ML_MP3

Clarice Tee

2025-02-26

Data Analysis

1.

```
# Import packages
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV, KFold
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from scipy import stats
%matplotlib inline
import seaborn as sns
sns.set_style("darkgrid")
import warnings
warnings.filterwarnings("ignore")
from sklearn.linear_model import LassoLarsIC
```

```python
directory = r"C:\Users\clari\OneDrive\Documents\Machine Learning\mp3"
covid_df = os.path.join(directory, "Data-Covid003.csv")
covid_df = pd.read_csv(covid_df, encoding="latin1")
print(covid_df.shape, "\n")
```

```
(3107, 93)
```

```python
# Filter the relevant variables
var_desc = os.path.join(directory, "PPHA_30545_MP03-Variable_Description.xlsx")
var_desc = pd.read_excel(var_desc)
var_desc["Source"].unique()
# "NY Times", "Opportunity Insights", "PM_COVID"
var_desc= var_desc[
    (var_desc["Source"] == "Opportunity Insights") | (
        var_desc["Source"] == "PM_COVID")
]
relevant_vars = list(var_desc["Variable"].unique())
covid_df = covid_df[["county", "state", "deathspc"] + relevant_vars]
# Set county as index
covid_df.set_index(["county"], inplace=True)
# Print the filtered dataset
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
print(covid_df.shape, "\n")
print(covid_df.head(1), "\n")
```

```
(3107, 62)

           state  deathspc  intersects_msa  cur_smoke_q1  cur_smoke_q2  \
county
Autauga  Alabama  0.544371               1      0.333333      0.238095

         cur_smoke_q3  cur_smoke_q4  bmi_obese_q1  bmi_obese_q2  bmi_obese_q3  \
county
Autauga      0.208333      0.133333         0.375      0.238095       0.26087

         bmi_obese_q4  exercise_any_q1  exercise_any_q2  exercise_any_q3  \
county
Autauga      0.133333              0.5         0.666667         0.666667

         exercise_any_q4  brfss_mia  puninsured2010  reimb_penroll_adj10  \
county
```

```
Autauga              0.8          0       13.601278              9489.02

         mort_30day_hosp_z  adjmortmeas_amiall30day  adjmortmeas_chfall30day  \
county
Autauga           0.799735                 0.146564                 0.111778

         med_prev_qual_z  primcarevis_10  diab_hemotest_10  diab_eyeexam_10  \
county
Autauga         0.574764        85.52827         84.771574        72.419628

         diab_lipids_10  mammogram_10  cs00_seg_inc  cs00_seg_inc_pov25  \
county
Autauga        83.92555     68.465909      0.036455            0.043313

         cs00_seg_inc_aff75  cs_race_theil_2000    gini99  poor_share  \
county
Autauga            0.037231            0.128216  0.379976    0.109228

         inc_share_1perc  frac_middleclass  scap_ski90pcm    rel_tot  \
county
Autauga          0.06143            0.5195      -0.897834  56.112755

         cs_frac_black  cs_frac_hisp  unemp_rate  cs_labforce  cs_elf_ind_man  \
county
Autauga      17.008999      1.396808    0.037379     0.651493        0.164787

         cs_born_foreign  mig_inflow  mig_outflow  pop_density  \
county
Autauga         1.165533     0.05704     0.051018    73.277412

         frac_traveltime_lt15    hhinc00  median_house_value  ccd_exp_tot  \
county
Autauga              0.233124  34379.539            126368.4     4.460057

          score_r  cs_fam_wkidsinglemom  subcty_exp_pc   taxrate  \
county
Autauga -7.308133              0.191595      1059.6693  0.011183

         tax_st_diff_top20       pm25  pm25_mia  summer_tmmx  summer_rmax  \
county
Autauga                0.0  11.712587         0    306.02344     96.05542

         winter_tmmx  winter_rmax  bmcruderate
```

```
county
Autauga    288.08508    85.651848    859.29999
```

2 .

```
# Check data types to find out next process
val_type = covid_df.dtypes.reset_index()
val_type = val_type[
(val_type[0] != "int64") & (val_type[0] != "float64")
]
print(val_type)
```

```
   index       0
0  state  object
```

```
# compute summary stats of relevant vars
summary_relevant_vars = covid_df.describe()
print(summary_relevant_vars)
```

```
          deathspc  intersects_msa  cur_smoke_q1  cur_smoke_q2  cur_smoke_q3  \
count  3107.000000     3107.000000   3107.000000   3107.000000   3107.000000
mean      0.751995        0.596717      0.212659      0.171048      0.134467
std       1.792700        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%       0.143666        1.000000      0.250000      0.198718      0.142857
75%       0.689514        1.000000      0.310931      0.250000      0.200000
max      50.773304        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

       exercise_any_q1  exercise_any_q2  exercise_any_q3  exercise_any_q4  \
count      3107.000000      3107.000000      3107.000000      3107.000000
mean          0.455995         0.555671         0.603792         0.638727
```

std	0.273874	0.322336	0.357861	0.376922
min	0.000000	0.000000	0.000000	0.000000
25%	0.312500	0.444444	0.354167	0.400000
50%	0.566563	0.707143	0.778364	0.833333
75%	0.641509	0.769231	0.841804	0.890497
max	1.000000	1.000000	1.000000	1.000000

