COVID19_Project

July 2, 2021

1 Description of the COVID-19 project

In this project, I am going to find the best Linear Regression model to make predictions of the number of confirmed and death cases due to COVID-19 in 48 counties of the United States from three dataset ("***confirmed" "death" "county***"). The datasets combined to a panel data, which contains information of 3106 counties over 88 days (from 1/21/2020 to 4/18/2020). The following is a mathematical expression of a Linear Regression Model of the data we're going to use for this project:

$\hat{y}_{it} = \hat{\beta}_0 + \hat{\beta}_1 x_{1,it} + \hat{\beta}_2 x_{2,it} + \hat{\beta}_3 x_{3,it} + ... \hat{y}_{it}$: the predicted number of confirmed/death cases of county i at time $t x_{1,2,3}$

There are two ways we can build models on this data: 1. Time fixed Goal: to understand what characteristics (eg. population, number of MDs, politics, etc.) of a county are correlated with (can be used to estimate)the number of confirmed and death cases of this county Features: time-constant variables in the 88 days range Procedure: look at all 3106 counties at one particular date, and find the best set of x that explains y. The training dataset contains data from all counties in random 80% states, and the test dataset contains data from all counties in the left 20% states.

2. County fixed Goal: to forcast the number of confirmed and death cases for a particular county Features: time-varying variables in the 88 days range Procedure: look at one county over 88 days. The training dataset contains data of all counties from first 70 days *1/21/2020* to *3/31/2020*, and the test dataset contains data of all counties from the rest 18 days *4/01/2020* to *4/18/2020*.
3. In the first section, I am going to explore the *confirmed* and *death* datasets using data visualiztions.
4. In the second section, I am going to combine *confirmed*, *death*, and *county* datasets, and perform data cleaning.
5. In the third section, I am going to build two kinds of models as we mentioned above.
6. I am going to compare training Root Mean Square Errors and cross validation Root Mean Square Errors of different models to find the one that performs the best in predicting the number of confirmed and death cases in counties, then interpret the results and evalute the best model.

1.1 Set up

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import Lasso, LassoCV
%matplotlib inline
```

2 Import Dataset

[2]:
```
#import dataset from GoogleDrive set up
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

#please authenticate.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

#files should be accessible using berkeley.edu
death_link = 'https://drive.google.com/a/berkeley.edu/file/d/
 ↪1di2u4FnBniDz7aQq2oyFZBkle5omUcUc/view?usp=sharing'
downloaded = drive.CreateFile({'id':"1di2u4FnBniDz7aQq2oyFZBkle5omUcUc"})
downloaded.GetContentFile('time_series_covid19_deaths_US.csv')
death = pd.read_csv('time_series_covid19_deaths_US.csv')

confirmed_link = "https://drive.google.com/a/berkeley.edu/file/d/
 ↪1j4lptFmM6ClNz_MjL44JEWEjQx7t4L6o/view?usp=sharing"
downloaded = drive.CreateFile({"id":"1j4lptFmM6ClNz_MjL44JEWEjQx7t4L6o"})
downloaded.GetContentFile("time_series_covid19_confirmed_US.csv")
confirmed = pd.read_csv("time_series_covid19_confirmed_US.csv")
```

```
county_link = "https://drive.google.com/file/d/
↪16ZZFK568QrndmEy9g_oKNqfh-funUh-R/view?usp=sharing"
downloaded = drive.CreateFile({"id":"16ZZFK568QrndmEy9g_oKNqfh-funUh-R"})
downloaded.GetContentFile("abridged_counties.csv")
county = pd.read_csv("abridged_counties.csv")
```

3 I. EDA

In this section, we're going to do look at the datasets, and make some adjustments for data visualization. **Figure 1** visualizes the total number of confirmed and death cases in the U.S. over time. **Figure 2** visualizes the total number of confirmed cases and the increase rate of each county using a bubble map.

Dataset Overview

[ ]:
```
death.columns, confirmed.columns
```

[ ]:
```
(Index(['UID', 'iso2', 'iso3', 'code3', 'FIPS', 'Admin2', 'Province_State',
        'Country_Region', 'Lat', 'Long_', 'Combined_Key', 'Population',
        '1/22/20', '1/23/20', '1/24/20', '1/25/20', '1/26/20', '1/27/20',
        '1/28/20', '1/29/20', '1/30/20', '1/31/20', '2/1/20', '2/2/20',
        '2/3/20', '2/4/20', '2/5/20', '2/6/20', '2/7/20', '2/8/20', '2/9/20',
        '2/10/20', '2/11/20', '2/12/20', '2/13/20', '2/14/20', '2/15/20',
        '2/16/20', '2/17/20', '2/18/20', '2/19/20', '2/20/20', '2/21/20',
        '2/22/20', '2/23/20', '2/24/20', '2/25/20', '2/26/20', '2/27/20',
        '2/28/20', '2/29/20', '3/1/20', '3/2/20', '3/3/20', '3/4/20', '3/5/20',
        '3/6/20', '3/7/20', '3/8/20', '3/9/20', '3/10/20', '3/11/20', '3/12/20',
        '3/13/20', '3/14/20', '3/15/20', '3/16/20', '3/17/20', '3/18/20',
        '3/19/20', '3/20/20', '3/21/20', '3/22/20', '3/23/20', '3/24/20',
        '3/25/20', '3/26/20', '3/27/20', '3/28/20', '3/29/20', '3/30/20',
        '3/31/20', '4/1/20', '4/2/20', '4/3/20', '4/4/20', '4/5/20', '4/6/20',
        '4/7/20', '4/8/20', '4/9/20', '4/10/20', '4/11/20', '4/12/20',
        '4/13/20', '4/14/20', '4/15/20', '4/16/20', '4/17/20', '4/18/20'],
       dtype='object'),
 Index(['UID', 'iso2', 'iso3', 'code3', 'FIPS', 'Admin2', 'Province_State',
        'Country_Region', 'Lat', 'Long_', 'Combined_Key', '1/22/20', '1/23/20',
        '1/24/20', '1/25/20', '1/26/20', '1/27/20', '1/28/20', '1/29/20',
        '1/30/20', '1/31/20', '2/1/20', '2/2/20', '2/3/20', '2/4/20', '2/5/20',
        '2/6/20', '2/7/20', '2/8/20', '2/9/20', '2/10/20', '2/11/20', '2/12/20',
        '2/13/20', '2/14/20', '2/15/20', '2/16/20', '2/17/20', '2/18/20',
        '2/19/20', '2/20/20', '2/21/20', '2/22/20', '2/23/20', '2/24/20',
        '2/25/20', '2/26/20', '2/27/20', '2/28/20', '2/29/20', '3/1/20',
        '3/2/20', '3/3/20', '3/4/20', '3/5/20', '3/6/20', '3/7/20', '3/8/20',
        '3/9/20', '3/10/20', '3/11/20', '3/12/20', '3/13/20', '3/14/20',
        '3/15/20', '3/16/20', '3/17/20', '3/18/20', '3/19/20', '3/20/20',
        '3/21/20', '3/22/20', '3/23/20', '3/24/20', '3/25/20', '3/26/20',
```

```
       '3/27/20', '3/28/20', '3/29/20', '3/30/20', '3/31/20', '4/1/20',
       '4/2/20', '4/3/20', '4/4/20', '4/5/20', '4/6/20', '4/7/20', '4/8/20',
       '4/9/20', '4/10/20', '4/11/20', '4/12/20', '4/13/20', '4/14/20',
       '4/15/20', '4/16/20', '4/17/20', '4/18/20'],
      dtype='object'))
```

[ ]:
```
county
```

[ ]:
```
     countyFIPS  STATEFP  …  HPSAServedPop HPSAUnderservedPop
0         01001      1.0  …            NaN                NaN
1         01003      1.0  …            NaN                NaN
2         01005      1.0  …         5400.0            18241.0
3         01007      1.0  …        14980.0             6120.0
4         01009      1.0  …        31850.0            25233.0
…           …        …  …             …                  …
3239      15005     15.0  …            NaN                NaN
3240      72039     72.0  …            NaN                NaN
3241      72069     72.0  …            NaN                NaN
3242      City1      NaN  …            NaN                NaN
3243      City2      NaN  …            NaN                NaN

