In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
%matplotlib inline
import seaborn as sns
sns.set()
sns.set_style("darkgrid")
import warnings
warnings.filterwarnings("ignore")
import math
from sklearn.metrics import accuracy_score
import statsmodels.api as sm
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold
from sklearn import preprocessing
```

Question 1

The “Variable Description.xlsx” spreadsheet contains a list of variables that we’ll use for our analyses. Note that this is not a full list of all the variables in the dataset, although it’s close (we ignoring a few perfectly co-linear predictors). Filter the full set of variables in the dataset down to the the Opportunity Insights and PM COVID variables listed in the spreadsheet along with county, state, and deathspc.

In [2]:

```
vd_data = pd.read_excel("PPHA_30545_MP03-Variable_Description.xlsx")

vd_data['Variable'].unique

vd_variables = vd_data['Variable'].tolist()

print(vd_variables)
```

```
['casespc', 'deathspc', 'intersects_msa', 'cur_smoke_q1', 'cur_smoke_q2', 'cur_smoke_q3', 'cur_smoke_q4', 'bmi_obese_q1', 'bmi_obe
se_q2', 'bmi_obese_q3', 'bmi_obese_q4', 'exercise_any_q1', 'exercise_any_q2', 'exercise_any_q3', 'exercise_any_q4', 'brfss_mia',
'puninsured2010', 'reimb_penroll_adj10', 'mort_30day_hosp_z', 'adjmortmeas_amiall30day', 'adjmortmeas_chfall30day', 'med_prev_qual
_z', 'primcarevis_10', 'diab_hemotest_10', 'diab_eyeexam_10', 'diab_lipids_10', 'mammogram_10', 'cs00_seg_inc', 'cs00_seg_inc_pov2
5', 'cs00_seg_inc_aff75', 'cs_race_theil_2000', 'gini99', 'poor_share', 'inc_share_1perc', 'frac_middleclass', 'scap_ski90pcm', 'r
el_tot', 'cs_frac_black', 'cs_frac_hisp', 'unemp_rate', 'cs_labforce', 'cs_elf_ind_man', 'cs_born_foreign', 'mig_inflow', 'mig_out
```

```
flow', 'pop_density', 'frac_traveltime_lt15', 'hhinc00', 'median_house_value', 'ccd_exp_tot', 'score_r', 'cs_fam_wkidsinglemom',
'subcty_exp_pc', 'taxrate', 'tax_st_diff_top20', 'pm25', 'pm25_mia', 'summer_tmmx', 'summer_rmax', 'winter_tmmx', 'winter_rmax',
'bmcruderate']
```

In [3]:

```
covid_data = pd.read_csv('Data-Covid002.csv', encoding='ISO-8859-1')
covid_data.columns
covid_data.set_index(['county'], inplace = True)
vd_variables.append('state')
covid_data1 = covid_data[vd_variables[1:]]
```

In [4]:

```
covid_data1
```

Out[4]:

	deathspc	intersects_msa	cur_smoke_q1	cur_smoke_q2	cur_smoke_q3	cur_smoke_q4	bmi_obese_q1	bmi_obese_q2	bmi_obese_q3	bmi_obese
county										
Autauga	16.548864	1	0.333333	0.238095	0.208333	0.133333	0.375000	0.238095	0.260870	0.133
Baldwin	8.959118	1	0.268097	0.233503	0.167464	0.176991	0.298050	0.262467	0.193237	0.135
Barbour	6.609756	0	0.228571	0.250000	0.181818	0.111111	0.294118	0.571429	0.545455	0.277
Bibb	6.038192	1	0.244444	0.280000	0.181818	0.150000	0.466667	0.375000	0.190476	0.100
Blount	1.503713	1	0.304348	0.260870	0.352941	0.166667	0.347826	0.318182	0.529412	0.235
...	...	...	...	...	...	...	...	...	...	
Sweetwater	0.000000	0	0.349304	0.302658	0.227799	0.178484	0.247209	0.226943	0.254593	0.257
Teton	10.535728	0	0.263415	0.182018	0.092105	0.053913	0.127451	0.099338	0.113333	0.072
Uinta	0.000000	0	0.345538	0.237718	0.197943	0.130682	0.283019	0.247154	0.224274	0.235
Washakie	14.836295	0	0.260163	0.195652	0.083333	0.076923	0.283186	0.139706	0.265306	0.194
Weston	0.000000	0	0.318841	0.241935	0.264706	0.266667	0.181818	0.290323	0.218750	0.285

3107 rows × 62 columns

Question 2

Compute descriptive (summary) statistics for the subset of Opportunity Insights and PM COVID variables you filtered in previous question.

In [5]:
```
print(covid_data1.describe())
```

