```
import requests
import pandas as pd
from matplotlib import pyplot as plt
from scipy.stats import gamma
import numpy as np
from datetime import date, datetime as dt
import c3aidatalake


print("pandas version", pd.__version__)
assert pd.__version__[0] >= "1", "To use this notebook, upgrade to the newest version of pand

     pandas version 1.1.4
```

UNUSED Fetch Historic Policy Data

[ ] ↳ *8 cells hidden*

UNUSED Needs to find max datetime in a list but can't

[ ] ↳ *1 cell hidden*

UNUSED Find policy

[ ] ↳ *4 cells hidden*

Final: Pre-processing Survey Data

```
states = [
  'Alabama_UnitedStates','Alaska_UnitedStates','Arizona_UnitedStates',
  'Arkansas_UnitedStates','California_UnitedStates','Colorado_UnitedStates',
  'Connecticut_UnitedStates','Delaware_UnitedStates','DistrictofColumbia_UnitedStates',
  'Florida_UnitedStates','Georgia_UnitedStates','Hawaii_UnitedStates',
  'Idaho_UnitedStates','Illinois_UnitedStates','Indiana_UnitedStates',
  'Iowa_UnitedStates','Kansas_UnitedStates','Kentucky_UnitedStates',
  'Louisiana_UnitedStates','Maine_UnitedStates','Maryland_UnitedStates',
  'Massachusetts_UnitedStates','Michigan_UnitedStates','Minnesota_UnitedStates',
  'Mississippi_UnitedStates','Missouri_UnitedStates','Montana_UnitedStates',
  'Nebraska_UnitedStates','Nevada_UnitedStates','NewHampshire_UnitedStates',
  'NewJersey_UnitedStates','NewMexico_UnitedStates','NewYork_UnitedStates',
  'NorthCarolina_UnitedStates','NorthDakota_UnitedStates','Ohio_UnitedStates',
  'Oklahoma_UnitedStates','Oregon_UnitedStates','Pennsylvania_UnitedStates',
  'PuertoRico_UnitedStates','RhodeIsland_UnitedStates','SouthCarolina_UnitedStates',
  'SouthDakota_UnitedStates','Tennessee_UnitedStates','Texas_UnitedStates',
  'Utah_UnitedStates','Vermont_UnitedStates','Virginia_UnitedStates',
  'Washington_UnitedStates','WestVirginia_UnitedStates','Wisconsin_UnitedStates',
  'Wyoming_UnitedStates']
import re
#getting case count data
metrics = [
    "JHU_ConfirmedCases",
    "JHU_ConfirmedDeaths"
]

complete_timeseries = c3aidatalake.evalmetrics(
    "outbreaklocation",
    {
        "spec" : {
            "ids" : states,
            "expressions" : metrics,
            "start" : "2020-04-15",
            "end" : "2020-07-01",
            "interval" : "DAY",
        }
    },
    get_all = True
)
state_from_location = lambda x: "_".join(x.split('_')[-2:]).replace("_UnitedStates", "")
def reshapeTimeseries(timeseries_df):

    reshaped_ts = pd.melt(
        timeseries_df,
        id_vars=['dates'],
        value_vars=[x for x in timeseries_df.columns if re.match('.*\.data', x)]
    ).rename(columns={"value": "data", "dates": "date"})

    reshaped_ts["state"] = (
        reshaped_ts["variable"]
        .str.replace("\..*", "")
        .apply(state_from_location)
    )

    reshaped_ts["metric"] = (
        reshaped_ts["variable"]
        .str.replace(".*UnitedStates\.", "")
        .str.replace("\..*", "")
    )


    return reshaped_ts
state_timeseries = reshapeTimeseries(complete_timeseries)
state_timeseries.head()
state_cases = (
    state_timeseries
    .loc[state_timeseries.date > '2020-03-10']
    .groupby(['date', 'state', 'metric'])['data']
    .sum()
    .unstack('metric')
    .reset_index()
)
state_cases['death_rate'] = state_cases.apply(
    lambda x: 0 if x.JHU_ConfirmedCases == 0
    else x.JHU_ConfirmedDeaths / x.JHU_ConfirmedCases,
    axis=1
)

#obtaining census data from 2018
population_limits = (
    f"contains(parent, 'UnitedStates') &&" # US data
    "gender == 'Male/Female' && year == 2018 && origin == 'United States Census'" # From 2018
)

census = c3aidatalake.fetch(
    "populationdata",
    {
      "spec": {
        "filter": population_limits
      }
    },
    get_all = True
)

census['state'] = census['parent.id'].apply(state_from_location)
census = census.rename(columns={'parent.id': 'location'})
census_cols = [
    "populationAge",
    "value",
    "location",
    "state"
]

census_by_state = (
    census[census_cols]
    .loc[census.state.isin(map(lambda x: x.replace("_UnitedStates", ""), states))]
    .groupby(["state", "populationAge"])['value']
    .sum()
    .unstack("populationAge")
    .reset_index()
)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-5-7dd714698505> in <module>()
     36         }
     37     },
---> 38     get_all = True
     39 )
     40 state_from_location = lambda x: "_".join(x.split('_')
[-2:]).replace("_UnitedStates", "")
```