[3244 rows x 87 columns]
```

Data Adjustment

[ ]:
```
death = death.melt(id_vars=['UID', 'iso2', 'iso3', 'code3', 'FIPS', 'Admin2',␣
↪'Province_State',
       'Country_Region', 'Lat', 'Long_', 'Combined_Key', "Population"],
        var_name="Date",
        value_name="num_of_death")
confirmed = confirmed.melt(id_vars=['UID', 'iso2', 'iso3', 'code3', 'FIPS',␣
↪'Admin2', 'Province_State',
       'Country_Region', 'Lat', 'Long_', 'Combined_Key'],
        var_name="Date",
        value_name="num_of_confirmed")
```

[State-Level] **Figure 1** visualizes the total number of confirmed and death cases in the states of interest over time. We're interested in 1) the state of the first death case 2) the second state that has confirmed case 3) the state who has the most number of death cases on the last day of this dataset 4) the state who has the most number of confirmed case of this dataset 5) California

1.1 Find the state with the fisrt death case in the United States

[ ]:
```
death.iloc[death.ne(0).idxmax()[-1]]
```

[ ]:
```
UID                          84053033
iso2                               US
iso3                              USA
```

```
code3                              840
FIPS                             53033
Admin2                            King
Province_State                Washington
Country_Region                        US
Lat                              47.4914
Long_                           -121.835
Combined_Key      King, Washington, US
Population                       2252782
Date                              2/29/20
num_of_death                            1
Name: 126664, dtype: object
```

1.2 Find the second state after Washington that has confirmed case.

[ ]:
```
confirmed.iloc[confirmed.loc[confirmed["Province_State"] != "Washington"].ne(0).
↪idxmax()[-1]]
```

[ ]:
```
UID                            84017031
iso2                                 US
iso3                                USA
code3                               840
FIPS                              17031
Admin2                             Cook
Province_State                 Illinois
Country_Region                       US
Lat                             41.8414
Long_                          -87.8166
Combined_Key         Cook, Illinois, US
Date                             1/24/20
num_of_confirmed                       1
Name: 7125, dtype: object
```

1.3 Find the state who has the most number of death cases on the last day of this dataset

[ ]:
```
death.iloc[death[death['Date'] == death['Date'].max()]['num_of_death'].idxmax()]
```

[ ]:
```
UID                                84036061
iso2                                     US
iso3                                    USA
code3                                   840
FIPS                                  36061
Admin2                             New York
Province_State                     New York
Country_Region                           US
Lat                                 40.7673
Long_                              -73.9715
Combined_Key      New York City, New York, US
```

```
Population                                5803210
Date                                      4/9/20
num_of_death                                5150
Name: 255753, dtype: object
```

1.4 Find the state who has the most number of confirmed cases on the last say of this dataset

[ ]:
```
confirmed.iloc[confirmed[confirmed['Date'] == confirmed['Date'].
↪max()]['num_of_confirmed'].idxmax()]
```

[ ]:
```
UID                                      84036061
iso2                                           US
iso3                                          USA
code3                                         840
FIPS                                        36061
Admin2                                   New York
Province_State                           New York
Country_Region                                 US
Lat                                       40.7673
Long_                                    -73.9715
Combined_Key        New York City, New York, US
Date                                       4/9/20
num_of_confirmed                            87028
Name: 255753, dtype: object
```

Figure 1 plots total confirmed cases in the United States over time.

[ ]:
```
total_confirmed_by_date = pd.DataFrame({"total_confirmed_by_day":confirmed.
↪groupby("Date").sum()["num_of_confirmed"]})
confirmed = confirmed.merge(total_confirmed_by_date, how = "left", on =␣
↪["Date"])

total_death_by_date = pd.DataFrame({"total_death_by_day":death.groupby("Date").
↪sum()["num_of_death"]})
death = death.merge(total_death_by_date, how = "left", on = ["Date"])
```

[ ]:
```
dates = confirmed['Date']
plt.plot(dates, confirmed["total_confirmed_by_day"])
plt.plot(dates, death["total_death_by_day"])
plt.legend(['confirmed', 'death'], loc='upper left')
plt.xticks(dates.unique()[np.arange(0, 89, 7)], rotation = 65)
plt.xlabel("dates")
plt.ylabel("Total confirmed/death cases in the U.S.")
plt.title("Total number of confirmed cases in the U.S. over time");
```

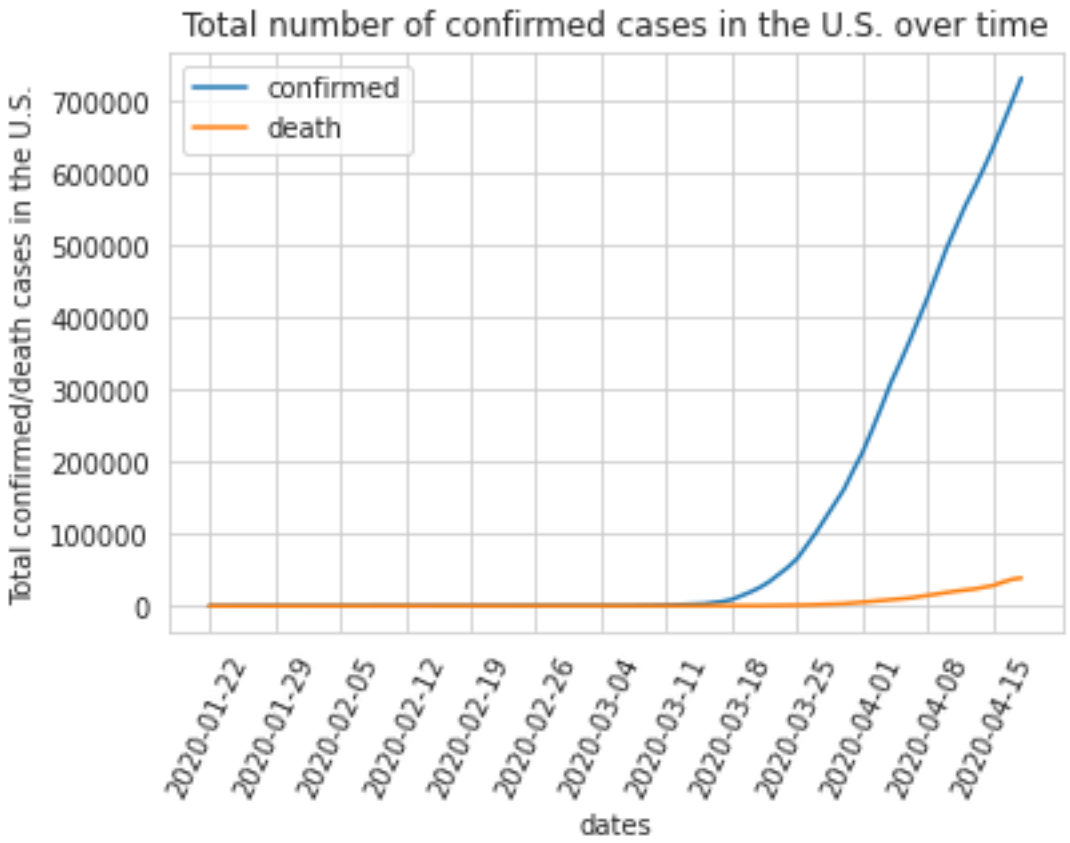


Figure 2 is a bubble map that visualizes the total number of confirmed cases and daily increased confirmed cases of each county on a U.S. map overtime. The size of bubble shows total number of confirmed cases, and the color of the bubble show the number of daily increased confirmed case.

[ ]:
```
fig = px.scatter_geo(confirmed, lon = "Long_", lat = "Lat", locationmode␣
↪="USA-states",
                     color = confirmed.groupby('UID')['num_of_confirmed'].
↪diff().fillna(0).apply(lambda x: np.log(x+1)),
                     hover_name = "Admin2",
                     size= confirmed['num_of_confirmed'].apply(lambda x: np.
↪log(x+1)),
                     animation_frame = confirmed["Date"].apply(lambda x: x.
↪strftime('%m-%d-%y')),
                     animation_group = "Province_State",
                     projection = "albers usa", opacity = 0.5,
                     title = "Total confirmed cases of each county each day",␣
↪size_max = 15
                     )
fig.show()
```

