of July - Times of India.” *The Times of India*, TOI, https://www.timesofindia.indiatimes.com/world/us/covids-us-toll-projected-to-drop-sharply-by-the-end-of-july/articleshow/82410893.cms