11 frames

```
/usr/local/lib/python3.6/dist-packages/numpy/core/numerictypes.py in issubdtype(arg1,
arg2)
    391     """
    392     if not issubclass_(arg1, generic):
--> 393         arg1 = dtype(arg1).type
    394     if not issubclass_(arg2, generic):
```

```
# Fetch participants who are located in California
survey = c3aidatalake.fetch(
    "surveydata",
    {
        "spec": {
            # "filter": "location == 'California_UnitedStates'"
        }
    },
    get_all = True
)



# Sorting survey data by state alphabetical
sorted_survey = survey.sort_values('location.id')
#  sorted_survey
loc= survey['location.id']
# l



sorted_survey_filtered = sorted_survey[sorted_survey['coronavirusIntent_Mask'].notnull()]


# sorting all rows with NaN in them
sorted_survey_filtered_v2 = sorted_survey.copy()
for col in sorted_survey.columns:
  sorted_survey_filtered_v2 = sorted_survey.loc[sorted_survey[col].notnull()]

# Y label data 0-100

# Get rid of columns we dont want example


# filter data to only after May 5, change F/T to 0/1
cleaned_survey_data = sorted_survey_filtered_v2.loc[sorted_survey_filtered_v2['startTime'] >
cleaned_survey_data = sorted_survey_filtered_v1.loc[sorted_survey_filtered_v1['startTime'] >
cleaned_survey_data[['coronaSimilarFlu', 'coronaOnlyElderly','youngInvulnerable','elderlyMore
                     'ethnicitySpreadsCovid','allSpreadCovid','nonNativesSpreadCovid','asympt
                     'infectFromAnimal']]*=1
# cleaned_survey_data


cleaned_survey_data = cleaned_survey_data.drop(columns = ["id","birthYear2020","coronavirusIn
                                                "coronavirusIntent_WashHands","coronavirusLocalC
                                               "zipcodePrefix"])
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3069: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  self[k1] = value[k2]
```

```
cleaned_survey_data.head(6)
```

	coronavirusConcern	coronavirusEmployment	coronavirusIntent_Mask	coronavirusSym
489	5.9	now-full	89.0	
136	5.2	was-full	72.0	
1339	5.6	now-jobless	62.0	muscl
1091	4.0	now-jobless	100.0	
1627	0.0	now-retired	10.0	
1757	10.0	was-disabled	100.0	

```
survey2=survey
survey2['startTime'] = pd.to_datetime(survey['startTime'])
survey2['startTime'] = survey['startTime'].apply(lambda t: t.replace(second=0))
survey2['startTime'] = survey['startTime'].apply(lambda t: t.replace(minute=0))
survey2['startTime'] = survey['startTime'].apply(lambda t: t.replace(hour=0))
surveydates= survey2['startTime'].value_counts()

import datetime
```

```
census_by_state2=census_by_state
census_and_cases = pd.merge(left=census_by_state2[['state','Median','Total']], right = state_
census_and_cases['state'] =  census_and_cases['state'].astype(str) + '_UnitedStates'

case_dates = list(census_and_cases['date'].value_counts().to_frame().index.values)
main_list = list(set(case_dates)-set(surveydates.to_frame().index.values))
for i in main_list:
  census_and_cases = census_and_cases[census_and_cases['date'] != i]

census_num = census_and_cases.select_dtypes(include=[np.number])
normalized_census = ((census_num-census_num.min())/(census_num.max()-census_num.min()))
census_and_cases[normalized_census.columns] = normalized_census
census_and_cases.head(6)
```

	state	Median	Total	date	JHU_ConfirmedCases	JHU_ConfirmedDeat
14	Alabama_UnitedStates	0.273277	0.110575	2020-04-29	0.016861	0.0080
15	Alabama_UnitedStates	0.273277	0.110575	2020-04-30	0.017313	0.0086
21	Alabama_UnitedStates	0.273277	0.110575	2020-05-06	0.021435	0.0109
30	Alabama_UnitedStates	0.273277	0.110575	2020-05-15	0.028332	0.0154

```
def value_to_col(dataframe, policy):
    temp = dataframe[policy].fillna(0)
    dfcount = temp.value_counts()
    options = list(dfcount.to_frame().index.values)
    for i in options:
        dataframe.loc[:,policy+"_"+i]=np.where(temp.str.contains(i),1,0)
    return dataframe

cleaned_survey_data2=cleaned_survey_data[['education','religion','ethnicity','gender']]
cleaned_survey_data2.loc[cleaned_survey_data2['education'].str.contains('highschool'), 'educa
cleaned_survey_data2=value_to_col(cleaned_survey_data2,"religion")
cleaned_survey_data2=value_to_col(cleaned_survey_data2,"ethnicity")
cleaned_survey_data2=value_to_col(cleaned_survey_data2,"education")
#cleaned_survey_data2=value_to_col(cleaned_survey_data2,"coronavirusEmployment")

cleaned_survey_data2.loc[cleaned_survey_data2['gender'].str.contains('female'), 'gender'] = '
temp = cleaned_survey_data2.gender.fillna("0")

cleaned_survey_data2.loc[:,'female'] = np.where(temp.str.contains("feMale"),1,0)
cleaned_survey_data2.loc[:,'male'] = np.where(temp.str.contains("male"),1,0)
cleaned_survey_data2 = cleaned_survey_data2.drop(columns=['education', 'religion', 'ethnicity

cleaned_survey_data2.head(6)
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py:670: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  iloc._setitem_with_indexer(indexer, value)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py:1596: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  self.obj[key] = _infer_fill_value(value)
/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py:1743: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  isetter(ilocs[0], value)
/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py:1763: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  isetter(loc, value)
```