```
Output hidden; open in https://colab.research.google.com to view.
```

4 II. Data cleaning

In this section, we're going to combine the tables, filter out data that is irrelevant to our analysis, and deal with NAs.

2.1 Combining

[ ]:
```
cols_to_use = death.columns.difference(confirmed.columns)
combined = pd.merge(confirmed, death[cols_to_use], left_index=True,␣
↪right_index=True, how='outer')
```

[ ]:
```
county = county[county['countyFIPS'] != 'City1'][county['countyFIPS'] !=␣
↪'City2']
county['countyFIPS'] = (str(840) + county['countyFIPS']).astype(str).astype(int)
data = pd.merge(combined, county, how='left', left_on='UID',␣
↪right_on='countyFIPS')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```

2.2 Filtering In the COVID-19 dataset, we're excluding data from Grand Princess and other cruise.

[ ]:
```
state48 = ["Alabama", "Arizona", "Arkansas", "California",␣
↪"Colorado","Connecticut", "Delaware",
           "Florida", "Georgia", "Idaho", "Illinois", "Indiana", "Iowa",␣
↪"Kansas", "Kentucky",
           "Louisiana", "Maine", "Maryland", "Massachusetts", "Michigan",␣
↪"Minnesota", "Mississippi",
           "Missouri", "Montana", "Nebraska","Nevada","New Hampshire","New␣
↪Jersey","New Mexico",
           "New York","North Carolina","North␣
↪Dakota","Ohio","Oklahoma","Oregon","Pennsylvania",
           "Rhode Island","South Carolina","South␣
↪Dakota","Tennessee","Texas","Utah","Vermont",
           "Virginia","Washington","West Virginia","Wisconsin","Wyoming"]
data = data[data['Province_State'].isin(state48)]
```

We delete all the counties that have invalid FIPS, which means marked unassigned, or are marked as out of the state.

[ ]:
```
data = data[data['FIPS'].notnull()][data['Admin2'] !=␣
↪'Unassigned'][data['Admin2'].str.contains('Out of') == False]
```

```
data = data.drop(columns=['iso2', 'iso3', 'code3', 'FIPS', 'Country_Region',␣
↪'Combined_Key', 'HPSAServedPop', 'HPSAUnderservedPop']).drop(columns=data.
↪loc[:, 'countyFIPS':'POP_LONGITUDE']).drop(columns=data.loc[:,␣
↪'3-YrMortalityAge<1Year2015-17':'mortality2015-17Estimated'])
data = data.reset_index(drop=True)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```

[ ]:
```
#drop columns providing similar info
data.drop(['PopTotalMale2017', 'PopTotalFemale2017'], axis = 1, inplace = True)
#eliminate columns contain 2010
data = data[data.columns.drop(list(data.filter(regex='2010')))]
```

In addition, a good independent variable for linear regression should have large variance. So last step is to eliminate features with no variance.

[ ]:
```
data.iloc[:,13:].std().sort_values()  #foreign travel ban and federal␣
↪guidelines have no variance
data.drop(["foreign travel ban", "federal guidelines"], axis = 1, inplace =␣
↪True)
```

2.3 Check NAs and Invalid Entries

[ ]:
```
data.isna().sum().sort_values(ascending=False).head(15)
```

[ ]:
```
3-YrDiabetes2015-17              149160
HPSAShortage                      95568
stay at home                      43120
>50 gatherings                    10472
>500 gatherings                   10472
entertainment/gym                  5808
MedicareEnrollment,AgedTot2017     1936
SVIPercentile                       176
StrokeMortality                     176
public schools                       88
CensusRegionName                     88
CensusDivisionName                   88
Rural-UrbanContinuumCode2013         88
PopulationEstimate2018               88
dem_to_rep_ratio                     88
dtype: int64
```

We find out that the county Oglala Lakota, South Dakota does not have any information from the *county* table (88 missing values for all columns after *CensusRegionName* column). Since Oglala Lakota provides little information about how other features perform in predicting the the total

number of confirmed and death cases, we decided to drop this county.

```
[ ]: data.drop(data[data["Admin2"] == "Oglala Lakota"].index, axis = 0, inplace =␣
     ↪True)
```

Features *"3-YrDiabetes2015-17", "SVIPercentile", "StrokeMortality", "MedicareEnrollment,AgedTot2017"* also have missing values. Since they are numeric values, missing values will be replace with the mean of all other counties in the same state.

```
[ ]: #use state mean to fill missing values of quantitative values
     def fill_state_mean(column, dataset=data):
       dataset.loc[data[column].isna(), column] = dataset.
     ↪groupby("Province_State")[column].transform(lambda x: x.fillna(x.mean()))
       return dataset

     for i in ["3-YrDiabetes2015-17", "SVIPercentile", "StrokeMortality",␣
     ↪"MedicareEnrollment,AgedTot2017","HPSAShortage"]:
       fill_state_mean(i)
```

After filling with state mean, there are still NAs in "*HPSAShortage*".This is because some states has no "*HPSAShortage*" data for all counties. Let's find out the state without HPSAShortage data.

```
[ ]: data.loc[data["HPSAShortage"].isna(), "Province_State"].unique()
     data.loc[data["HPSAShortage"].isna(), "HPSAShortage"] = data["HPSAShortage"].
     ↪mean() + np.random.normal(0,1,data["HPSAShortage"].isna().sum())
```

Missing values in columns "*Stay at home", ">500 gatherings", ">50 gatherings", "entertainment/gym*" are interpreted as the policy was not excuted in this county by April 18th. We're going to transform these column using the following rule: if the date of a row is before the date of "*stay at home*", the value of "*stay at home*" column is zero; if the date of a row is after the date of "*stay at home*", the value of "*stay at home*" column is one.

```
[ ]: data['ordinal_date'] = data['Date'].apply(pd.Timestamp).apply(pd.Timestamp.
     ↪toordinal)
```

```
[ ]: na_date = pd.Timestamp.toordinal(pd.Timestamp(year=2020, month=4, day=19,␣
     ↪hour=0))
     to_days_features = ["stay at home", ">50 gatherings", ">500 gatherings",␣
     ↪"public schools",
                         "restaurant dine-in", "entertainment/gym"]
     data[to_days_features] = data[to_days_features].fillna(na_date)
     data[to_days_features] = data[to_days_features].transform(lambda x: np.where(x␣
     ↪< data['ordinal_date'], 1, 0))
```

5 III. Time Fixed Model

In order to understand what characteristics of a county can be used to estimate the number of confirmed and death cases of this county, we are going to look at data of different counties at a

particular date.