```
          deathspc  intersects_msa  cur_smoke_q1  cur_smoke_q2  cur_smoke_q3  \
count  3107.000000     3107.000000   3107.000000   3107.000000   3107.000000   
mean     23.790131        0.596717      0.212659      0.171048      0.134467   
std      67.852145        0.490636      0.149348      0.128130      0.132181   
min       0.000000        0.000000      0.000000      0.000000      0.000000   
25%       0.000000        0.000000      0.000000      0.000000      0.000000   
50%       3.802303        1.000000      0.250000      0.198718      0.142857   
75%      21.461759        1.000000      0.310931      0.250000      0.200000   
max    2279.610600        1.000000      1.000000      1.000000      1.000000   

       cur_smoke_q4  bmi_obese_q1  bmi_obese_q2  bmi_obese_q3  bmi_obese_q4  \
count   3107.000000   3107.000000   3107.000000   3107.000000   3107.000000   
mean       0.098316      0.239166      0.214580      0.209621      0.186739   
std        0.110110      0.165928      0.153237      0.175849      0.167227   
min        0.000000      0.000000      0.000000      0.000000      0.000000   
25%        0.000000      0.080128      0.000000      0.000000      0.000000   
50%        0.096535      0.272076      0.241590      0.223124      0.194118   
75%        0.148719      0.335532      0.304348      0.297220      0.266667   
max        1.000000      1.000000      1.000000      1.000000      1.000000   

       ...  subcty_exp_pc      taxrate  tax_st_diff_top20         pm25  \
count  ...    3107.000000  3107.000000        3106.000000  3107.000000   
mean   ...    2119.407531     0.023089           0.775634     8.371871   
std    ...     999.833466     0.013848           1.470989     2.565927   
min    ...       0.000000     0.000000           0.000000     0.000000   
25%    ...    1510.192750     0.014993           0.000000     6.309710   
50%    ...    1935.919400     0.020339           0.000000     8.784647   
75%    ...    2505.411100     0.027164           1.000000    10.483764   
max    ...   20541.918000     0.209907           7.220000    15.786018   

          pm25_mia  summer_tmmx  summer_rmax  winter_tmmx  winter_rmax  \
count  3107.000000  3107.000000  3107.000000  3107.000000  3107.000000   
mean      0.003540   303.126997    88.970517   280.404875    87.469432   
std       0.059405     3.173950     9.689271     6.597855     4.811207   
min       0.000000   290.455540    31.643282   264.693820    58.159798   
25%       0.000000   300.848035    88.052494   275.113020    85.093342   
50%       0.000000   303.290440    91.320313   280.154690    88.028793   
75%       0.000000   305.817430    94.812389   285.543750    90.747704   
```

```
max       1.000000   313.872680    99.778748   298.340360    97.672874

       bmcruderate
count   3107.00000
mean    1029.15597
std      248.38181
min      189.30000
25%      864.29999
50%     1036.30000
75%     1194.10000
max     1978.60000

[8 rows x 61 columns]
```

Question 3

Note that some variables have missing values. This causes problems when estimating the models. Normally we'd impute missing values by replacing them with their mean or median value, but to keep things simple, given the size of our data, you should drop all observations (rows) with missing values.

In [6]:

```
covid_data1.dropna(inplace=True)
```

In [7]:

```
covid_data1
```

Out[7]:

	deathspc	intersects_msa	cur_smoke_q1	cur_smoke_q2	cur_smoke_q3	cur_smoke_q4	bmi_obese_q1	bmi_obese_q2	bmi_obese_q3	bmi_obese
county										
Autauga	16.548864	1	0.333333	0.238095	0.208333	0.133333	0.375000	0.238095	0.260870	0.133
Baldwin	8.959118	1	0.268097	0.233503	0.167464	0.176991	0.298050	0.262467	0.193237	0.135
Barbour	6.609756	0	0.228571	0.250000	0.181818	0.111111	0.294118	0.571429	0.545455	0.277
Bibb	6.038192	1	0.244444	0.280000	0.181818	0.150000	0.466667	0.375000	0.190476	0.100
Blount	1.503713	1	0.304348	0.260870	0.352941	0.166667	0.347826	0.318182	0.529412	0.235
...	...	...	...	...	...	...	...	...	...	
Sweetwater	0.000000	0	0.349304	0.302658	0.227799	0.178484	0.247209	0.226943	0.254593	0.257

county	deathspc	intersects_msa	cur_smoke_q1	cur_smoke_q2	cur_smoke_q3	cur_smoke_q4	bmi_obese_q1	bmi_obese_q2	bmi_obese_q3	bmi_obese
Teton	10.535728	0	0.263415	0.182018	0.092105	0.053913	0.127451	0.099338	0.113333	0.072
Uinta	0.000000	0	0.345538	0.237718	0.197943	0.130682	0.283019	0.247154	0.224274	0.235
Washakie	14.836295	0	0.260163	0.195652	0.083333	0.076923	0.283186	0.139706	0.265306	0.194
Weston	0.000000	0	0.318841	0.241935	0.264706	0.266667	0.181818	0.290323	0.218750	0.285

2915 rows × 62 columns

Question 4

Create a separate dummy variable for each of the 48 states and the District of Columbia in the dataset (so you'll create 49 dummy variables in total, but dropping observations with missing values may reduce this number).

In [8]:

```
dummies = pd.get_dummies(covid_data1['state'])
covid_dummies = pd.concat([covid_data1, dummies], axis=1).reset_index()
```

In [9]:

```
covid_dummies
```

Out[9]:

	county	deathspc	intersects_msa	cur_smoke_q1	cur_smoke_q2	cur_smoke_q3	cur_smoke_q4	bmi_obese_q1	bmi_obese_q2	bmi_obese_q3	...
0	Autauga	16.548864	1	0.333333	0.238095	0.208333	0.133333	0.375000	0.238095	0.260870	...
1	Baldwin	8.959118	1	0.268097	0.233503	0.167464	0.176991	0.298050	0.262467	0.193237	...
2	Barbour	6.609756	0	0.228571	0.250000	0.181818	0.111111	0.294118	0.571429	0.545455	...
3	Bibb	6.038192	1	0.244444	0.280000	0.181818	0.150000	0.466667	0.375000	0.190476	...
4	Blount	1.503713	1	0.304348	0.260870	0.352941	0.166667	0.347826	0.318182	0.529412	...
...	...	...	...	...	...	...	...	...	...	...	...
2910	Sweetwater	0.000000	0	0.349304	0.302658	0.227799	0.178484	0.247209	0.226943	0.254593	...

	county	deathspc	intersects_msa	cur_smoke_q1	cur_smoke_q2	cur_smoke_q3	cur_smoke_q4	bmi_obese_q1	bmi_obese_q2	bmi_obese_q3	...
2911	Teton	10.535728	0	0.263415	0.182018	0.092105	0.053913	0.127451	0.099338	0.113333	...
2912	Uinta	0.000000	0	0.345538	0.237718	0.197943	0.130682	0.283019	0.247154	0.224274	...
2913	Washakie	14.836295	0	0.260163	0.195652	0.083333	0.076923	0.283186	0.139706	0.265306	...
2914	Weston	0.000000	0	0.318841	0.241935	0.264706	0.266667	0.181818	0.290323	0.218750	...

2915 rows × 110 columns

Question 5

Split the sample into a training set (80%) and a test set (20%). Be sure to set a random seed so you can replicate your work.

In [10]:

```
covid_dummies.drop([ 'county','state'], axis =1, inplace = True)

predictors = covid_dummies.columns.tolist()

print(predictors)
```

```
['deathspc', 'intersects_msa', 'cur_smoke_q1', 'cur_smoke_q2', 'cur_smoke_q3', 'cur_smoke_q4', 'bmi_obese_q1', 'bmi_obese_q2', 'bm
i_obese_q3', 'bmi_obese_q4', 'exercise_any_q1', 'exercise_any_q2', 'exercise_any_q3', 'exercise_any_q4', 'brfss_mia', 'puninsured2
010', 'reimb_penroll_adj10', 'mort_30day_hosp_z', 'adjmortmeas_amiall30day', 'adjmortmeas_chfall30day', 'med_prev_qual_z', 'primca
revis_10', 'diab_hemotest_10', 'diab_eyeexam_10', 'diab_lipids_10', 'mammogram_10', 'cs00_seg_inc', 'cs00_seg_inc_pov25', 'cs00_se
g_inc_aff75', 'cs_race_theil_2000', 'gini99', 'poor_share', 'inc_share_1perc', 'frac_middleclass', 'scap_ski90pcm', 'rel_tot', 'cs
_frac_black', 'cs_frac_hisp', 'unemp_rate', 'cs_labforce', 'cs_elf_ind_man', 'cs_born_foreign', 'mig_inflow', 'mig_outflow', 'pop_
density', 'frac_traveltime_lt15', 'hhinc00', 'median_house_value', 'ccd_exp_tot', 'score_r', 'cs_fam_wkidsinglemom', 'subcty_exp_p
c', 'taxrate', 'tax_st_diff_top20', 'pm25', 'pm25_mia', 'summer_tmmx', 'summer_rmax', 'winter_tmmx', 'winter_rmax', 'bmcruderate',
'Alabama', 'Arizona', 'Arkansas', 'California', 'Colorado', 'Connecticut', 'Delaware', 'Florida', 'Georgia', 'Idaho', 'Illinois',
'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota', 'Mississipp
i', 'Missouri', 'Montana', 'Nebraska', 'Nevada', 'New Hampshire', 'New Mexico', 'New York', 'North Carolina', 'North Dakota', 'Ohi
o', 'Oklahoma', 'Oregon', 'Pennsylvania', 'Rhode Island', 'South Carolina', 'South Dakota', 'Tennessee', 'Texas', 'Utah', 'Vermon
t', 'Virginia', 'Washington', 'West Virginia', 'Wisconsin', 'Wyoming']
```

In [11]:

```
#train, test = train_test_split(covid_dummies, test_size=0.2, random_state=10)


X= covid_dummies[predictors[1:]]
```

```
y= covid_dummies['deathspc']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)

X_train = pd.DataFrame(preprocessing.scale(X_train))
X_test = pd.DataFrame(preprocessing.scale(X_test))
```

Question 6

Using the training set, fit a model of COVID-19 deaths per capita (y =deathspc) as a func- tion of the Opportunity Insights and PM COVID predictors listed in the spreadsheet, as well as state-level fixed effects (the state dummy variables) using OLS.

In [12]:

```
#X= covid_dummies[predictors[1:]]
#y= covid_dummies['deathspc']

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)


#X_train = sm.add_constant(X_train)
#X_test = sm.add_constant(X_test)


X_train_norm = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
model_norm = sm.OLS(y_train, X_train_norm).fit()
print(model_norm.summary())

y_train_pred = model_norm.predict(X_train_norm)
mse_train = mean_squared_error(y_train, y_train_pred)
print("MSE for the training set:", mse_train)

X_test_norm = np.hstack((np.ones((X_test.shape[0], 1)), X_test))
y_test_pred = model_norm.predict(X_test_norm)
mse_test = mean_squared_error(y_test, y_test_pred)
print("MSE for the test set:", mse_test)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               deathspc   R-squared:                       0.432
Model:                            OLS   Adj. R-squared:                  0.405
Method:                 Least Squares   F-statistic:                     16.12
Date:                Wed, 22 Feb 2023   Prob (F-statistic):         1.63e-202
Time:                        22:32:24   Log-Likelihood:                -11985.
No. Observations:                2332   AIC:                         2.418e+04
```

Df Residuals:	2226	BIC:	2.479e+04
Df Model:	105		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	23.3145	0.875	26.640	0.000	21.598	25.031
x1	1.8467	1.204	1.534	0.125	-0.514	4.207
x2	0.5730	1.454	0.394	0.694	-2.278	3.424
x3	-0.7181	1.272	-0.565	0.572	-3.212	1.776
x4	-0.8748	1.070	-0.818	0.414	-2.973	1.223
x5	0.4054	1.047	0.387	0.699	-1.648	2.459
x6	-0.5829	1.444	-0.404	0.686	-3.414	2.249
x7	0.7901	1.407	0.562	0.575	-1.969	3.550
x8	-2.8279	1.186	-2.384	0.017	-5.154	-0.502
x9	0.3917	1.125	0.348	0.728	-1.815	2.598
x10	-1.3011	2.146	-0.606	0.544	-5.510	2.908
x11	5.8906	2.536	2.322	0.020	0.917	10.864
x12	-1.6728	2.435	-0.687	0.492	-6.448	3.103
x13	-3.6157	2.576	-1.403	0.161	-8.668	1.436
x14	-3.2319	3.579	-0.903	0.367	-10.251	3.787
x15	-2.5734	2.486	-1.035	0.301	-7.448	2.301
x16	-2.4877	1.683	-1.478	0.140	-5.788	0.813
x17	0.1480	2.282	0.065	0.948	-4.326	4.622
x18	0.7174	1.644	0.436	0.663	-2.506	3.941