	religion_catholic	religion_nothing-in-particular	religion_something-else	religion_other-protestant	relig
489	0	1	0	0	
136	0	0	0	0	
1339	0	0	1	0	
1091	0	0	0	0	
1627	0	0	1	0	
1757	0	1	0	0	

```
concat = pd.concat([cleaned_survey_data2, cleaned_survey_data], axis=1)
print(len(concat))
concat.head(6)
```

```
16375
```

	religion_catholic	religion_nothing-in-particular	religion_something-else	religion_other-protestant	relig
489	0	1	0	0	
136	0	0	0	0	
1339	0	0	1	0	

```
policy_united_states = c3aidatalake.fetch(
  "locationpolicysummary",
  {
      "spec" : {
          "filter" : "contains(location.id, 'UnitedStates')",
          "include": "stayAtHome, mandatoryQuarantine, largeGatherings,schoolClosure,easingOr
          "limit" : -1
      }
  }
)

states_list = policy_united_states['id'].tolist()
print(states_list)
pd.set_option('display.max_colwidth', None)
policy_united_states.head(6)
```

```
['Alaska_UnitedStates_Policy', 'Arizona_UnitedStates_Policy', 'Arkansas_UnitedStates_Po
```

	easingOrder	stayAtHome	mandatoryQuarantine	largeGatherings	schoolClosure	emerge
0	Proceeding with Reopening	Lifted	All Travelers	Lifted	Closed for School Year	
1	New Restrictions Imposed	Lifted	Lifted	New Limit on Large Gatherings in Place	Closed for School Year	
2	Paused	No Action	Lifted	Lifted	Closed for School Year	
3	New			All Gatherings	Recommended	

```
states_list = policy_united_states['id'].tolist()
states_list.pop(0)
history_policies = []
for state in states_list:
  if state == 'United States_UnitedStates_Policy':
    continue
  try:
    policy_state = c3aidatalake.read_data_json(
        "locationpolicysummary",
        "allversionsforpolicy",
        {
        "this": {
            "id": state
        }
    }
    )
  except:
    print(state)

  history_policies = history_policies + [policy_state]

df = pd.DataFrame(list(history_policies[0][2].items()),columns = ['column1','column2']).T
df = df.rename(columns=df.iloc[0])
df = df.drop(df.index[0])

df3= df[['location', 'lastSavedTimestamp','stayAtHome', 'mandatoryQuarantine','largeGathering
print(len(df3))
for i in range(len(history_policies)):
    for j in range(len(history_policies[i])):
        df2= pd.DataFrame(list(history_policies[i][j].items()),columns = ['column1','column2'
        df2 = df2.rename(columns=df2.iloc[0])
        df2 = df2.drop(df2.index[0])
        df2=df2[['location', 'lastSavedTimestamp','stayAtHome', 'mandatoryQuarantine','largeG
        df3 = df3.append(df2)

dfcount = df3['lastSavedTimestamp'].value_counts()
df3.head(6)
```

```
     Connecticut_UnitedStates_Policy
     Delaware_UnitedStates_Policy
     Georgia_UnitedStates_Policy
     Indiana UnitedStates Policy
```

```
df3= df[['location', 'lastSavedTimestamp','stayAtHome', 'mandatoryQuarantine','largeGathering
for i in range(len(history_policies)):
    for j in range(len(history_policies[i])):
        df2= pd.DataFrame(list(history_policies[i][j].items()),columns = ['column1','column2'
        df2 = df2.rename(columns=df2.iloc[0])
        df2 = df2.drop(df2.index[0])
        df2=df2[['location', 'lastSavedTimestamp','stayAtHome', 'mandatoryQuarantine','largeG
        df3 = df3.append(df2)

def value_to_col(dataframe, policy):
    temp = dataframe[policy].fillna(0)
    dfcount = temp.value_counts()
    options = list(dfcount.to_frame().index.values)
    for i in options:
        dataframe.loc[:,policy+i]=np.where(temp.str.contains(i),1,0)
    return dataframe

import warnings
warnings.filterwarnings("ignore", 'This pattern has match groups')
df6 = value_to_col(df3, "stayAtHome")
df6 = value_to_col(df3, "mandatoryQuarantine")
df6 = value_to_col(df3, "easingOrder")
df6 = value_to_col(df3, "emergencyDeclaration")
df6 = value_to_col(df3, "largeGatherings")

df3 = df3.reset_index()
a=[]
for i in range(len(df3)):
    a.append(df3.loc[i,'location'].get('id'))
df3['location.id'] = a
df3['lastSavedTimestamp'] = pd.to_datetime(df3['lastSavedTimestamp'])
df3['lastSavedTimestamp']= df3['lastSavedTimestamp'].dt.strftime('%Y-%m-%d %H:%M:%S')
df3['lastSavedTimestamp'] = pd.to_datetime(df3['lastSavedTimestamp'])
df3['lastSavedTimestamp'] = df3['lastSavedTimestamp'].apply(lambda t: t.replace(second=0))
df3['lastSavedTimestamp'] = df3['lastSavedTimestamp'].apply(lambda t: t.replace(minute=0))
df3['lastSavedTimestamp'] = df3['lastSavedTimestamp'].apply(lambda t: t.replace(hour=0))
#print(df3.loc[0,'lastSavedTimestamp']<df3.loc[1,'lastSavedTimestamp'])
#print(df3['lastSavedTimestamp'].value_counts())
print(len(df3))
df3=df3.loc[df3.astype(str).drop_duplicates().index]
df3.head(6)
```

```
280
```