```
[ ]: #Define a function that takes two argument: a date, a name of dependent␣
      ↪variable (death/confirmed), a dataframe
     #and return a dataframe of features X, and an araay of dependent variable y
     def produce_X_y(date, y_name, dataset = data):
       model_data = dataset.loc[dataset['Date'] == date, :]
       y = model_data["num_of_" + y_name]
       X = model_data.drop(list(model_data.filter(regex=y_name)), axis = 1)
       return X,y
```

```
[ ]: X,y = produce_X_y("4/18/20", "death")
```

3.1 Train/Test Split

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
      ↪random_state = 45)
```

3.2 Select Features

```
[ ]: # define a function that select time constant numeric variables
     def time_constant_numvar(df):
       return df.drop(['CensusRegionName', 'CensusDivisionName'] + to_days_features,␣
      ↪axis = 1).iloc[:, 6:]
```

```
[ ]: time_constant_var = time_constant_numvar(X)
```

```
[ ]: # def a function that select k best features by SKLearn
     from sklearn.feature_selection import SelectKBest, chi2

     def k_best(numeric_X, y, k):
       selector = SelectKBest(chi2, k=k)
       selector.fit(numeric_X, y)
       cols = selector.get_support(indices=True)
       features_df_new = numeric_X.iloc[:,cols]
       return features_df_new.columns
```

```
[ ]: categorical_features = ["Province_State"]

     best_8 = k_best(time_constant_var, y, 8).to_list()
     #['num_of_confirmed', 'Population','PopulationEstimate2018',
     #'PopulationEstimate65+2017','#EligibleforMedicare2018',
     #'MedicareEnrollment,AgedTot2017', '#FTEHospitalTotal2017',
     #"TotalM.D.'s,TotNon-FedandFed2017"]

     best_15 = k_best(time_constant_var, y, 15).to_list()
     #['num_of_confirmed', 'num_of_confirmed_log', 'Population',
     #'PopulationEstimate2018','PopulationEstimate65+2017','#EligibleforMedicare2018',
```

```
#'MedicareEnrollment,AgedTot2017','3-YrDiabetes2015-17','HeartDiseaseMortality',
#'#FTEHospitalTotal2017',"TotalM.D.'s,TotNon-FedandFed2017",
#'#HospParticipatinginNetwork2017','#Hospitals','#ICU_beds','HPSAShortage']
```

3.3 Build Models We're going to build three kinds of models. Linear Regression Model, Ridge Linear Regression Model, and Lasso Linear Regression Model.

[ ]:
```
models = {}

#Linear Regression model with top 8 numeric features and categorical features
model_8 = Pipeline([
  ("SelectColumns", ColumnTransformer([
      ("keep", StandardScaler(), best_8),
      ("cat", OneHotEncoder(), categorical_features)
  ])),
  ("Imputation", SimpleImputer()),
  ("LinearModel", LinearRegression())
])

#Linear Regression model with top 15 numeric features and categorical features
model_15 = Pipeline([
  ("SelectColumns", ColumnTransformer([
      ("keep", StandardScaler(), best_15),
      ("cat", OneHotEncoder(), categorical_features)
  ])),
  ("Imputation", SimpleImputer()),
  ("LinearModel", LinearRegression())
])
#Ridge model with top 15 numeric features and catgeorical features
alphas = np.linspace(0.5, 3, 30)
model_Ridge = Pipeline([
        ("SelectColumns", ColumnTransformer([
            ("keep", StandardScaler(), best_15),
            ("cat", OneHotEncoder(), categorical_features)
        ])),
        ("Imputation", SimpleImputer()),
        ("LinearModel", RidgeCV(alphas = alphas))
    ])
#Lasso model with top 15 numeric features and catgeorical features
model_Lasso = Pipeline([
        ("SelectColumns", ColumnTransformer([
            ("keep", StandardScaler(), best_15),
            ("cat", OneHotEncoder(), categorical_features)
        ])),
        ("Imputation", SimpleImputer()),
        ("LinearModel", LassoCV(cv = 5, max_iter = 5000, random_state = 19))
    ])
```

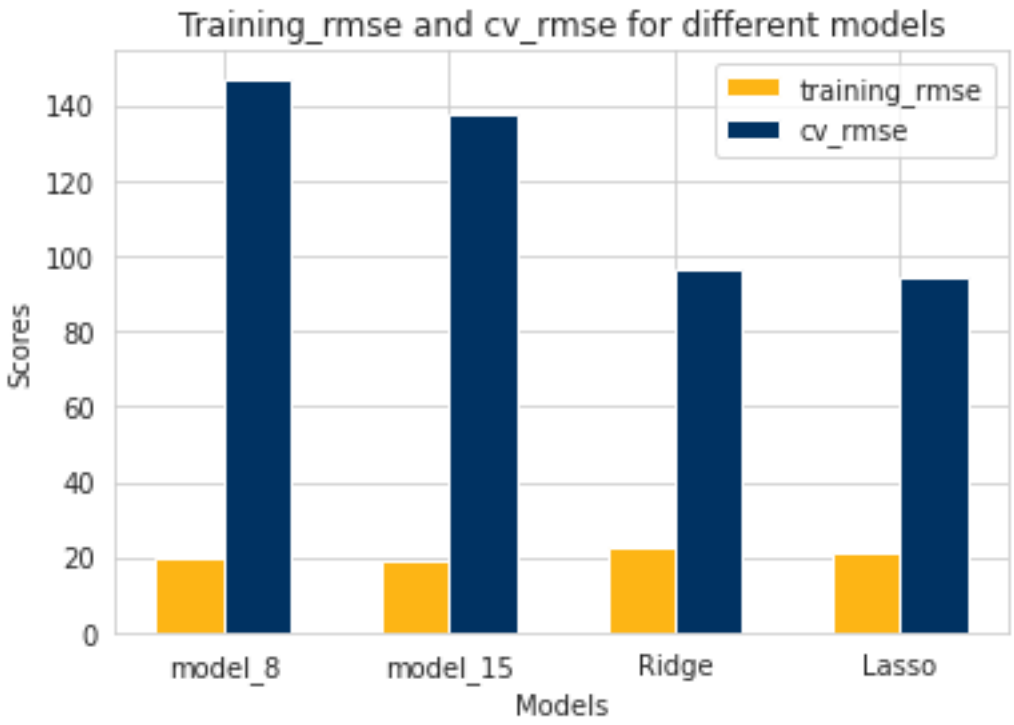
```
model_8.fit(X_train,y_train)
model_15.fit(X_train,y_train)
model_Ridge.fit(X_train,y_train)
model_Lasso.fit(X_train, y_train)

models["model_8"] = model_8
models["model_15"] = model_15
models["Ridge"] = model_Ridge
models["Lasso"] = model_Lasso
```

3.4 Evaluate Models

[ ]:
```
def rmse_score(model, X, y):
    return np.sqrt(np.mean((y - model.predict(X))**2))
```

[ ]:
```
def compare_models(models, test_rmse = False):
    training_rmse = [rmse_score(model, X_train, y_train) for model in models.
 ↪values()]
    validation_rmse = [np.mean(cross_val_score(model, X_train, y_train,␣
 ↪scoring=rmse_score, cv=5))
                       for model in models.values()]
    names = list(models.keys())
    ind = np.arange(len(names))
    fig = plt.figure()
    ax = plt.subplot(111)
    w = 0.3
    train_bar = ax.bar(ind + 0.5*w , training_rmse, width = w, color =␣
 ↪"#FDB515", align = "center")
    cv_bar = ax.bar(ind + 1.5*w, validation_rmse, width = w, color = "#003262",␣
 ↪align = "center")
    ax.set_xticks(ind+w)
    ax.set_xticklabels(names)
    ax.legend((train_bar[0], cv_bar[0]), ('training_rmse', 'cv_rmse'))
    ax.set(xlabel = "Models",
        ylabel = "Scores",
        title = "Training_rmse and cv_rmse for different models")
    if test_rmse:
      test_rmse = [rmse_score(model, X_test, y_test) for model in models.
 ↪values()]
      test_bar = ax.bar(ind + 2.5*w, test_rmse, width = w, color = "#B9D3B6",␣
 ↪align = "center")
      ax.legend((train_bar[0], cv_bar[0], test_bar[0]), ('training_rmse',␣
 ↪'cv_rmse', "test_rmse"))
    return None
```