	brfss_mia	puninsured2010	reimb_penroll_adj10	mort_30day_hosp_z \
count	3107.000000	3107.000000	3103.000000	3106.000000
mean	0.249437	18.469460	9302.737743	0.457806
std	0.432757	5.536651	1590.926253	1.206493
min	0.000000	3.625483	3663.530000	-7.778000
25%	0.000000	14.410248	8159.340000	-0.255867
50%	0.000000	18.147072	9193.770000	0.400088
75%	0.000000	21.961417	10285.430000	1.147822
max	1.000000	41.366287	18443.220000	8.472745

	adjmortmeas_amiall30day	adjmortmeas_chfall30day	med_prev_qual_z \
count	3106.000000	3107.000000	3012.000000
mean	0.165483	0.108969	-0.148547
std	0.039408	0.023565	0.863881
min	0.000000	0.000000	-4.853847
25%	0.145312	0.096301	-0.615591
50%	0.162727	0.107242	-0.090228
75%	0.183402	0.120155	0.444429
max	0.444663	0.344451	3.478521

	primcarevis_10	diab_hemotest_10	diab_eyeexam_10	diab_lipids_10 \
count	3098.000000	3069.000000	3054.000000	3057.000000
mean	80.865348	83.706025	66.080221	78.307420
std	7.401457	6.594153	7.598549	7.854145
min	18.331749	16.911765	31.372549	19.661336
25%	78.803466	81.108462	61.258165	75.000000
50%	82.202779	84.782609	65.977073	79.759036
75%	84.957865	87.679083	70.906526	83.342526
max	95.665079	100.000000	90.000000	94.482759

	mammogram_10	cs00_seg_inc	cs00_seg_inc_pov25	cs00_seg_inc_aff75 \
count	3029.000000	3107.000000	3107.000000	3107.000000
mean	63.110073	0.025892	0.024278	0.026463
std	8.397699	0.030576	0.030757	0.032920
min	30.000000	-0.013363	-0.019502	-0.001993
25%	57.943925	0.005047	0.004164	0.003455

50%	63.618290	0.013647	0.013136	0.012577
75%	68.907563	0.036453	0.034737	0.037337
max	95.238095	0.438241	0.749106	0.196959

	cs_race_theil_2000	gini99	poor_share	inc_share_1perc	\
count	3107.000000	3008.000000	3107.000000	3008.000000	
mean	0.075402	0.379021	0.141739	0.094808	
std	0.084131	0.086677	0.065460	0.050631	
min	0.000000	0.160954	0.000000	0.018570	
25%	0.015591	0.317518	0.095383	0.062577	
50%	0.047192	0.369998	0.129621	0.083600	
75%	0.104508	0.429472	0.175282	0.113570	
max	0.712014	1.091437	0.569170	0.734770	

	frac_middleclass	scap_ski90pcm	rel_tot	cs_frac_black	\
count	3106.000000	3107.000000	3106.000000	3107.000000	
mean	0.554244	0.000182	53.224564	8.744503	
std	0.093099	1.347960	18.502524	14.483719	
min	0.215630	-4.258739	1.816347	0.000000	
25%	0.491883	-0.964225	39.669796	0.264501	
50%	0.559830	-0.091105	51.328668	1.691121	
75%	0.622758	0.818039	64.786780	10.031043	
max	0.875000	9.911112	164.527310	85.965088	

	cs_frac_hisp	unemp_rate	cs_labforce	cs_elf_ind_man	\
count	3107.000000	3107.000000	3107.000000	3107.000000	
mean	6.209190	0.049871	0.609344	0.159118	
std	12.050404	0.017738	0.070393	0.090862	
min	0.082034	0.016092	0.319209	0.000000	
25%	0.917235	0.037422	0.567037	0.088637	
50%	1.783438	0.046908	0.616551	0.149391	
75%	5.107685	0.058742	0.657982	0.219933	
max	97.539047	0.176995	0.860937	0.485540	

	cs_born_foreign	mig_inflow	mig_outflow	pop_density	\
count	3107.000000	3017.000000	3017.000000	3107.000000	
mean	3.441958	0.028677	0.027522	244.325026	
std	4.836270	0.019034	0.013780	1676.096088	
min	0.000000	0.000000	0.000000	0.099542	
25%	0.898505	0.016502	0.018767	17.479568	
50%	1.727323	0.024430	0.025111	43.130142	
75%	3.922074	0.036320	0.033038	104.991115	
max	50.935669	0.168671	0.153256	66940.078000	

	frac_traveltime_lt15	hhinc00	median_house_value	ccd_exp_tot
count	3107.000000	3107.000000	3.107000e+03	3080.000000
mean	0.403803	32853.502978	1.121801e+05	6.092697
std	0.137215	6975.837500	6.318905e+04	2.103573
min	0.099878	10511.805000	0.000000e+00	3.032457
25%	0.299927	28733.524500	7.704740e+04	5.027049
50%	0.385816	32234.641000	1.007748e+05	5.785282
75%	0.499088	36039.471000	1.285012e+05	6.735288
max	0.817636	77942.648000	1.333001e+06	53.258174

	score_r	cs_fam_wkidsinglemom	subcty_exp_pc	taxrate
count	3069.000000	3107.000000	3107.000000	3107.000000
mean	0.077348	0.194598	2119.407531	0.023089
std	9.007980	0.067828	999.833466	0.013848
min	-38.687138	0.024793	0.000000	0.000000
25%	-4.969633	0.152436	1510.192750	0.014993
50%	0.834938	0.182469	1935.919400	0.020339
75%	5.990181	0.221578	2505.411100	0.027164
max	32.985218	0.543878	20541.918000	0.209907