	index	location	lastSavedTimestamp	stayAtHome	mandatoryQuarantine
0	column2	{'id': 'Arizona_UnitedStates'}	2020-05-29	Lifted	Lifted
1	column2	{'id': 'Arizona_UnitedStates'}	2020-09-12	Lifted	Lifted
4	column2	{'id': 'Arizona_UnitedStates'}	2020-05-29	Statewide	From Certain States
5	column2	{'id': 'Arizona_UnitedStates'}	2020-05-05	Statewide	From Certain States

```
concat[["coronavirusConcern","hasCoronavirusBelief","politicalBelief","politicalParty","relig
concat['startTime'] = pd.to_datetime(concat['startTime'])
concat['startTime']= concat['startTime'].dt.strftime('%Y-%m-%d %H:%M:%S')
concat['startTime'] = pd.to_datetime(concat['startTime'])
concat['startTime'] = concat['startTime'].apply(lambda t: t.replace(second=0))
concat['startTime'] = concat['startTime'].apply(lambda t: t.replace(minute=0))
concat['startTime'] = concat['startTime'].apply(lambda t: t.replace(hour=0))
concat.head(6)
```

	religion_catholic	religion_nothing-in-particular	religion_something-else	religion_other-protestant	relig
489	0	1	0	0	
136	0	0	0	0	
1339	0	0	1	0	
1091	0	0	0	0	
1627	0	0	1	0	
1757	0	1	0	0	

```
concat2=concat.copy()
concat2['startTime'] = pd.to_datetime(concat2['startTime'], utc = True)
census_and_cases['date']=pd.to_datetime(census_and_cases['date'],utc=True)
concat2 = pd.merge(right=census_and_cases, left = concat2, right_on=['state','date'],left_on=
concat2.head(6)
```

```
concat2.head(6)
```

	religion_catholic	religion_nothing-in-particular	religion_something-else	religion_other-protestant	religion
0	0	1	0	0	
1	0	0	0	0	
2	0	1	0	0	
3	1	0	0	0	
4	0	0	0	0	
5	1	0	0	0	

```
final_df=concat2.copy()
final_df['startTime'] = pd.to_datetime(final_df['startTime'])
final_df['startTime'] = final_df['startTime'].dt.tz_localize(None)
```

```
temp = []
import datetime
time = list(pd.to_datetime(final_df['startTime']))
df3=df3.rename(columns={"lastSavedTimestamp": "time"})
print(df3.head(6))
df3 = df3[df3['time'] < datetime.datetime(2020,9,12)]
```

```
     index  ...             location.id
0  column2  ...   Arizona_UnitedStates
1  column2  ...   Arizona_UnitedStates
4  column2  ...   Arizona_UnitedStates
5  column2  ...   Arizona_UnitedStates
7  column2  ...  Arkansas_UnitedStates
9  column2  ...  Arkansas_UnitedStates

[6 rows x 45 columns]
```

```
temp=[]
for i in range(len(final_df)):
  if (time[i].date()<datetime.date(2020,5,29)):
    temp.append(datetime.datetime(2020,5,5))
  elif ((time[i].date()>= datetime.date(2020,5,29)) & (time[i].date()<= datetime.date(2020,9,
    temp.append(datetime.datetime(2020,5,29))
  elif (time[i].date()>datetime.date(2020,9,12)):
    temp.append(datetime.datetime(2020,9,12))
```

```
df2 = pd.DataFrame(index=final_df.index)
df2['temp']=temp
final_df['time']=df2['temp']
final_df['Time_Diff'] = (pd.to_datetime(final_df.startTime) - pd.to_datetime(final_df.time)).
states = list(df3['location.id'].value_counts().to_frame().index.values)
states2= list(final_df['location.id'].value_counts().to_frame().index.values)
main_list = list(set(states2) - set(states))
for i in main_list:
  final_df = final_df[final_df['location.id'] != i]
df3=df3.drop_duplicates(subset=['location.id','time'])
```

```
final_df.head(6)
```