[ ]:
```
compare_models(models)
```



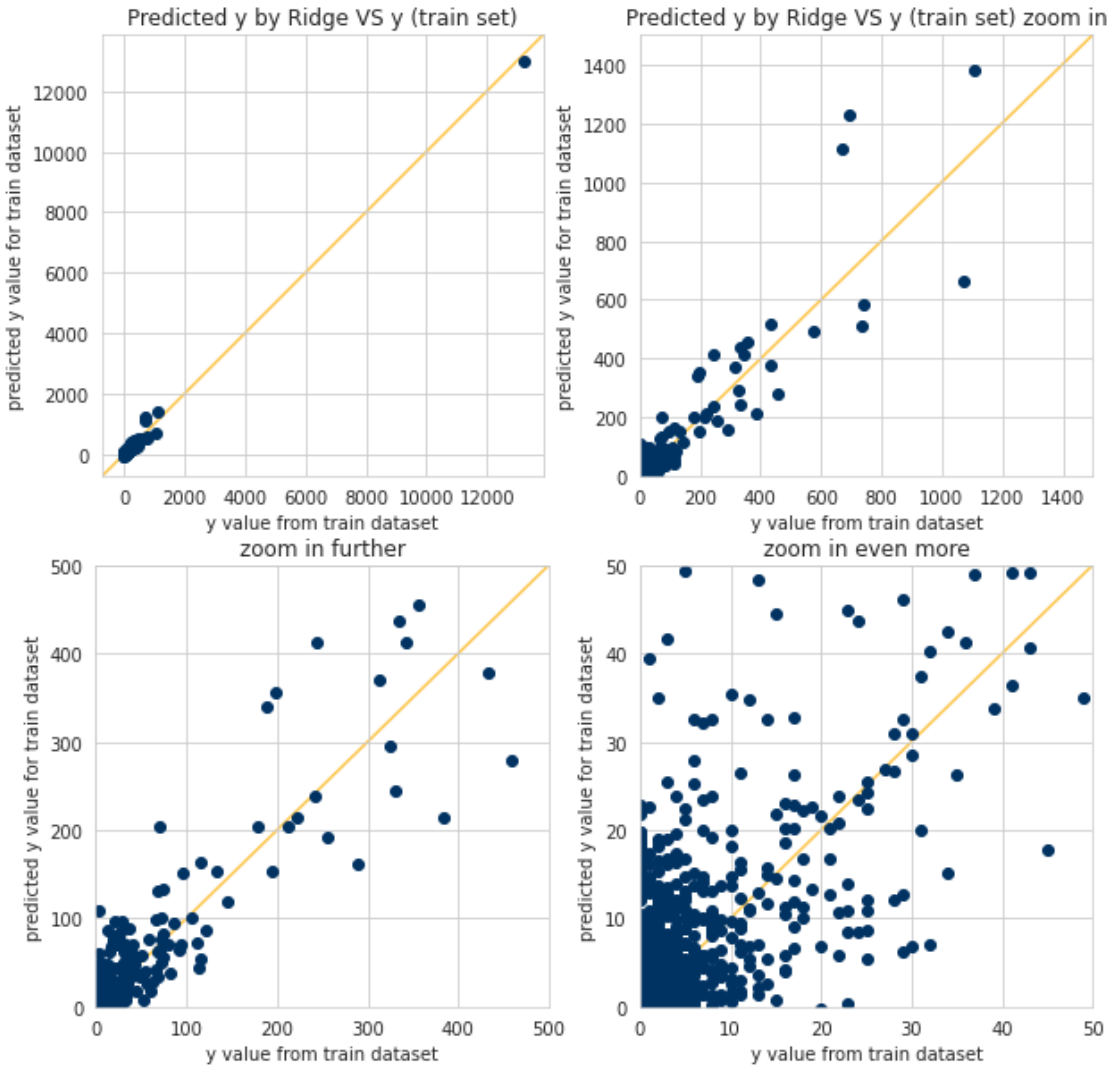
3.5 Interpret results

Training RMSE score is much lower than cross validation RMSE score for all models. So, we can expect high test RMSE, which indicates that it is not a good way to use charateristics of a county to predict the number of death cases due to COVID-19. From the plot above, Ridge model and Lasso model performs a lot better than Linear Regression Models.

[ ]:
```
best_result = pd.DataFrame({"county": X_train['Admin2'] + "," +␣
 ↪X_train["Province_State"], "y_hat": models['Ridge'].predict(X_train), "y":␣
 ↪y_train})
fig, ((ax1,ax2),(ax3,ax4)) = plt.subplots(2, 2, figsize=(10,10))

for ax in [ax1,ax2,ax3,ax4]:
  ax.scatter(best_result["y"], best_result["y_hat"], c = "#003262")
lims = [
    np.min([ax1.get_xlim(), ax.get_ylim()]),
    np.max([ax1.get_xlim(), ax.get_ylim()]),]
ax1.plot(lims, lims, 'k-', alpha=0.75, zorder=0, c = "#FDB515")
ax1.set_aspect('equal')
ax1.set_xlim(lims)
ax1.set_ylim(lims)
ax1.set(xlabel = "y value from train dataset",
        ylabel = "predicted y value for train dataset",
        title = "Predicted y by Ridge VS y (train set)")
```

```
ax2.plot((0, 1500), (0, 1500), 'k-', alpha=0.75, zorder=0, c = "#FDB515")
ax2.set_xlim(0, 1500)
ax2.set_ylim(0, 1500)
ax2.set(xlabel = "y value from train dataset",
        ylabel = "predicted y value for train dataset",
        title = "Predicted y by Ridge VS y (train set) zoom in")


ax3.plot((0, 500), (0, 500), 'k-', alpha=0.75, zorder=0, c = "#FDB515")
ax3.set_xlim(0, 500)
ax3.set_ylim(0, 500)
ax3.set(xlabel = "y value from train dataset",
        ylabel = "predicted y value for train dataset",
        title = "zoom in further")


ax4.plot((0, 50), (0, 50), 'k-', alpha=0.75, zorder=0, c = "#FDB515" )
ax4.set_xlim(0, 50)
ax4.set_ylim(0, 50)
ax4.set(xlabel = "y value from train dataset",
        ylabel = "predicted y value for train dataset",
        title = "zoom in even more")
;
```