x19	0.1335	1.588	0.084	0.933	-2.980	3.247
x20	5.1777	4.574	1.132	0.258	-3.793	14.148
x21	-1.7907	1.474	-1.215	0.225	-4.681	1.100
x22	-6.4472	1.873	-3.442	0.001	-10.120	-2.774
x23	0.1356	1.878	0.072	0.942	-3.547	3.819
x24	-1.0490	2.004	-0.524	0.601	-4.979	2.881
x25	-1.1155	1.880	-0.593	0.553	-4.801	2.570
x26	39.1466	15.028	2.605	0.009	9.675	68.618
x27	-26.2382	7.442	-3.526	0.000	-40.832	-11.645
x28	-13.9143	8.482	-1.641	0.101	-30.547	2.718
x29	0.3628	1.305	0.278	0.781	-2.196	2.921
x30	-5.0277	2.443	-2.058	0.040	-9.819	-0.236
x31	1.5883	2.628	0.604	0.546	-3.565	6.741
x32	1.0183	1.779	0.572	0.567	-2.471	4.508
x33	-7.5686	2.148	-3.523	0.000	-11.781	-3.356
x34	-5.0455	1.897	-2.660	0.008	-8.766	-1.325
x35	2.7129	1.441	1.883	0.060	-0.112	5.538
x36	12.7262	2.463	5.167	0.000	7.896	17.556
x37	-1.3030	2.093	-0.622	0.534	-5.408	2.802

x38	-3.2877	1.543	-2.131	0.033	-6.313	-0.262
x39	-1.9666	2.038	-0.965	0.335	-5.963	2.029
x40	3.7595	1.461	2.573	0.010	0.894	6.625
x41	4.7141	2.011	2.344	0.019	0.770	8.658
x42	0.3743	2.120	0.177	0.860	-3.783	4.532
x43	-4.5086	1.967	-2.293	0.022	-8.365	-0.652
x44	18.3812	1.357	13.545	0.000	15.720	21.042
x45	-1.0386	1.732	-0.600	0.549	-4.434	2.357
x46	4.1132	2.734	1.504	0.133	-1.248	9.475
x47	0.8789	2.387	0.368	0.713	-3.803	5.561
x48	2.2306	1.237	1.803	0.071	-0.195	4.656
x49	1.9761	1.617	1.222	0.222	-1.194	5.147
x50	0.8865	2.530	0.350	0.726	-4.075	5.848
x51	-0.1006	1.229	-0.082	0.935	-2.511	2.310
x52	0.2795	1.452	0.192	0.847	-2.568	3.127
x53	0.5988	0.658	0.909	0.363	-0.692	1.890
x54	-1.8387	2.635	-0.698	0.485	-7.006	3.329
x55	-0.4794	1.102	-0.435	0.664	-2.640	1.681
x56	1.0306	3.063	0.336	0.737	-4.977	7.038
x57	-5.3779	3.422	-1.571	0.116	-12.089	1.333
x58	9.3939	4.773	1.968	0.049	0.034	18.754
x59	-1.3182	1.975	-0.668	0.504	-5.191	2.554
x60	0.8314	2.029	0.410	0.682	-3.148	4.810
x61	-2.8590	1.099	-2.602	0.009	-5.014	-0.704
x62	-2.5789	1.245	-2.071	0.038	-5.021	-0.137
x63	-3.8565	1.012	-3.811	0.000	-5.841	-1.872
x64	-7.0115	1.479	-4.740	0.000	-9.912	-4.111
x65	0.3000	1.239	0.242	0.809	-2.130	2.730
x66	4.6079	0.937	4.918	0.000	2.771	6.445
x67	0.8789	0.899	0.977	0.329	-0.885	2.643
x68	-4.5637	1.469	-3.106	0.002	-7.445	-1.683
x69	1.8338	1.439	1.274	0.203	-0.988	4.656
x70	-0.3684	1.401	-0.263	0.793	-3.116	2.380
x71	2.1191	1.142	1.855	0.064	-0.121	4.359
x72	7.4891	1.159	6.463	0.000	5.217	9.762
x73	2.2759	1.267	1.797	0.073	-0.208	4.760
x74	-0.6066	1.125	-0.539	0.590	-2.813	1.600
x75	1.3076	1.120	1.167	0.243	-0.889	3.504
x76	7.7762	1.295	6.003	0.000	5.236	10.317
x77	0.4000	1.035	0.387	0.699	-1.629	2.430
x78	0.2704	0.942	0.287	0.774	-1.576	2.117
x79	7.2678	0.980	7.414	0.000	5.345	9.190
x80	6.0944	1.159	5.257	0.000	3.821	8.368
x81	2.0004	1.414	1.414	0.157	-0.773	4.774