	religion_catholic	religion_nothing-in-particular	religion_something-else	religion_other-protestant	religi
318	1	0	0	0	
319	1	0	0	0	
320	0	0	0	1	
321	0	0	0	1	
322	0	0	1	0	
323	0	0	0	0	

```
merged =  final_df.merge(df3, on=['time','location.id'], how='left',sort=False)
numeric_final = merged.drop(columns = ['time','location.id','startTime','education','gender',
                                       'mandatoryQuarantine','stayAtHome','location','largeGa
numeric_final.apply(lambda numeric_final: pd.to_numeric(numeric_final, errors='coerce').notnu
numeric_final.head(6)
```

	religion_catholic	religion_nothing-in-particular	religion_something-else	religion_other-protestant	religion
0	1	0	0	0	
1	1	0	0	0	
2	0	0	0	1	
4	0	0	1	0	

```
numeric_final['Time_Diff']=(numeric_final['Time_Diff']-min(numeric_final['Time_Diff']))/(max(
```

```
numeric_final= numeric_final.drop(columns=['state','date'])
print(numeric_final.dtypes['stayAtHomeHigh-Risk Groups'])
```

```
int64
```

Training Data

```
numeric_final.head(1)
```

	religion_catholic	religion_nothing-in-particular	religion_something-else	religion_other-protestant	religion
0	1	0	0	0	

1 rows × 86 columns

```
import keras
from keras.models import Sequential
from keras.layers import Dense
import tensorflow as tf
from numpy import asarray
from sklearn.utils import shuffle
from sklearn.linear_model import LinearRegression
```

```
print(numeric_final.shape)
numeric_final.head(3)
```

```
(11014, 86)
```

	religion_catholic	religion_nothing-in-particular	religion_something-else	religion_other-protestant	religion
0	1	0	0	0	
1	1	0	0	0	

```
rows_with_nan = []
for index, row in numeric_final.iterrows():
    is_nan_series = row.isnull()
    if is_nan_series.any():
        rows_with_nan.append(index)

print(len(rows_with_nan))
```

```
0
```

```
numeric_final = numeric_final.drop(index = rows_with_nan)
```

```
y_labels = numeric_final["coronavirusIntent_Mask"]
X_model_features = numeric_final.drop(["coronavirusIntent_Mask"],axis = 1)
```

```
print(X_model_features.shape)
```

```
(11014, 85)
```

```
indices_header = y_labels -1
one_hot = tf.one_hot(indices_header,100,dtype = tf.uint8)
Y_model_labels = asarray(one_hot)
```

```
print(Y_model_labels)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

```
X_model_features, Y_model_labels = shuffle(X_model_features, Y_model_labels)
X_model_features, Y_model_labels = shuffle(X_model_features, Y_model_labels)
X_model_features, Y_model_labels = shuffle(X_model_features, Y_model_labels)
X_model_features, Y_model_labels = shuffle(X_model_features, Y_model_labels)
```

```
print(Y_model_labels)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 1]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 0]]
```

```
INPUT_DIM = X_model_features.shape[1]
print(INPUT_DIM)
LEARNING_RATE = 0.001
```

```
85
```

```
# Neural network
first_model = Sequential()

first_model.add(Dense(128, input_dim=INPUT_DIM, activation="relu"))
first_model.add(Dense(64, activation="relu"))
first_model.add(Dense(100, activation="softmax"))
opt = keras.optimizers.Adam(learning_rate=LEARNING_RATE)
first_model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
first_model.summary()
weights = first_model.get_weights()
#print(weights)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 128)               11008
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 100)               6500
=================================================================
Total params: 25,764
Trainable params: 25,764
Non-trainable params: 0
_________________________________________________________________
```

```
history_model = first_model.fit(X_model_features, Y_model_labels, validation_split=0.2,epochs
```

```
Epoch 1/10
1469/1469 [==============================] - 3s 2ms/step - loss: 3.3629 - accuracy: 0.32
Epoch 2/10
1469/1469 [==============================] - 2s 2ms/step - loss: 4.1571 - accuracy: 0.29
Epoch 3/10
1469/1469 [==============================] - 2s 2ms/step - loss: 19.6162 - accuracy: 0.1
Epoch 4/10
```