	tax_st_diff_top20	pm25	pm25_mia	summer_tmmx	summer_rmax
count	3106.000000	3107.000000	3107.000000	3107.000000	3107.000000
mean	0.775634	8.371871	0.003540	303.126997	88.970517
std	1.470989	2.565927	0.059405	3.173950	9.689271
min	0.000000	0.000000	0.000000	290.455540	31.643282
25%	0.000000	6.309710	0.000000	300.848035	88.052494
50%	0.000000	8.784647	0.000000	303.290440	91.320313
75%	1.000000	10.483764	0.000000	305.817430	94.812389
max	7.220000	15.786018	1.000000	313.872680	99.778748

	winter_tmmx	winter_rmax	bmcruderate
count	3107.000000	3107.000000	3107.00000
mean	280.404875	87.469432	1029.15597
std	6.597855	4.811207	248.38181
min	264.693820	58.159798	189.30000
25%	275.113020	85.093342	864.29999
50%	280.154690	88.028793	1036.30000
75%	285.543750	90.747704	1194.10000
max	298.340360	97.672874	1978.60000

3.

```
# Count NAs in each column
na_count_val = pd.DataFrame(
    np.sum(covid_df.isna(), axis=0), columns=["Count NAs"]
)
na_count_val = na_count_val[na_count_val["Count NAs"] != 0]
print(na_count_val)
```

```
                         Count NAs
reimb_penroll_adj10              4
mort_30day_hosp_z                1
adjmortmeas_amiall30day          1
med_prev_qual_z                 95
primcarevis_10                   9
diab_hemotest_10                38
diab_eyeexam_10                 53
diab_lipids_10                  50
mammogram_10                    78
gini99                          99
inc_share_1perc                 99
frac_middleclass                 1
rel_tot                          1
mig_inflow                      90
mig_outflow                     90
ccd_exp_tot                     27
score_r                         38
tax_st_diff_top20                1
```

Drop

```
covid_df = covid_df.dropna()
print(covid_df.shape)
```

```
(2915, 62)
```

4. Making dummy vars

```
# Check how many states there are
print(covid_df["state"].describe(), "\n")
# Convert categorical variable into dummy variables
covid_df = pd.get_dummies(covid_df, columns=["state"], drop_first=True)
print(covid_df.head())
```

```
count       2915
unique        47
top        Texas
freq         229
Name: state, dtype: object

         deathspc  intersects_msa  cur_smoke_q1  cur_smoke_q2  cur_smoke_q3  \
county
Autauga  0.544371               1      0.333333      0.238095      0.208333
Baldwin  0.290043               1      0.268097      0.233503      0.167464
Barbour  0.200296               0      0.228571      0.250000      0.181818
Bibb     0.251591               1      0.244444      0.280000      0.181818
Blount   0.100248               1      0.304348      0.260870      0.352941

         cur_smoke_q4  bmi_obese_q1  bmi_obese_q2  bmi_obese_q3  bmi_obese_q4  \
county
Autauga      0.133333      0.375000      0.238095      0.260870      0.133333
Baldwin      0.176991      0.298050      0.262467      0.193237      0.135747
Barbour      0.111111      0.294118      0.571429      0.545455      0.277778
Bibb         0.150000      0.466667      0.375000      0.190476      0.100000
Blount       0.166667      0.347826      0.318182      0.529412      0.235294

         exercise_any_q1  exercise_any_q2  exercise_any_q3  exercise_any_q4  \
county
Autauga         0.500000         0.666667         0.666667         0.800000
Baldwin         0.599432         0.748677         0.835000         0.837963
Barbour         0.542857         0.571429         0.909091         0.722222
Bibb            0.422222         0.560000         0.681818         0.750000
Blount          0.565217         0.478261         0.705882         0.722222

         brfss_mia  puninsured2010  reimb_penroll_adj10  mort_30day_hosp_z  \
county
Autauga          0       13.601278              9489.02           0.799735
Baldwin          0       19.085325              9618.34          -0.113602
Barbour          0       18.513805              9761.77           0.552571
Bibb             0       17.718826             11269.81           1.857274
Blount           0       19.280413             10238.20           1.781737

         adjmortmeas_amiall30day  adjmortmeas_chfall30day  med_prev_qual_z  \
county
Autauga                 0.146564                 0.111778         0.574764
Baldwin                 0.145558                 0.107229         0.118566
Barbour                 0.169922                 0.107575         0.067363
```

Bibb	0.234408	0.112190	-0.235594
Blount	0.177953	0.117951	-0.003070

county	primcarevis_10	diab_hemotest_10	diab_eyeexam_10	diab_lipids_10	\
Autauga	85.528270	84.771574	72.419628	83.925550	
Baldwin	83.215372	80.871050	64.047822	78.992314	
Barbour	88.866842	85.831622	67.556468	81.519507	
Bibb	83.723878	85.478548	64.356436	80.528053	
Blount	84.625169	84.466019	68.155340	79.223301	

county	mammogram_10	cs00_seg_inc	cs00_seg_inc_pov25	cs00_seg_inc_aff75	\
Autauga	68.465909	0.036455	0.043313	0.037231	
Baldwin	69.873998	0.032571	0.021566	0.041103	
Barbour	64.768683	0.021680	0.021777	0.020898	
Bibb	57.228916	0.017462	0.028094	0.002048	
Blount	66.025641	0.013379	0.015049	0.013171	