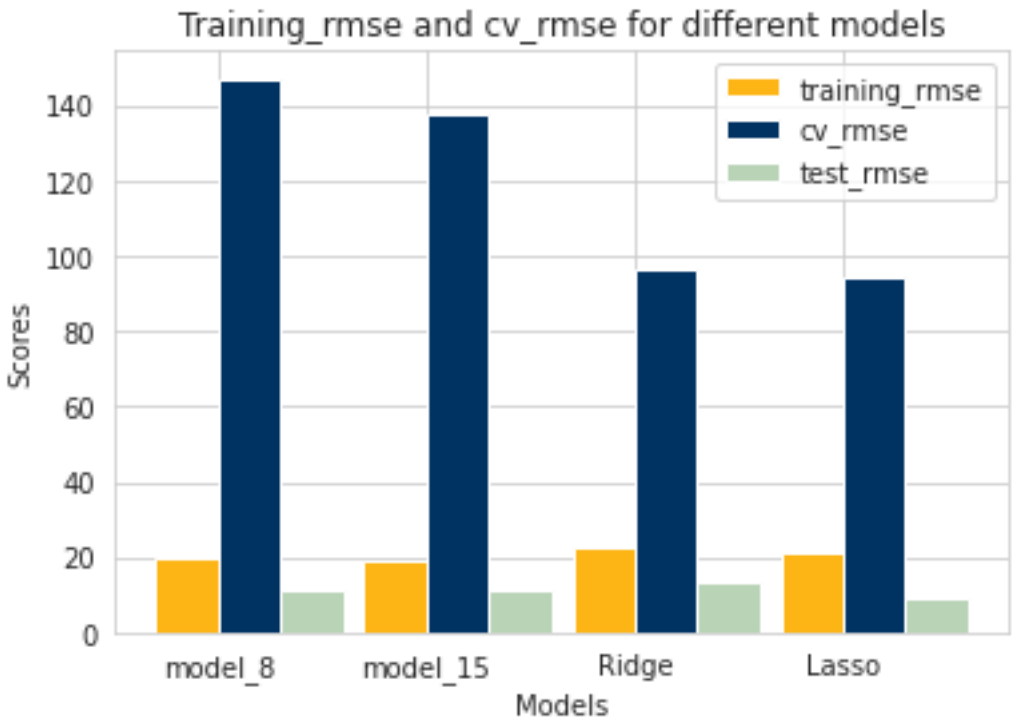
[ ]: ''



As we zoom in, we see that the predicted number of death cases by the model deviates a lot from the actual number of death cases for many counties. It looks like almost randomly distributed. Therefore, we conclude that it is not a good way to use charateristics of a county (population, #hospitals, etc.) to predict the number of death cases in a county at a particular day.

3.6 Test Models

[ ]:
```
#Warning: take a minute or two to run
compare_models(models, test_rmse = True)
```



6 IV. County Fixed Model

By looking at the plot of total number of confirmed cases over time, we have two possible models in mind:
Model 1:

$$log(\hat{y}_t) = \hat{\beta}_0 + \hat{\beta}_1 log(y_{t-1}) + \hat{\beta}_2 * D_{\text{Shelter in Place}}$$

Model 2:

$$\hat{y}_t = \hat{\beta}_0 + \hat{\beta}_1 (y_{t-1}) + \hat{\beta}_2 (y_{t-1})^2 + \hat{\beta}_3 * D_{\text{Shelter in Place}}$$

For both models:

$\hat{y}_t$: the predicted number of confirmed cases at time ty_{t-1} : the number of confirmed cases reported in the past d

4.1 Data transformation

Inspired by EDA, the number of confirmed and death cases grow exponentially. In the following, we make log transformations in order to make the data suitable for linear models.Since there are many 0 entries in the columns *num_of_death*, *num_of_confirmed*, we're going to make **log(x+1)** tranformation to avoid error.

Since this is a time series data, the number of confirmed and death cases today give us a lot of insight about the number of confirmed and death cases in the future. Therefore, we're going to add a column called *yeasterday*, which contains the number of confirmed and death cases of this

county in the past day. The first day in this dataset 1/21/2020 would have no information of the past day Therefore, we're going to drop all data in the first day 1//21/2020.

[ ]:
```
#define a function that product X and y
def time_series_data(county_UID, y, dataset = data, sip = to_days_features):
  model_data = dataset.loc[data['UID'] == county_UID, ['Date', y] + sip]
  model_data['log'] = model_data[y].apply(lambda x: np.log(x+1))
  model_data['yesterday'] = model_data[y].shift()
  model_data = model_data[model_data.Date != "1/22/20"]
  model_data['yesterday_log'] = model_data['yesterday'].apply(lambda x : np.
 ↪log(x+1))
  model_data['yesterday_sq'] = model_data['yesterday'] ** 2
  return model_data
```

[ ]:
```
alameda_confirmed = time_series_data(84006001, "num_of_confirmed")
```

[ ]:
```
# a function that produce X and y that takes in following argument
# y_name: "num_of_confirmed" OR "num_of death"
# method: "log" or "poly"
def time_series_X_y(method, y_name, dataset, sip = to_days_features):
  if method == "log":
    y = dataset[['log']].apply(lambda x: x/x.max() if x.max() != 0 else x)
    X = dataset[['yesterday_log'] + to_days_features].apply(lambda x: x/x.max()␣
 ↪if x.max() != 0 else x)
  elif method == "poly":
    y = dataset[[y_name]].apply(lambda x: x/x.max() if x.max() != 0 else x)
    X = dataset[['yesterday', 'yesterday_sq'] + to_days_features].apply(lambda␣
 ↪x: x/x.max() if x.max() != 0 else x)
  else:
    raise Exception("wrong method")
  return X,y
```

[ ]:
```
#For log model
X_log,y_log = time_series_X_y("log", "num_of_confirmed", alameda_confirmed)
#for poly model
X_poly, y_poly = time_series_X_y("poly", "num_of_confirmed", alameda_confirmed)
```

4.2 Train/Test split

[ ]:
```
def time_series_split(X,y):
  return X[:70], X[70:], y[:70], y[70:]
```

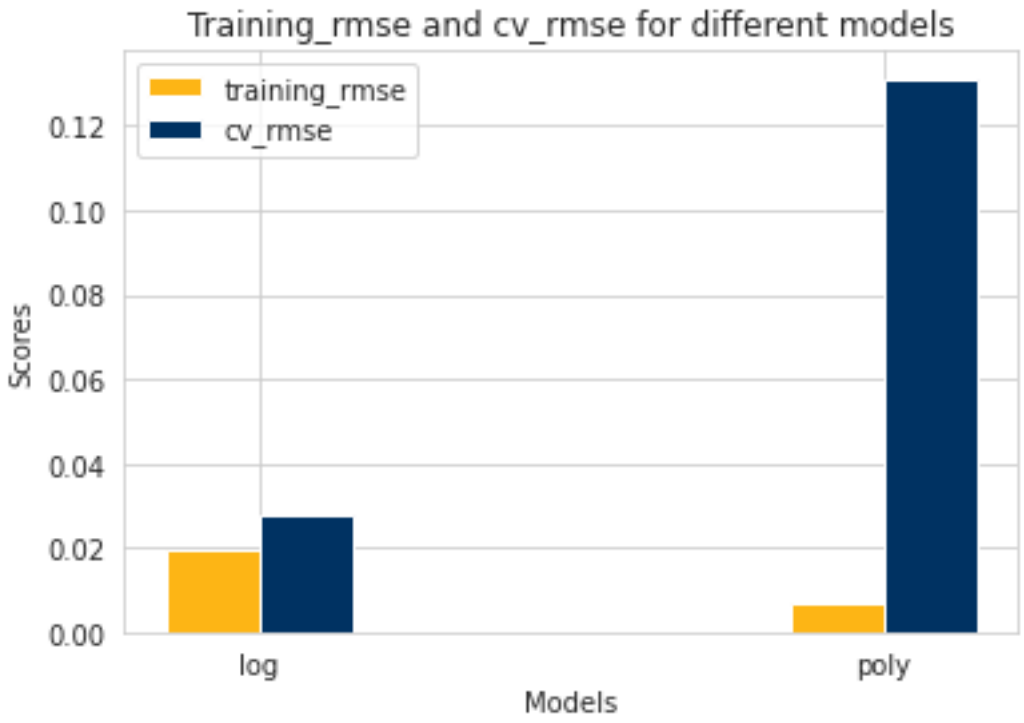
[ ]:
```
X_log_train, X_log_test, y_log_train, y_log_test = time_series_split(X_log,␣
 ↪y_log)
X_poly_train, X_poly_test, y_poly_train, y_poly_test =␣
 ↪time_series_split(X_poly, y_poly)
```

