

Problem 1:

Q(i) "Not" means choosing p from all,
 "Yes" means choosing p from top 10 correlated p to target.

	Name_Not	Params_Not	Name_Yes	Params_Yes
0	state_28	-0.978944	PctIlleg	0.297512
1	state_12	0.582829	racePctWhite	-0.215177
2	state_13	-0.545023	FemalePctDiv	0.162579
3	state_51	-0.540313	PctKids2Par	-0.660482
4	state_42	-0.3721	PctFam2Par	0.379547
5	state_6	0.376383	pctWInvInc	0.035502
6	pctUrban	0.071043	racepctblack	-0.033854
7	MedRent	0.423081	PctYoungKids2Par	0.022763
8	PctWorkMom	-0.134557	pctWPubAsst	0.011648
9	state_39	-0.227767	PctTeen2Par	0.014326

DLS

	Name_Yes	value_Yes(Correlation)
0	PctKids2Par	0.738424
1	PctIlleg	0.737957
2	PctFam2Par	0.706667
3	racePctWhite	0.684770
4	PctYoungKids2Par	0.666059
5	PctTeen2Par	0.661582
6	racepctblack	0.631264
7	pctWInvInc	0.576324
8	pctWPubAsst	0.574665
9	FemalePctDiv	0.556032

Best

Subset

	Name_Yes	Name_Not
0	PctKids2Par	state_12
1	PctIlleg	state_13
2	PctFam2Par	state_25
3	racePctWhite	state_28
4	FemalePctDiv	racePctWhite
5	NaN	MalePctDivorce
6	NaN	PctKids2Par
7	NaN	PctWorkMom
8	NaN	PctIlleg
9	NaN	HousVacant

Step

Use

Lasso

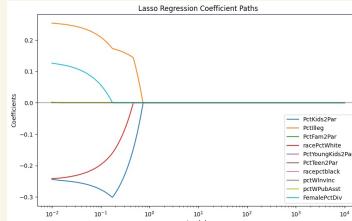
	Name_Yes	Value_Yes	Name_Not	Value_Not
0	PctKids2Par	-0.266173	racePctWhite	-0.185912
1	PctIlleg	0.224178	pctUrban	0.006777
2	racePctWhite	-0.212139	MalePctDivorce	0.064956
3	FemalePctDiv	0.079820	PctKids2Par	-0.266554
4	NaN	NaN	PctWorkMom	-0.004336
5	NaN	NaN	PctIlleg	0.192499
6	NaN	NaN	PctPersDenseHous	0.04448
7	NaN	NaN	HousVacant	0.08634
8	NaN	NaN	PctVacantBoarded	0.005694
9	NaN	NaN	NumStreet	0.035718

Elastic

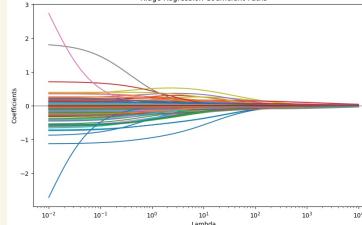
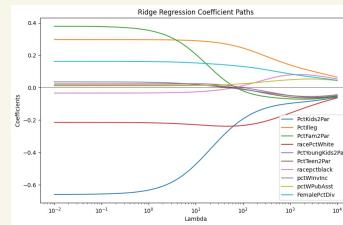
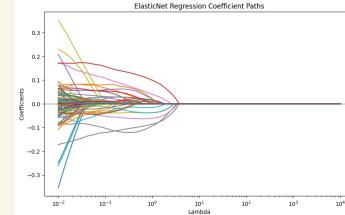
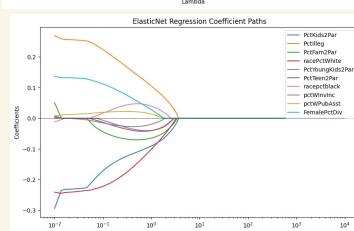
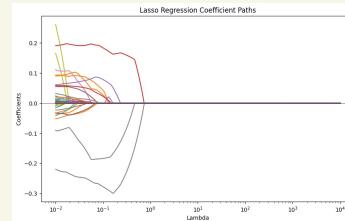
Net