county	cs_race_theil_2000	gini99	poor_share	inc_share_1perc	\
Autauga	0.128216	0.379976	0.109228	0.06143	
Baldwin	0.100549	0.489738	0.101471	0.12719	
Barbour	0.039993	0.490637	0.267998	0.09734	
Bibb	0.103550	0.417098	0.206075	0.06622	
Blount	0.079703	0.333042	0.117428	0.06281	

county	frac_middleclass	scap_ski90pcm	rel_tot	cs_frac_black	\
Autauga	0.51950	-0.897834	56.112755	17.008999	
Baldwin	0.49911	-0.362414	47.242817	10.224691	
Barbour	0.40833	-1.499349	41.170193	46.039669	
Bibb	0.46136	-0.930472	46.365120	22.010948	
Blount	0.59722	-1.304095	50.791782	1.171997	

county	cs_frac_hisp	unemp_rate	cs_labforce	cs_elf_ind_man	\
Autauga	1.396808	0.037379	0.651493	0.164787	
Baldwin	1.756223	0.039112	0.598249	0.125441	
Barbour	1.646119	0.068132	0.480174	0.313745	
Bibb	1.008355	0.061639	0.528893	0.237480	
Blount	5.326905	0.032847	0.605729	0.195234	

county	cs_born_foreign	mig_inflow	mig_outflow	pop_density	\
Autauga	1.165533	0.057040	0.051018	73.277412	
Baldwin	2.105901	0.044860	0.023537	87.960236	
Barbour	1.498037	0.016565	0.023073	32.814877	
Bibb	0.427350	0.035821	0.029338	33.427227	
Blount	3.098542	0.038825	0.033572	79.035255	

county	frac_traveltime_lt15	hhinc00	median_house_value	ccd_exp_tot	\
Autauga	0.233124	34379.539	126368.4	4.460057	
Baldwin	0.295529	39219.598	163292.5	4.596590	
Barbour	0.394128	24274.195	91443.8	4.734407	
Bibb	0.246582	24927.521	99441.8	4.154157	
Blount	0.213904	30229.857	115704.4	3.998976	

county	score_r	cs_fam_wkidsinglemom	subcty_exp_pc	taxrate	\
Autauga	-7.308133	0.191595	1059.66930	0.011183	
Baldwin	-13.628747	0.186778	2209.91040	0.011756	
Barbour	-15.955111	0.337853	1570.24830	0.012059	
Bibb	-14.839100	0.197729	1338.44120	0.008007	
Blount	-10.218139	0.121988	987.43884	0.007012	

county	tax_st_diff_top20	pm25	pm25_mia	summer_tmmx	summer_rmax	\
Autauga	0.0	11.712587	0	306.02344	96.055420	
Baldwin	0.0	10.077723	0	305.51663	97.971542	
Barbour	0.0	10.981967	0	306.06226	97.371674	
Bibb	0.0	11.998714	0	305.98218	96.293076	
Blount	0.0	11.793022	0	305.17886	94.630951	

county	winter_tmmx	winter_rmax	bmcruderate	state_Arizona	state_Arkansas	\
Autauga	288.08508	85.651848	859.29999	False	False	
Baldwin	290.20886	89.730972	976.20001	False	False	
Barbour	289.24210	88.633575	1040.90000	False	False	
Bibb	287.36282	86.485870	1028.80000	False	False	
Blount	285.56567	85.449142	993.70001	False	False	

county	state_California	state_Colorado	state_Connecticut	state_Delaware	\
Autauga	False	False	False	False	

```
Baldwin              False          False              False          False
Barbour              False          False              False          False
Bibb                 False          False              False          False
Blount               False          False              False          False

         state_Florida  state_Georgia  state_Idaho  state_Illinois  \
county
Autauga          False          False        False           False
Baldwin          False          False        False           False
Barbour          False          False        False           False
Bibb             False          False        False           False
Blount           False          False        False           False

         state_Indiana  state_Iowa  state_Kansas  state_Kentucky  \
county
Autauga          False       False         False           False
Baldwin          False       False         False           False
Barbour          False       False         False           False
Bibb             False       False         False           False
Blount           False       False         False           False

         state_Louisiana  state_Maine  state_Maryland  state_Massachusetts  \
county
Autauga            False        False           False                False
Baldwin            False        False           False                False
Barbour            False        False           False                False
Bibb               False        False           False                False
Blount             False        False           False                False

         state_Michigan  state_Minnesota  state_Mississippi  state_Missouri  \
county
Autauga           False            False              False           False
Baldwin           False            False              False           False
Barbour           False            False              False           False
Bibb              False            False              False           False
Blount            False            False              False           False

         state_Montana  state_Nebraska  state_Nevada  state_New Hampshire  \
county
Autauga          False           False         False                False
Baldwin          False           False         False                False
Barbour          False           False         False                False
Bibb             False           False         False                False
```

```
Blount           False           False         False                 False

         state_New Mexico  state_New York  state_North Carolina  \
county
Autauga             False           False                 False
Baldwin             False           False                 False
Barbour             False           False                 False
Bibb                False           False                 False
Blount              False           False                 False

         state_North Dakota  state_Ohio  state_Oklahoma  state_Oregon  \
county
Autauga               False       False           False         False
Baldwin               False       False           False         False
Barbour               False       False           False         False
Bibb                  False       False           False         False
Blount                False       False           False         False

         state_Pennsylvania  state_Rhode Island  state_South Carolina  \
county
Autauga               False               False                 False
Baldwin               False               False                 False
Barbour               False               False                 False
Bibb                  False               False                 False
Blount                False               False                 False

         state_South Dakota  state_Tennessee  state_Texas  state_Utah  \
county
Autauga               False            False        False       False
Baldwin               False            False        False       False
Barbour               False            False        False       False
Bibb                  False            False        False       False
Blount                False            False        False       False

         state_Vermont  state_Virginia  state_Washington  state_West Virginia  \
county
Autauga          False           False             False                False
Baldwin          False           False             False                False
Barbour          False           False             False                False
Bibb             False           False             False                False
Blount           False           False             False                False

         state_Wisconsin  state_Wyoming
```

```
county
Autauga                False          False
Baldwin                False          False
Barbour                False          False
Bibb                   False          False
Blount                 False          False
```