4.3 Model evaluation

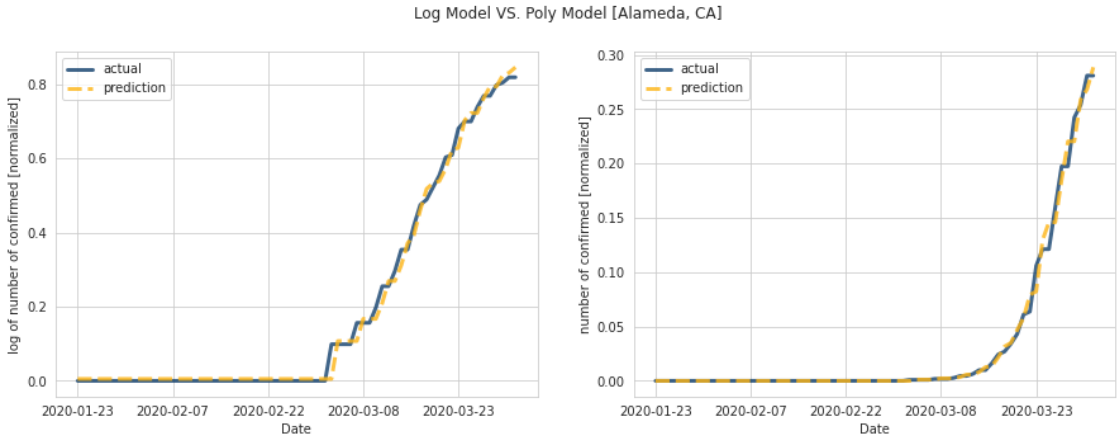
[ ]:
```
log_model = LinearRegression()
log_model.fit(X_log_train, y_log_train)

poly_model = LinearRegression()
poly_model.fit(X_poly_train, y_poly_train);
```

[ ]:
```
training_rmse = [rmse_score(log_model, X_log_train, y_log_train)[0],␣
↪rmse_score(poly_model, X_poly_train, y_poly_train)[0]]
cv_rmse = [np.mean(cross_val_score(log_model, X_log_train, y_log_train,␣
↪scoring=rmse_score, cv=5)),
           np.mean(cross_val_score(poly_model, X_poly_train, y_poly_train,␣
↪scoring=rmse_score, cv=5))]
```

[ ]:
```
names = ['log', 'poly']
ind = np.arange(len(names))
fig = plt.figure()
ax = plt.subplot(111)
w = 0.15
train_bar = ax.bar(ind + 0.5*w , training_rmse, width = w, color = "#FDB515",␣
↪align = "center")
cv_bar = ax.bar(ind + 1.5*w, cv_rmse, width = w, color = "#003262", align =␣
↪"center")
ax.set_xticks(ind+w)
ax.set_xticklabels(names)
ax.legend((train_bar[0], cv_bar[0]), ('training_rmse', 'cv_rmse'))
ax.set(xlabel = "Models",
    ylabel = "Scores",
    title = "Training_rmse and cv_rmse for different models");
```



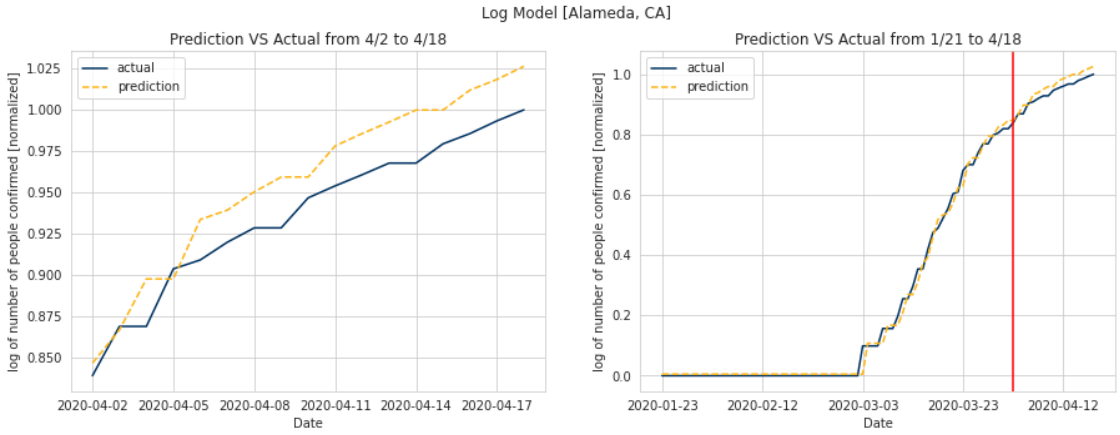
[ ]:
```
log_train_predictions = log_model.predict(X_log_train)
poly_train_predictions = poly_model.predict(X_poly_train)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (15, 5))
x_axis = alameda_confirmed['Date'].astype(str).unique()[:70]

ax1.plot(x_axis, y_log_train, c = "#003262", linewidth=3, alpha = 0.75)
ax1.plot(x_axis, log_train_predictions, c = "#FDB515", linewidth=3, linestyle =␣
↪'--', alpha = 0.8)
ax1.set_ylabel("log of number of confirmed [normalized]")
ax1.legend(['actual','prediction'])

ax2.plot(x_axis, y_poly_train, c = "#003262", linewidth=3, alpha = 0.75)
ax2.plot(x_axis, poly_train_predictions, c = "#FDB515", linewidth=3, linestyle␣
↪= '--', alpha = 0.8)
ax2.set_ylabel("number of confirmed [normalized]")
ax2.legend(['actual', 'prediction'])

plt.setp([ax1,ax2], xlabel = 'Date',
         xticks=np.arange(0,70,15),
         xticklabels = x_axis[np.arange(0,70,15)])

fig.suptitle('Log Model VS. Poly Model [Alameda, CA]');
```



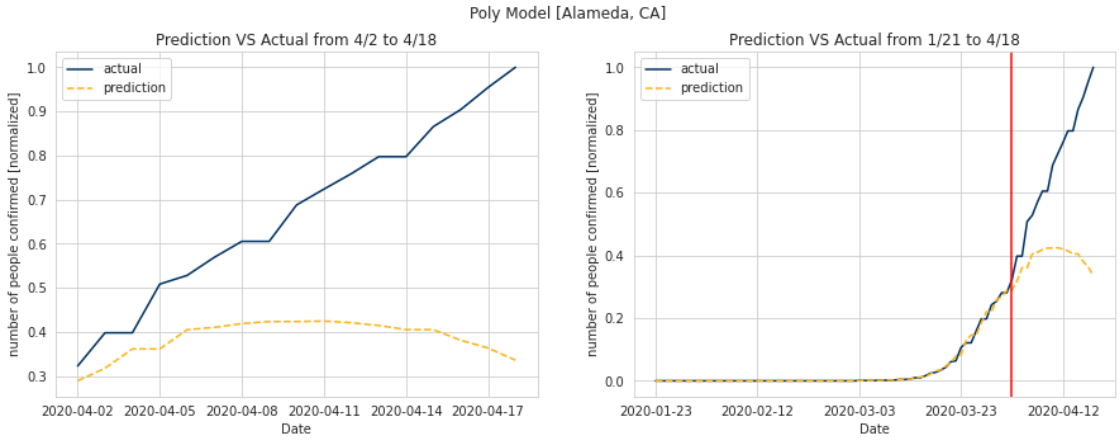
4.4 Predictions

Accuracy : Log Model > Polynomial model

[ ]:
```
log_test_predictions = log_model.predict(X_log_test)
x_axis = alameda_confirmed['Date'].astype(str).unique()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (15, 5))
ax1.plot(x_axis[70:], y_log_test, c = "#003262")
ax1.plot(x_axis[70:], log_test_predictions, c = "#FDB515", linestyle = '--')
ax1.set(xticks = (x_axis[70:][np.arange(0,18,3)]),
        title = "Prediction VS Actual from 4/2 to 4/18")
ax1.legend(['actual', 'prediction'])


ax2.plot(x_axis, y_log, c = "#003262")
ax2.plot(x_axis, np.concatenate((log_train_predictions,log_test_predictions)),␣
 ↪c = "#FDB515", linestyle = '--' )
ax2.set(xticks = (x_axis[np.arange(0,88,20)]),
        title = "Prediction VS Actual from 1/21 to 4/18")
ax2.axvline(x='2020-04-02', c = 'r')
ax2.legend(['actual', 'prediction'])

plt.setp([ax1,ax2], xlabel = 'Date', ylabel = "log of number of people␣
 ↪confirmed [normalized]")
fig.suptitle('Log Model [Alameda, CA]');
```