```
x82          -0.6219      1.250     -0.498      0.619      -3.073       1.829
x83          -0.4302      0.990     -0.434      0.664      -2.372       1.512
x84           0.8096      1.129      0.717      0.474      -1.405       3.024
x85           0.9028      1.092      0.827      0.408      -1.238       3.044
x86          -2.0615      1.187     -1.737      0.082      -4.389       0.265
x87           0.0174      0.933      0.019      0.985      -1.812       1.847
x88          -2.2463      1.288     -1.745      0.081      -4.771       0.279
x89           6.7316      1.214      5.544      0.000       4.350       9.113
x90          -3.7375      1.189     -3.144      0.002      -6.069      -1.406
x91           1.7203      1.267      1.358      0.175      -0.763       4.204
x92           1.8889      1.130      1.671      0.095      -0.327       4.105
x93          -1.3299      1.088     -1.223      0.222      -3.463       0.803
x94          -0.9965      1.258     -0.792      0.428      -3.464       1.471
x95           2.6481      1.097      2.413      0.016       0.496       4.800
x96          -0.5128      0.902     -0.569      0.570      -2.281       1.255
x97          -4.5864      1.100     -4.169      0.000      -6.743      -2.429
x98           0.9014      1.170      0.770      0.441      -1.393       3.196
x99          -1.8221      1.032     -1.765      0.078      -3.846       0.202
x100         -7.0452      1.878     -3.752      0.000     -10.728      -3.362
x101         -2.3114      1.454     -1.590      0.112      -5.162       0.540
x102          0.3696      0.978      0.378      0.705      -1.547       2.287
x103         -4.6794      1.127     -4.153      0.000      -6.889      -2.470
x104          0.6940      1.400      0.496      0.620      -2.051       3.439
x105          0.0501      1.025      0.049      0.961      -1.960       2.060
x106          0.9260      1.291      0.718      0.473      -1.605       3.457
x107         -1.1689      1.087     -1.075      0.282      -3.300       0.963
==============================================================================
Omnibus:                     2270.523   Durbin-Watson:                   2.091
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           166465.540
Skew:                           4.505   Prob(JB):                         0.00
Kurtosis:                      43.398   Cond. No.                     1.64e+15
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 9.52e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
MSE for the training set: 1704.9352722841477
MSE for the test set: 1476.592736354548
```

Part b

Why might you be concerned about overfitting in this context? Is there any evidence of overfitting? Briefly explain.

Above, we have actually fit 105 predictors on a sample of just 2332. In this context, overfitting could occur because the model seems too complex as it has a lot of parameters.

R-squared: R-squared is 0.432, which means that the predictors explain only 43.2% of variation in the dependent variable. Adjusted R-squared: This Adjusted R-squared value takes into account the number of independent variables in the model whose value is 0.405. This suggests that there may be some evidence of overfitting in the model. However, this may not be a right metric to observe overfitting.F-statistic: The F-statistic is 16.12 and the associated p-value is very small. This says that the model is statistically significant. AIC and BIC: These are information criteria that are used to compare different models. Lower values indicate better models. In this case, the AIC is 2.418e+04 and the BIC is 2.479e+04. These huge values suggests that the model may be overfitting, as it is penalized for including a large number of predictors. However, this may not be a right metric to observe overfitting. Thus from the above values from the regression results, i.e looking at Adjusted R squared, F-Statistic, AIC, BIC, we can say that the large number of predictors for such a smalll sample maybe penalizing the model. This means that the model may be over stating the performance on training data set. However, the model maynot generalize well to the new data.

However in the results we have observed, we see that the evidence in the form of MSE for training and test data set suggests that there is no overfitting. However, if we observe that there is a change in performace i.e decrease in performance on test set but on the other side the perfromance on training set increases, this can be taken as overfitting. To analyse this further, we can try to attempt to run cross validation either 5 or 10 fold, and detect for overfitting.

Question 7

Use the training set to estimate ridge regression and the lasso analogs to the OLS model in the previous question. For each, you should report a plot of the cross-validation estimates of the test error as a function of the value of the hyperparameter (λ) that indicates the tuned value of λ. Hint: to do so you should be sure standardize your predictors and tune the hyperparameter by:

In [13]:

```
from sklearn.linear_model import RidgeCV, LassoCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
```

In [14]:

```
#X= covid_dummies[predictors[1:]]
#X = pd.DataFrame(preprocessing.scale(X))
#y= covid_dummies['deathspc']

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=25)
```

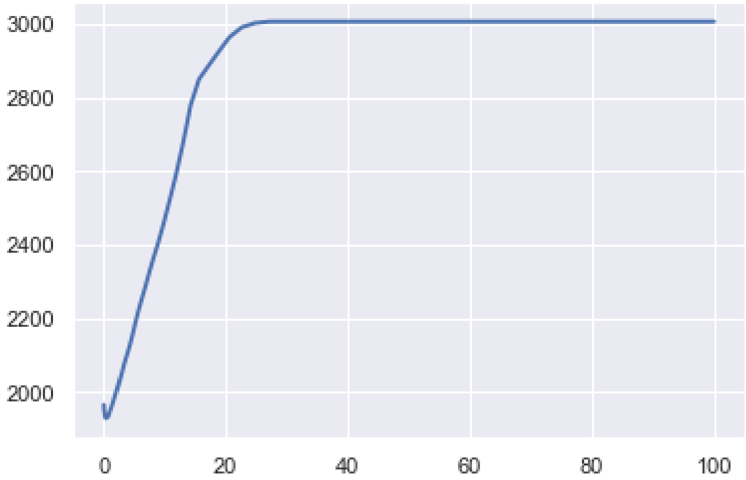
```
#X_train = pd.DataFrame(preprocessing.scale(X_train))
#X_test = pd.DataFrame(preprocessing.scale(X_test))
```

In [15]:

```
#Lasso
kf = KFold(n_splits=10, random_state = 25, shuffle=True)
tkf = kf.split(X_train, y_train)#change
lasso = Lasso()
alphacl = (np.logspace(-2, 2, 100))
alpha_lgrid = [{'alpha': alphacl}]
def vector_values(grid_search, trials):
    mean_vec = np.zeros(trials)
    std_vec = np.zeros(trials)
    i = 0
    final = grid_search.cv_results_


    for mean_score, std_score in zip(final["mean_test_score"], final["std_test_score"]):
        mean_vec[i] = -mean_score
        std_vec[i] = std_score
        i = i+1
    return mean_vec, std_vec

grid_search_lasso = GridSearchCV(lasso, alpha_lgrid, cv = tkf, scoring = 'neg_mean_squared_error')
grid_search_lasso.fit(X_train, y_train) #change
mean_vec, std_vec = vector_values(grid_search_lasso, 100)

results_cv = pd.DataFrame(
    {
        "alphas": alphacl,
        "MSE": mean_vec,
    }
)
min_mse = min(results_cv['MSE'])
results_cv.loc[results_cv['MSE']==min_mse]
plt.plot(
    alphacl,
    mean_vec,
    linewidth=2,
)


optimal_alpha = results_cv.loc[results_cv['MSE'] == min_mse]['alphas'].values[0]
```

```python
print("MSE at optimal alpha: ", min_mse)


lasso_optimal = Lasso(alpha=optimal_alpha)
lasso_optimal.fit(X_train, y_train) #change


y_pred_lasso = lasso_optimal.predict(X_train) #change
mse_lasso = mean_squared_error(y_train, y_pred_lasso) #change
print("MSE for lasso with optimal alpha:", mse_lasso)

print("R square value for lasso with optimal alpha: " , lasso_optimal.score(X_train, y_train))
```

```
MSE at optimal alpha:  1927.757957321408
MSE for lasso with optimal alpha: 1741.391623952136
R square value for lasso with optimal alpha:  0.4198535762904305
```

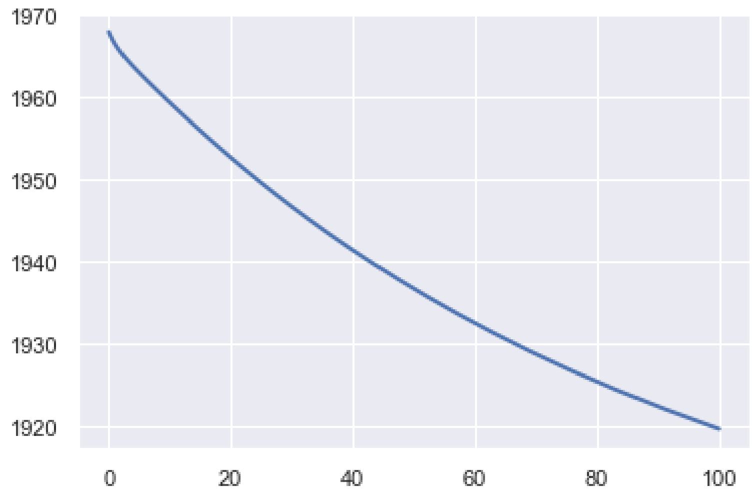


In [16]:

```python
# Ridge

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=25)


#X_train = pd.DataFrame(preprocessing.scale(X_train))

kf = KFold(n_splits=10, random_state = 25, shuffle=True)
tkf = kf.split(X_train, y_train) #change
ridge = Ridge()
```