The dataset has 47 states

5. Split the sample into a training set (80%) and a test set (20%).

```
# Split the features and target
X = covid_df.copy()
y = X.pop("deathspc")
# Split training test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=11)
```

6. Using the training set, fit a model of COVID-19 deaths per capita (y =deathspc) as a function 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.

```
ols = LinearRegression().fit(X_train, y_train)
```

a.

```
# Predict and get  MSE of the training set
y_pred_train_ols = ols.predict(X_train)
mse_train_ols = mean_squared_error(y_train, y_pred_train_ols)
print("OLS Training MSE: ", round(mse_train_ols, 2))
# Predict and and MSE on the test set
y_pred_test_ols = ols.predict(X_test)
mse_test_ols = mean_squared_error(y_test, y_pred_test_ols)
print("OLS Test MSE: ", round(mse_test_ols, 2))
```

```
OLS Training MSE:  1.29
OLS Test MSE:  1.83
```

b. Overfitting may be an issue due to the number of predictors we have, even after filtering. This uses up the dgrees of freeom and reduces flexibility. The test error is about 50% larger than the training error, which may cause a substantial difference considering the range of values of deathspc".

```
print(covid_df["deathspc"].mean())
print(covid_df["deathspc"].median())
print(max(covid_df["deathspc"]))
```

```
0.7226499997742709
0.16261908
16.980742
```

7. Scaling predictors

```
print(covid_df["deathspc"].mean())
print(covid_df["deathspc"].median())
print(max(covid_df["deathspc"]))
```

```
0.7226499997742709
0.16261908
16.980742
```

Ridge REgression a. Set up the model and a grid of values of lambda

```
# set up a ridge regression model
ridge = Ridge()
# Choosing alpha
alpha_param = np.power(10, (np.linspace(-2, 2, 100)))
print(alpha_param)
# Create a parameters grid
param_grid = [{
    "alpha": alpha_param
}]

# Helper functions
def vector_values(grid_search):
    final = grid_search.cv_results_
    mean_vec = -np.array(final["mean_test_score"])
    std_vec = np.array(final["std_test_score"])
    return mean_vec, std_vec

def highlight_min(data, color="yellow"):
    attr = f"background-color: {color}"
    if data.ndim == 1:  # Series
        # data == data.min() compares each value to the minimum
```

```
        # is_min becomes a Boolean series marking minimum values
        is_min = data == data.min()
        return [attr if v else "" for v in is_min]
    return ""
```

```
[1.00000000e-02 1.09749877e-02 1.20450354e-02 1.32194115e-02
 1.45082878e-02 1.59228279e-02 1.74752840e-02 1.91791026e-02
 2.10490414e-02 2.31012970e-02 2.53536449e-02 2.78255940e-02
 3.05385551e-02 3.35160265e-02 3.67837977e-02 4.03701726e-02
 4.43062146e-02 4.86260158e-02 5.33669923e-02 5.85702082e-02
 6.42807312e-02 7.05480231e-02 7.74263683e-02 8.49753436e-02
 9.32603347e-02 1.02353102e-01 1.12332403e-01 1.23284674e-01
 1.35304777e-01 1.48496826e-01 1.62975083e-01 1.78864953e-01
 1.96304065e-01 2.15443469e-01 2.36448941e-01 2.59502421e-01
 2.84803587e-01 3.12571585e-01 3.43046929e-01 3.76493581e-01
 4.13201240e-01 4.53487851e-01 4.97702356e-01 5.46227722e-01
 5.99484250e-01 6.57933225e-01 7.22080902e-01 7.92482898e-01
 8.69749003e-01 9.54548457e-01 1.04761575e+00 1.14975700e+00
 1.26185688e+00 1.38488637e+00 1.51991108e+00 1.66810054e+00
 1.83073828e+00 2.00923300e+00 2.20513074e+00 2.42012826e+00
 2.65608778e+00 2.91505306e+00 3.19926714e+00 3.51119173e+00
 3.85352859e+00 4.22924287e+00 4.64158883e+00 5.09413801e+00
 5.59081018e+00 6.13590727e+00 6.73415066e+00 7.39072203e+00
 8.11130831e+00 8.90215085e+00 9.77009957e+00 1.07226722e+01
 1.17681195e+01 1.29154967e+01 1.41747416e+01 1.55567614e+01
 1.70735265e+01 1.87381742e+01 2.05651231e+01 2.25701972e+01
 2.47707636e+01 2.71858824e+01 2.98364724e+01 3.27454916e+01
 3.59381366e+01 3.94420606e+01 4.32876128e+01 4.75081016e+01
 5.21400829e+01 5.72236766e+01 6.28029144e+01 6.89261210e+01
 7.56463328e+01 8.30217568e+01 9.11162756e+01 1.00000000e+02]
```

b. Define function for estimation by 10FCV and c. plot

```
# Part b
kfcv = KFold(n_splits=10, random_state=25, shuffle=True)
def tune_10fcv_ridge(grid):
    grid_search_ridge = GridSearchCV(
        ridge,
        grid,
        cv=kfcv,
        scoring="neg_mean_squared_error"
    )
    grid_search_ridge.fit(X_train, y_train)
```