[ ]:
```
poly_test_predictions = poly_model.predict(X_poly_test)
x_axis = alameda_confirmed['Date'].astype(str).unique()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (15, 5))
ax1.plot(x_axis[70:], y_poly_test, c = "#003262")
ax1.plot(x_axis[70:], poly_test_predictions, c = "#FDB515", linestyle = '--')
ax1.set(xticks = (x_axis[70:][np.arange(0,18,3)]),
        title = "Prediction VS Actual from 4/2 to 4/18")
ax1.legend(['actual', 'prediction'])


ax2.plot(x_axis, y_poly, c = "#003262")
ax2.plot(x_axis, np.
 ↪concatenate((poly_train_predictions,poly_test_predictions)), c = "#FDB515",␣
 ↪linestyle = '--' )
ax2.set(xticks = (x_axis[np.arange(0,88,20)]),
        title = "Prediction VS Actual from 1/21 to 4/18")
ax2.axvline(x='2020-04-02', c = 'r')
ax2.legend(['actual', 'prediction'])

plt.setp([ax1,ax2], xlabel = 'Date', ylabel = "number of people confirmed␣
 ↪[normalized]")
fig.suptitle('Poly Model [Alameda, CA]');
```



As we can see from the plot above, the polynomial model made poor prediction.

4.5 Apply to other counties

[ ]:
```
random_30_counties = np.random.choice(data['UID'],25)
random_30_counties_names = data.loc[data['UID'].isin(random_30_counties),␣
↪["Admin2", "Province_State"]].agg(',',.join, axis=1).unique()
```

[ ]:
```
# Write a function that does the same prcoess as above
def log_model(county, y, test_rmse = False):
  result = []
  data = time_series_data(county, y)
  X,y = time_series_X_y("log", y, data)
  X_train, X_test, y_train, y_test = time_series_split(X, y)
  log_model = LinearRegression()
  log_model.fit(X_train, y_train)
  training_rmse = rmse_score(log_model, X_train, y_train)
  cv_rmse = np.mean(cross_val_score(log_model, X_train, y_train,␣
↪scoring=rmse_score, cv=5))
  result += [training_rmse, cv_rmse]
  if test_rmse:
    test_rmse = rmse_score(log_model, X_train, y_train)
    result += [test_rmse]
  return result
```

[ ]:
```
training_rmse = {}
cv_rmse = {}
for county in random_30_counties:
  training_rmse[county] = log_model(county, "num_of_confirmed")[0][0]
  cv_rmse[county] = log_model(county, "num_of_confirmed")[1]
```

[ ]:
```
np.mean(list(training_rmse.values())), np.mean(list(cv_rmse.values()))
```

[ ]:
```
(0.01852341568708034, 0.030265245743082633)
```

4.6 Model extension Can we use this model to predict the number of death cases?

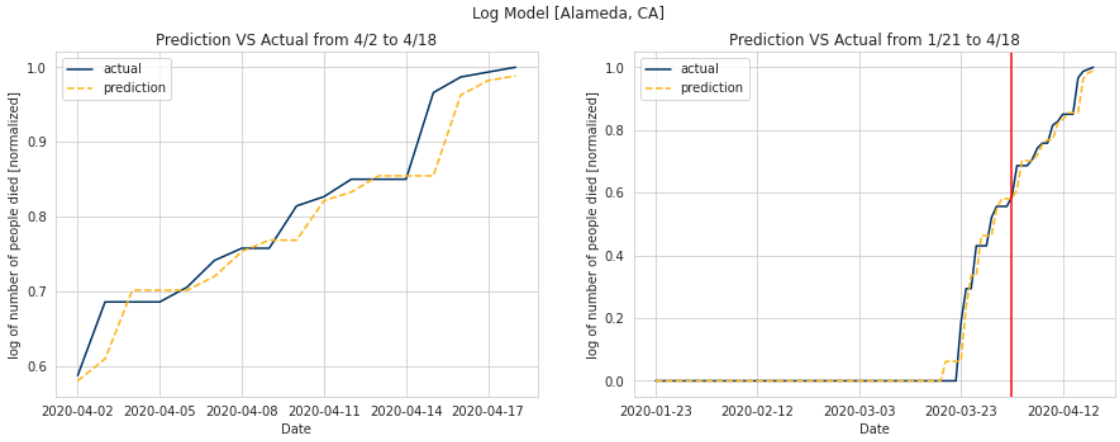
[ ]:
```
training_rmse = {}
cv_rmse = {}
for county in random_30_counties:
  training_rmse[county] = log_model(county, "num_of_death")[0][0]
  cv_rmse[county] = log_model(county, "num_of_death")[1]
```

[ ]:
```
np.mean(list(training_rmse.values())), np.mean(list(cv_rmse.values()))
```

[ ]:
```
(0.0033555487067026307, 0.004980244950852312)
```

[ ]:
```
alameda_death = time_series_data(84006001, "num_of_death")
X,y = time_series_X_y("log", "num_of_death", alameda_death)
X_train, X_test, y_train, y_test = time_series_split(X, y)
log_model = LinearRegression()
log_model.fit(X_train, y_train);
```

[ ]:
```
train_predictions = log_model.predict(X_train)
test_predictions = log_model.predict(X_test)
x_axis = alameda_death['Date'].astype(str).unique()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (15, 5))
ax1.plot(x_axis[70:], y_test, c = "#003262")
ax1.plot(x_axis[70:], test_predictions, c = "#FDB515", linestyle = '--')
ax1.set(xticks = (x_axis[70:][np.arange(0,18,3)]),
        title = "Prediction VS Actual from 4/2 to 4/18")
ax1.legend(['actual', 'prediction'])


ax2.plot(x_axis, y, c = "#003262")
ax2.plot(x_axis, np.concatenate((train_predictions,test_predictions)), c =␣
↪"#FDB515", linestyle = '--' )
ax2.set(xticks = (x_axis[np.arange(0,88,20)]),
        title = "Prediction VS Actual from 1/21 to 4/18")
ax2.axvline(x='2020-04-02', c = 'r')
ax2.legend(['actual', 'prediction'])

plt.setp([ax1,ax2], xlabel = 'Date', ylabel = "log of number of people died␣
↪[normalized]")
fig.suptitle('Log Model [Alameda, CA]');
```



Log model also fits death data pretty well.

7 V. Summary

For the time fixed model, predictions on the number of death cases due to COVID-19 by the Linear Regression, Ridge Regression, and Lasso Regression do not perform well, which indicates that characteristics of a county do not explain variations in number of confirmed and death cases across counties in the United States. For the county fixed model, the log model works better than the polynomial model for prediction on the number of confirmed cases. And we use the log model to predict for the number of death cases, it also makes good predictions.