```
1469/1469 [==============================] - 2s 2ms/step - loss: 59.2948 - accuracy: 0.:
Epoch 5/10
1469/1469 [==============================] - 2s 2ms/step - loss: 108.4118 - accuracy: 0
Epoch 6/10
1469/1469 [==============================] - 2s 2ms/step - loss: 166.6763 - accuracy: 0
Epoch 7/10
1469/1469 [==============================] - 2s 2ms/step - loss: 242.5825 - accuracy: 0
Epoch 8/10
1469/1469 [==============================] - 2s 2ms/step - loss: 334.9890 - accuracy: 0
Epoch 9/10
1469/1469 [==============================] - 2s 2ms/step - loss: 427.2556 - accuracy: 0
Epoch 10/10
1469/1469 [==============================] - 2s 2ms/step - loss: 557.4230 - accuracy: 0
```

```
# Test out multivariate regression
```

```
total_data = numeric_final.copy()
total_data=numeric_final.drop(columns=['religion_atheist','ethnicity_white','education_colleg
                                      'mandatoryQuarantineNo Action','easingOrderYes','largeG
total_data['Time_Diff']=(total_data['Time_Diff']-min(total_data['Time_Diff']))/(max(total_dat
```

```
# Shuffle data around
total_data = total_data.sample(frac = 1).reset_index(drop=True)
```

```
print(len(total_data))
```

```
11014
```

```
test_set = total_data.tail(int(len(total_data)/10))
train_set = total_data.head(-int(len(total_data)/10))
```

```
train_set
```

	religion_catholic	religion_nothing-in-particular	religion_something-else	religion_other-protestant	relig
0	0	1	0	0	
1	1	0	0	0	
2	0	1	0	0	
3	0	0	0	1	
4	1	0	0	0	

```
Y_train = train_set[["coronavirusIntent_Mask"]]
X_train = train_set.drop(columns = ["coronavirusIntent_Mask"])
Y_test = test_set["coronavirusIntent_Mask"]
X_test = test_set.drop(columns = ["coronavirusIntent_Mask"])
```

9910	0	1	0	0

```
headers = X_train.head(0)
```

```
lr = LinearRegression()
lr.fit(X_train,Y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
lr.coef_[0]
```

```
array([-2.22277211e+00, -2.96211215e+00, -5.41424089e+00, -4.05378929e+00,
       -5.73697722e+00, -1.73035274e+00, -2.81346879e+00, -1.07676549e+01,
       -4.29295581e+00, -7.48817807e+00, -5.66637381e+00, -5.12314374e+00,
        2.32671371e+00,  2.32695687e+00,  2.05907633e+00,  6.22458945e+00,
       -2.43724843e+00, -1.63682687e+00,  2.33647393e+00, -2.21284310e+00,
        4.06437805e+00,  3.01926109e+00,  4.67394846e+01, -6.60183694e-01,
        1.40482301e-01,  3.77692326e+00,  8.93747459e+00, -1.30385142e+01,
       -4.60991693e+00, -4.20303571e+00, -1.49817682e+00, -6.73339401e-02,
        5.20004047e+00,  2.58619043e+00,  3.56114445e+00,  1.86019506e+00,
       -1.96036443e+00,  5.72342528e+00,  5.83840036e-01,  1.26408711e+00,
       -1.64233002e-01,  1.47315166e-01, -7.90577845e+00,  5.74992861e+01,
       -1.66482622e+00, -1.10129766e-01, -2.52456331e+00, -5.25391817e+00,
        4.43886393e+00, -4.90339053e+00, -1.11022302e-16, -3.55271368e-15,
       -5.32907052e-15,  1.98545326e-01, -3.03806537e+00,  2.83536321e+00,
       -9.16664011e-02,  4.35579546e-01,  8.88178420e-16, -1.77635684e-15,
       -6.20260474e-01,  8.88178420e-16,  2.22044605e-16, -8.88178420e-16,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00, -1.72910554e+00,
       -6.71542625e+00, -1.46247311e+00, -2.82262364e+00,  0.00000000e+00,
        0.00000000e+00, -2.38487853e+00, -2.17482239e+00, -7.04635160e-01,
        5.30055071e-01,  0.00000000e+00])
```

```
pred = lr.predict(X_test)
```

```
heads = []
for col in headers.columns:
  heads = heads + [col]


coeff = pd.DataFrame([lr.coef_[0]],columns = heads)
sorted_coeff_survey = coeff.sort_values(by=0, ascending=False, axis=1)
sorted_coeff_survey
from google.colab import files
#sorted_coeff.to_csv('filename.csv')
#files.download("filename.csv")


print(Y_test)
```

```
     9913      97.0
     9914      97.0
     9915      18.0
     9916      25.0
     9917       3.0
              ...
     11009    100.0
     11010    100.0
     11011    100.0
     11012     61.0
     11013     97.0
     Name: coronavirusIntent_Mask, Length: 1101, dtype: float64
```