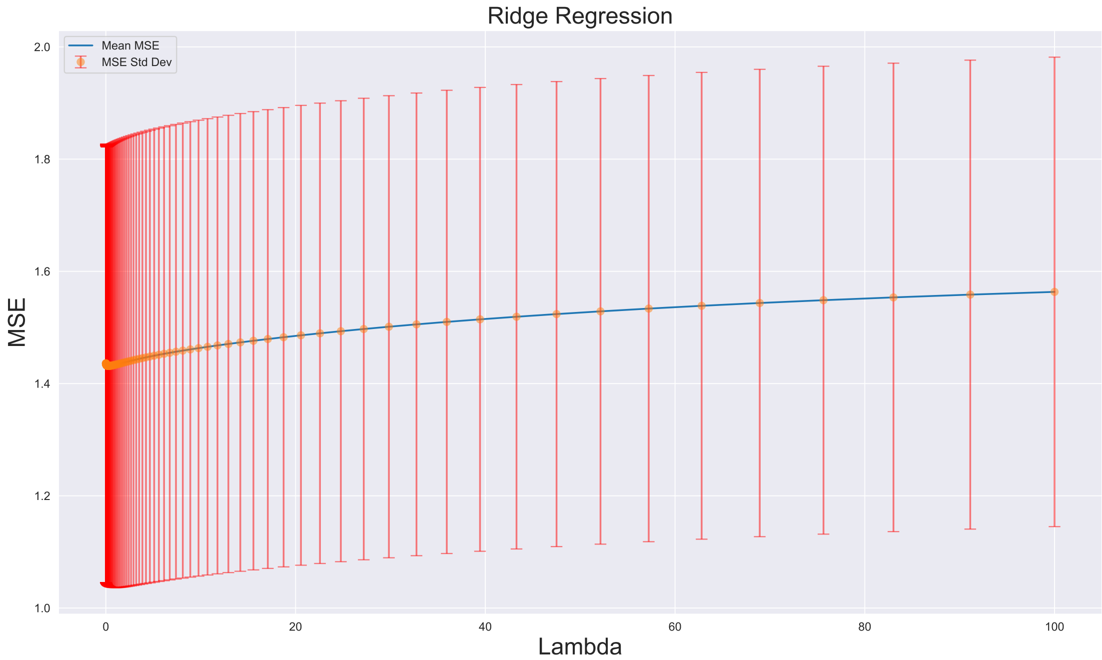
```
    tested_alphas = [params["alpha"] for params in grid_search_ridge.cv_results_["params"
    mean_vec_ridge, std_vec_ridge = vector_values(grid_search_ridge)

    return tested_alphas, mean_vec_ridge, std_vec_ridge, grid_search_ridge

# Part c:
tested_alphas, mean_vec_ridge, std_vec_ridge, grid_search_ridge = tune_10fcv_ridge(param_

# Plotting code
fig, ax = plt.subplots(figsize=(16, 9))
ax.set_title("Ridge Regression", fontsize=20)
ax.plot(tested_alphas, mean_vec_ridge, label="Mean MSE")
ax.errorbar(tested_alphas, mean_vec_ridge, yerr=std_vec_ridge, fmt="o",
            ecolor="r", capsize=5, alpha=0.5, label="MSE Std Dev")
ax.set_ylabel("MSE", fontsize=20)
ax.set_xlabel("Lambda", fontsize=20)
plt.legend()
plt.show()

# Get best parameters
best_alpha_ridge = grid_search_ridge.best_params_["alpha"]
best_mse_ridge = -grid_search_ridge.best_score_
print(f"Best hyper-parameter: {best_alpha_ridge}")
print(f"Best MSE: {best_mse_ridge}")
```




```
Best hyper-parameter: 0.28480358684358O2
Best MSE: 1.4314711516791183
```

```
# Try  the starting grid
_ = tune_10fcv_ridge(param_grid)
```

It seems that the estimated MSE increases as lambda increases, so we are not sure that lambda = 100 is the optimum where MSE is at least minimized before it starts to increase again. Thus, we need to try another grid covering lambda larger than 100. By trying the grid ranging from 0 to 1000 , we can observe qhere the optimallambda lies.