```python
alphacr = (np.logspace(-2, 2, 100))
alpha_rgrid = [{'alpha': alphacr}]
grid_search_ridge = GridSearchCV(ridge, alpha_rgrid, cv = tkf, scoring ='neg_mean_squared_error')
grid_search_ridge.fit(X_train, y_train) #change
mean_vec, std_vec = vector_values(grid_search_ridge, 100)
results_cv = pd.DataFrame(
    {
        "alphas": alphacr,
        "MSE": mean_vec,
    }
)
min_mse1 = min(results_cv['MSE'])
results_cv.loc[results_cv['MSE']==min_mse1]
plt.plot(
    alphacr,
    mean_vec,
    linewidth=2,
)


optimal_alpha1 = results_cv.loc[results_cv['MSE'] == min_mse1]['alphas'].values[0]
ridge_optimal = Ridge(alpha=optimal_alpha1)
ridge_optimal.fit(X_train, y_train) #change

print("MSE at optimal alpha: ", min_mse1)


y_pred_ridge = ridge_optimal.predict(X_train)
mse_ridge = mean_squared_error(y_train, y_pred_ridge)
print("MSE for Ridge with optimal alpha:", mse_ridge)


print("R square value for Ridge with optimal alpha:" , ridge_optimal.score(X_train, y_train))
```

```
MSE at optimal alpha:  1919.5891999810708
MSE for Ridge with optimal alpha: 1718.4122366607544
R square value for Ridge with optimal alpha: 0.42750918297462714
```



Question8

Using the ridge regression and the lasso models you trained based on the optimal values of λyou found in the previous question, calculate and report the training and test set prediction errors (MSE) for each model. Did ridge regression and/or the lasso improve your predic- tion over OLS? Which model performs the best? Briefly explain which model you would recommend to the CDC and why.

In [19]:

```
#Lasso

y_pred_train_lasso = lasso_optimal.predict(X_train) #change
mse_train_lasso = mean_squared_error(y_train, y_pred_train_lasso) #change
print("MSE for lasso with optimal alpha on train data:", mse_train_lasso)
print("R square value for lasso with optimal alpha on train data: " , lasso_optimal.score(X_train, y_train))




y_pred_test_lasso = lasso_optimal.predict(X_test) #change
mse_test_lasso = mean_squared_error(y_test, y_pred_test_lasso) #change
print("MSE for lasso with optimal alpha on test data:", mse_test_lasso)
print("R square value for lasso with optimal alpha on test data: " , lasso_optimal.score(X_test, y_test))
```

```
MSE for lasso with optimal alpha on train data: 1741.391623952136
R square value for lasso with optimal alpha on train data:  0.4198535762904305
MSE for lasso with optimal alpha on test data: 1441.6756819004447
R square value for lasso with optimal alpha on test data:  0.0988812619607149
```

In [20]:

```
# Ridge

y_pred_train_ridge = ridge_optimal.predict(X_train)
mse_train_ridge = mean_squared_error(y_train, y_pred_train_ridge)
print("MSE for Ridge with optimal alpha on training data:", mse_train_ridge)
print("R square value for Ridge with optimal alpha on training data:" , ridge_optimal.score(X_train, y_train))


y_pred_test_ridge = ridge_optimal.predict(X_test)
mse_test_ridge = mean_squared_error(y_test, y_pred_test_ridge)
print("MSE for Ridge with optimal alpha on test data:", mse_test_ridge)
print("R square value for Ridge with optimal alpha on test data:" , ridge_optimal.score(X_test, y_test))
```

```
MSE for Ridge with optimal alpha on training data: 1718.4122366607544
R square value for Ridge with optimal alpha on training data: 0.42750918297462714
MSE for Ridge with optimal alpha on test data: 1417.3555674076758
R square value for Ridge with optimal alpha on test data: 0.11408253861109552
```

MSE for lasso with optimal alpha on train data: 1741.391623952136

MSE for lasso with optimal alpha on test data: 1441.6756819004447

MSE for Ridge with optimal alpha on training data: 1718.4122366607544

MSE for Ridge with optimal alpha on test data: 1417.3555674076758

OLS - MSE for the training set: 1704.9352722841477

OLS - MSE for the test set: 1476.592736354548

Comparing the mean squared errors (MSE) obtained from the three models, we can see that, for both ridge regression and lasso improved the prediction over OLS. The MSE for both the ridge and lasso models on the test set is lower than the MSE for OLS on the same set, indicating better performance.

Based on the mean squared error (MSE) results provided, it appears that Ridge regression performs the best out of the three models on the test set. Hence I would recommend Ridge regression for this particular data set. THeoretically, the Ridge adds a penalty term to the OLS and minimzes the coefficients to close to zero resulting in a balance between the bias and variance. This will help with the problem omf overfitting and also imrpoves performance on any new data that we test on. Also due to the senstivity of the data set i.e COVID death prediction, I would want to opt for a model that performs better or you could say best of the options that are available. Thus I would choose the Ridge model based on an extensive analysis and tactical comparision of OLS, Lasso and Ridge based on the MSE values we have observed.

In [ ]:
```

```