```
tweets=pd.read_csv("https://raw.githubusercontent.com/alicezhang09/Cell-Painting/master/filte
print(tweets.columns)
# dtime = tweet['created_at']
check=tweets
count_tweets= check['user_location'].value_counts()
check["created_at"] = check["created_at"].astype('datetime64[ns]')
check["created_at"] = check.created_at.dt.to_pydatetime()
check["created_at"] = check["created_at"].apply(lambda t: t.replace(second=0))
check["created_at"] = check["created_at"].apply(lambda t: t.replace(minute=0))
check["created_at"] = check["created_at"].apply(lambda t: t.replace(hour=0))
census_and_cases = pd.merge(left=census_by_state2[['state','Median','Total']], right = state_
census_and_cases['date'] = census_and_cases['date'].dt.strftime('%Y-%m-%d')
count_tweet_dates = check['created_at'].value_counts()
check['created_at']=check['created_at'].dt.strftime('%Y-%m-%d')
print(df3['time'])
df3['time']=pd.to_datetime(df3['time'])
tweets_and_cases= pd.merge(right=census_and_cases, left = check, right_on=['state','date'],le

tweets_num = tweets_and_cases.select_dtypes(include=[np.number])
tweets_norm= (tweets_num-tweets_num.min())/(tweets_num.max()-tweets_num.min())
tweets_and_cases[tweets_norm.columns] = tweets_norm
tweets_and_cases = tweets_and_cases.drop(columns=['user_location', 'user_verified', 'id'])

temp2 = []
```

```
time2 = list(pd.to_datetime(tweets_and_cases['date']))
for i in range(len(tweets_and_cases)):
  if (time2[i].date()<datetime.date(2020,5,29)):
    temp2.append(datetime.datetime(2020,5,5))
  else:
    temp2.append(datetime.datetime(2020,5,29))

tweet_copy = pd.DataFrame(index=tweets_and_cases.index)
tweet_copy['temp']=temp2
tweets_and_cases['time']=tweet_copy['temp']
tweets_and_cases['Time_Diff'] = (pd.to_datetime(tweets_and_cases.date) - pd.to_datetime(tweet
tweets_and_cases=tweets_and_cases.dropna()
tweets_and_cases['state']=tweets_and_cases['state']+"_UnitedStates"
states4 = list(df3['location.id'].value_counts().to_frame().index.values)
states5= list(tweets_and_cases['state'].value_counts().to_frame().index.values)
main_list = list(set(states5) - set(states4))
for i in main_list:
  tweets_and_cases = tweets_and_cases[tweets_and_cases['state'] != i]

tweets_and_cases = tweets_and_cases.rename(columns={'state': 'location.id'})
merged =  tweets_and_cases.merge(df3, on=['time','location.id'], how='left',sort=False)
```

```
Index(['created_at', 'id', 'retweet_count', 'user_followers_count',
       'user_location', 'user_verified', 'sentiment'],
      dtype='object')
0     2020-05-29
5     2020-05-05
9     2020-05-29
11    2020-05-05
14    2020-05-29
         ...
267   2020-05-05
271   2020-05-29
273   2020-05-05
277   2020-05-29
279   2020-05-05
Name: time, Length: 68, dtype: datetime64[ns]
```

```
#merged=merged.drop(columns=['created_at','location.id','date','stayAtHome', 'mandatoryQuaran
merged['Time_Diff']=(merged['Time_Diff']-min(merged['Time_Diff']))/(max(merged['Time_Diff'])-
merged.head(6)
```

	retweet_count	user_followers_count	sentiment	Median	Total	JHU_ConfirmedCase
0	0.008567	0.000536	0.500000	0.407437	0.066148	0.04988
2	0.115007	0.000014	0.500000	0.285881	0.531605	0.14575
4	0.000046	0.000021	0.421591	0.223469	1.000000	0.29525

```
# Test out multivariate regression

total_data = merged

# Shuffle data around
total_data = total_data.sample(frac = 1).reset_index(drop=True)
```

```
print(len(total_data))
```

```
     53383
```

```
test_set = total_data.tail(int(len(total_data)/10))
train_set = total_data.head(-int(len(total_data)/10))
```

```
train_set
```

```
Y_train = train_set[["sentiment"]]
X_train = train_set.drop(columns = ["sentiment"])
Y_test = test_set["sentiment"]
X_test = test_set.drop(columns = ["sentiment"])
```

```
    1        0.003183              0.000183    0.300000   0.223409   1.000000            0.3
```