d.

```
# Choosing and reporting the optimal value of lambda
# Try another grid to findout the best point
# Set the range of the hyperparameter
alpha_param = np.power(10, (np.linspace(-2, 3, 100)))
print(alpha_param)
# Create a parameters grid
param_grid = [{
    "alpha": alpha_param
}]

_ = tune_10fcv_ridge(param_grid)
```

```
[1.00000000e-02 1.12332403e-02 1.26185688e-02 1.41747416e-02
 1.59228279e-02 1.78864953e-02 2.00923300e-02 2.25701972e-02
 2.53536449e-02 2.84803587e-02 3.19926714e-02 3.59381366e-02
 4.03701726e-02 4.53487851e-02 5.09413801e-02 5.72236766e-02
 6.42807312e-02 7.22080902e-02 8.11130831e-02 9.11162756e-02
 1.02353102e-01 1.14975700e-01 1.29154967e-01 1.45082878e-01
 1.62975083e-01 1.83073828e-01 2.05651231e-01 2.31012970e-01
 2.59502421e-01 2.91505306e-01 3.27454916e-01 3.67837977e-01
 4.13201240e-01 4.64158883e-01 5.21400829e-01 5.85702082e-01
 6.57933225e-01 7.39072203e-01 8.30217568e-01 9.32603347e-01
 1.04761575e+00 1.17681195e+00 1.32194115e+00 1.48496826e+00
 1.66810054e+00 1.87381742e+00 2.10490414e+00 2.36448941e+00
 2.65608778e+00 2.98364724e+00 3.35160265e+00 3.76493581e+00
 4.22924287e+00 4.75081016e+00 5.33669923e+00 5.99484250e+00
 6.73415066e+00 7.56463328e+00 8.49753436e+00 9.54548457e+00
 1.07226722e+01 1.20450354e+01 1.35304777e+01 1.51991108e+01
 1.70735265e+01 1.91791026e+01 2.15443469e+01 2.42012826e+01
 2.71858824e+01 3.05385551e+01 3.43046929e+01 3.85352859e+01
 4.32876128e+01 4.86260158e+01 5.46227722e+01 6.13590727e+01
 6.89261210e+01 7.74263683e+01 8.69749003e+01 9.77009957e+01
 1.09749877e+02 1.23284674e+02 1.38488637e+02 1.55567614e+02
 1.74752840e+02 1.96304065e+02 2.20513074e+02 2.47707636e+02
 2.78255940e+02 3.12571585e+02 3.51119173e+02 3.94420606e+02
 4.43062146e+02 4.97702356e+02 5.59081018e+02 6.28029144e+02
 7.05480231e+02 7.92482898e+02 8.90215085e+02 1.00000000e+03]
```

(https://devpress.csdn.net/python/62fd9c6b7e66823466192a80.html)

```
# Try another grid to findout the best point
# Set the range of the hyperparameter
alpha_param = np.linspace(196, 221, 50)
print(alpha_param)
# Create a parameters grid
param_grid = [{
    "alpha": alpha_param
}]
# Properly unpack the returned values
_, _, _, grid_search_final = tune_10fcv_ridge(param_grid)
best_alpha_ridge = grid_search_final.best_params_["alpha"]
```

```
[196.         196.51020408 197.02040816 197.53061224 198.04081633
```

```
198.55102041 199.06122449 199.57142857 200.08163265 200.59183673
201.10204082 201.6122449  202.12244898 202.63265306 203.14285714
203.65306122 204.16326531 204.67346939 205.18367347 205.69387755
206.20408163 206.71428571 207.2244898  207.73469388 208.24489796
208.75510204 209.26530612 209.7755102  210.28571429 210.79591837
211.30612245 211.81632653 212.32653061 212.83673469 213.34693878
213.85714286 214.36734694 214.87755102 215.3877551  215.89795918
216.40816327 216.91836735 217.42857143 217.93877551 218.44897959
218.95918367 219.46938776 219.97959184 220.48979592 221.        ]
```

e. Reestimate the model

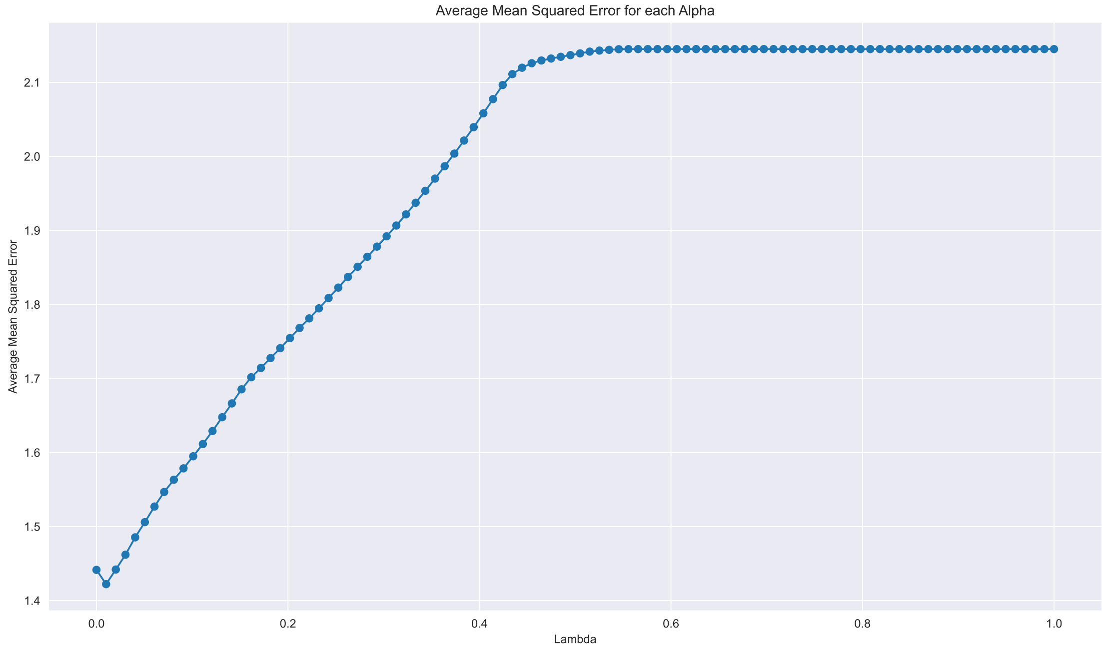
```
ridge = Ridge(alpha=best_alpha_ridge).fit(X_train, y_train)
```