```
headers = X_train.head(0)
```

```
    3        0.000000              0.000001    0.300000   0.365876   0.241620            0.1
```

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train,Y_train)
```

```
     LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
lr.coef_[0]
```

```
     array([-1.75799176e-01,  2.98918053e-03,  1.53849667e-03,  2.42311255e-03,
            -2.05285537e-02,  1.21065213e-02,  6.92713515e-03,  4.15462161e-03,
            -2.29529107e-03, -4.09164166e-03,  4.30741046e-03, -5.71218286e-03,
             7.79170514e-03, -1.56125113e-17,  1.64798730e-17, -1.04083409e-17,
            -3.43322768e-03,  3.12303941e-03, -3.94780250e-03, -4.58252728e-03,
             1.79548684e-03,  7.04503121e-03,  1.30104261e-18,  8.67361738e-19,
             1.18803137e-03, -1.18803137e-03,  0.00000000e+00,  0.00000000e+00,
             0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
            -1.70260205e-03, -2.43056690e-03,  2.41810173e-04,  1.69868725e-03,
             5.08096414e-03,  0.00000000e+00,  0.00000000e+00, -2.42135653e-03,
            -4.82838642e-03,  7.92246681e-04, -2.68358088e-03,  0.00000000e+00])
```

```
pred = lr.predict(X_test)
```

```
heads = []
for col in headers.columns:
  heads = heads + [col]
```

```
coeff = pd.DataFrame([lr.coef_[0]],columns = heads)
sorted_coeff_tweets = coeff.sort_values(by=0, ascending=False, axis=1)
sorted_coeff_tweets
```

	JHU_ConfirmedDeaths	stayAtHomeHigh-Risk Groups	mandatoryQuarantineRolled Back to Certain States	death_rate	largeGa
0	0.012107	0.007792	0.007045	0.006927	

```
sorted_coeff_survey
```

	JHU_ConfirmedDeaths	coronavirusConcern	religiosity	ethnicity_asian	asymptomaticS
0	57.499286	46.739485	8.937475	6.224589	5.7

```
list(sorted_coeff_survey)[0:15]
```

```
['JHU_ConfirmedDeaths',
 'coronavirusConcern',
 'religiosity',
 'ethnicity_asian',
 'asymptomaticSpread',
 'coronaAllHospitalize',
 'stayAtHomeRolled Back to High Risk Groups',
 'female',
 'politicalParty',
 'ethnicitySpreadsCovid',
 'male',
 'mandatoryQuarantineLifted',
 'coronaKillsMost',
 'education_postgrad',
 'ethnicity_hispanic-latino']
```

```
list(sorted_coeff_survey)[-15:]
```

```
['religion_other-protestant',
 'coronaOnlyElderly',
 'religion_muslim',
 'coronaSimilarFlu',
 'stayAtHomeHigh-Risk Groups',
 'religion_hindu',
 'stayAtHomeNo Action',
 'religion_something-else',
 'religion_buddhist',
 'religion_evangelical-protestant',
 'largeGatheringsLifted',
 'religion_orthodox',
 'JHU_ConfirmedCases',
 'religion_mormon',
 'trumpApproval']
```

```
list(sorted_coeff_tweets)[:15]
```

```
['JHU_ConfirmedDeaths',
 'stayAtHomeHigh-Risk Groups',
 'mandatoryQuarantineRolled Back to Certain States',
 'death_rate',
 'largeGatheringsNo Action',
 'stayAtHomeNo Action',
 'Time_Diff',
 'mandatoryQuarantineFrom Certain States',
 'user_followers_count',
 'Total',
 'mandatoryQuarantineAll Air Travelers',
 'largeGatheringsExpanded to New Limit Above 25',
 'Median',
 'easingOrderNo',
 'largeGatheringsExpanded to >10 People Prohibited']
```

```
list(sorted_coeff_tweets)[-15:]
```

```
['stayAtHomeHigh-risk Groups',
 'easingOrderYes',
 'largeGatherings>10 People Prohibited',
 'stayAtHomeStatewide',
 'largeGatheringsExpanded to New Limit Below 25',
 'largeGatheringsAll Gatherings Prohibited',
 'largeGatheringsExpanded to New Limit of 25',
 'mandatoryQuarantineNo Action',
 'mandatoryQuarantineAll Travelers',
 'stayAtHomeLifted',
 'mandatoryQuarantineLifted',
 'largeGatheringsOther',
 'stayAtHomeRolled Back to High Risk Groups',
 'JHU_ConfirmedCases',
 'retweet_count']
```

```
pd.concat([sorted_coeff_survey,sorted_coeff_tweets],join='inner')
```

	JHU_ConfirmedDeaths	stayAtHomeRolled Back to High Risk Groups	mandatoryQuarantineLifted	largeGatheringsExp to New Limit
0	57.499286	4.438864	2.835363	0.5
0	0.012107	-0.005712	-0.004583	-0.0

```
pd.concat([sorted_coeff_tweets,sorted_coeff_survey],join='inner')
```

	JHU_ConfirmedDeaths	stayAtHomeHigh-Risk Groups	mandatoryQuarantineRolled Back to Certain States	death_rate	largeGa
0	0.012107	0.007792	0.007045	0.006927	
0	57.499286	-4.903391	0.435580	-1.664826	

```
sorted_coeff_tweets.to_csv('tweets_coeff.csv')
files.download("tweets_coeff.csv")
sorted_coeff_survey.to_csv('survey_coeff.csv')
files.download("survey_coeff.csv")
```

Normalize numeric columns

↳ *5 cells hidden*

Parse Twitter Sentiment Data

New Section

[ ] ↳ *1 cell hidden*