LAsso

```
# Set the grid of the hyperparameter
alpha_range = np.linspace(0, 1, 100)
print(alpha_range)
param_grid = {"lasso__alpha": alpha_range}
# Set up a pipeline and GridSearchCV object
pipeline = make_pipeline(
    StandardScaler(),
    Lasso(random_state=1)
)
grid_search = GridSearchCV(
    pipeline, param_grid,
    cv=kfcv,
    scoring="neg_mean_squared_error"
)
# Fitting the GridSearchCV object
grid_search.fit(X_train, y_train)
# Extract results and convert "mean_test_score" to positive values
# Note: the term mean_test_score refers to the average mean squared error (MSE)
# across the cross-validation folds for each hyper-parameter value when using GridSearch(
results = pd.DataFrame(grid_search.cv_results_)
results["mean_test_score"] = -results["mean_test_score"]
# Plotting mean test scores for each hyper-parameter value using fig, ax
fig, ax = plt.subplots(figsize=(16, 9))
ax.plot(alpha_range, results["mean_test_score"], marker="o")
ax.set_xlabel("Lambda")
ax.set_ylabel("Average Mean Squared Error")
ax.set_title("Average Mean Squared Error for each Alpha")
```

```python
plt.show()
# Getting the best hyper-parameter value and corresponding MSE
best_alpha_lasso = grid_search.best_params_["lasso__alpha"]
best_mse_lasso = grid_search.best_score_
print(f"Best hyper-parameter: {best_alpha_lasso}")
print(f"Best MSE: {-best_mse_lasso}")
```

```
[0.         0.01010101 0.02020202 0.03030303 0.04040404 0.05050505
 0.06060606 0.07070707 0.08080808 0.09090909 0.1010101  0.11111111
 0.12121212 0.13131313 0.14141414 0.15151515 0.16161616 0.17171717
 0.18181818 0.19191919 0.2020202  0.21212121 0.22222222 0.23232323
 0.24242424 0.25252525 0.26262626 0.27272727 0.28282828 0.29292929
 0.3030303  0.31313131 0.32323232 0.33333333 0.34343434 0.35353535
 0.36363636 0.37373737 0.38383838 0.39393939 0.4040404  0.41414141
 0.42424242 0.43434343 0.44444444 0.45454545 0.46464646 0.47474747
 0.48484848 0.49494949 0.50505051 0.51515152 0.52525253 0.53535354
 0.54545455 0.55555556 0.56565657 0.57575758 0.58585859 0.5959596
 0.60606061 0.61616162 0.62626263 0.63636364 0.64646465 0.65656566
 0.66666667 0.67676768 0.68686869 0.6969697  0.70707071 0.71717172
 0.72727273 0.73737374 0.74747475 0.75757576 0.76767677 0.77777778
 0.78787879 0.7979798  0.80808081 0.81818182 0.82828283 0.83838384
 0.84848485 0.85858586 0.86868687 0.87878788 0.88888889 0.8989899
 0.90909091 0.91919192 0.92929293 0.93939394 0.94949495 0.95959596
 0.96969697 0.97979798 0.98989899 1.        ]
```




```
Best hyper-parameter: 0.010101010101010102
Best MSE: 1.4222292966664898
```

re-estimate the model

```
lasso = Lasso(alpha=best_alpha_lasso).fit(X_train, y_train)
```

8.

```
# Predict and calc Ridge model MSE on the training/test set
y_pred_train_ridge = ridge.predict(X_train)
mse_train_ridge = mean_squared_error(y_train, y_pred_train_ridge)
print("Ridge Regression Training MSE: ", round(mse_train_ridge, 2))
y_pred_test_ridge = ridge.predict(X_test)
mse_test_ridge = mean_squared_error(y_test, y_pred_test_ridge)
print("Ridge Regression Test MSE: ", round(mse_test_ridge, 2))
# Predict and calc Lasso model MSE on the training/test set
y_pred_train_lasso = lasso.predict(X_train)
mse_train_lasso = mean_squared_error(y_train, y_pred_train_lasso)
print("Lasso Training MSE: ", round(mse_train_lasso, 2))
y_pred_test_lasso = lasso.predict(X_test)
mse_test_lasso = mean_squared_error(y_test, y_pred_test_lasso)
print("Lasso Test MSE: ", round(mse_test_lasso, 2))
```

```
Ridge Regression Training MSE:  1.51
Ridge Regression Test MSE:  1.9
Lasso Training MSE:  1.48
Lasso Test MSE:  1.87
```

It looks like the Ridge regression and Lasso result in a better prediction than the OLS.

While OLS achieves the lowest training MSE the regularization penalties in Ridgeand Lasso effectively reduce model variance, yielding better test set performance. Plus the OLS inherently has a lower bias.