	Name_Yes	Value_Yes	Name_Not	Value_Not
0	PctKids2Par	-0.095074	racepctblack	0.018048
1	PctIlleg	0.102120	racePctWhite	-0.089296
2	PctFam2Par	-0.063209	pctWInvInc	-0.001308
3	racePctWhite	-0.089257	FemalePctDiv	0.003679
4	PctYoungKids2Par	-0.030703	TotalPctDiv	0.001234
5	PctTeen2Par	-0.028707	PctFam2Par	-0.06309
6	racepctblack	0.018050	PctKids2Par	-0.094948
7	pctWInvInc	-0.001408	PctYoungKids2Par	-0.030612
8	FemalePctDiv	0.004179	PctTeen2Par	-0.0286
9	NaN	NaN	PctIlleg	0.102142

(ii) with top 10 correlated P



with all P



Reflecting on the feature selection results, we observe that different methods prioritize different features due to their unique selection criteria.

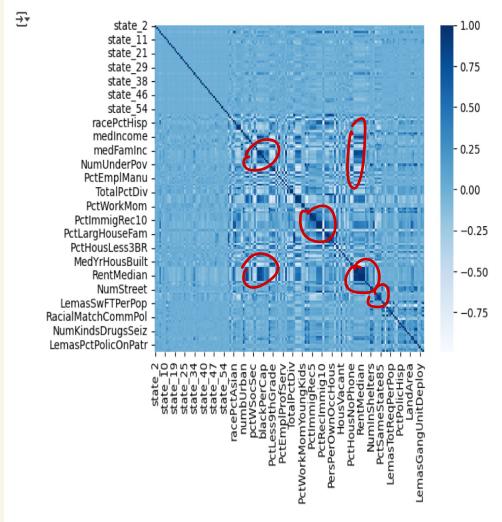
- OLS identifies features based on statistical significance**, selecting those with low p-values. In my selected params, most are state dummies. However, other models do not quite find state dummies significant, except for step-wise model. And this model does not choose features ('PctKids2Par', 'PctIllleg', 'racePctWhite') which are chosen by all other models.
- LASSO selects a sparse set of features**, dropping those with high collinearity. Based on the regularization paths with top correlated features, it shows that 'PctKids2Par' increases as 'FemalePctDiv' decreases to zero, indicating their high correlation. Finally, all betas shrink to zero. Different lambda to be set will lead to different number and value of betas.
- Ridge retains all features but shrinks them differently depending on multicollinearity**. Unlike Lasso and Elastic_Net, Ridge won't make betas ultimately to zeros. Instead, it forces betas to converge into several groups.
- Best Subsets chooses a feature set optimizing predictive accuracy**. It compares all combination of betas, which takes much longer time.
- Stepwise approaches iteratively adding features to improve model fit**. It returns the results much faster than the Best Subset model, but it won't check all possible combinations.
- Elastic Net balances LASSO and Ridge behaviors, selecting features based on both sparsity and correlation handling**. Because it contains norm 1 penalty, it did shrink betas to zero finally, but due to the l1_ratio it shrunk betas with slow pace.

Different tuning parameters led to different selected features, especially in **LASSO** and **Elastic Net**, where higher regularization forces more coefficients to zero. However, some features were **consistently selected across all methods**, indicating their strong predictive power.

Most Important Features: After comparing all methods and getting 27 significant features. The most consistently selected and impactful features are:

- 'PctKids2Par', 'PctIllleg', 'racePctWhite': Selected in four models.
- 'PctWorkMom': Selected in three models.
- 'state_28', 'state_12', 'state_13', 'pctUrban', 'MalePctDivorce', 'HousVacant', 'racePctBlack', 'pctInvlinc', 'FemalePctDiv', 'PctFam2Par', 'PctYoungKids2Par', 'PctTeen2Par': Selected in two models.
- 'state_51', 'state_42', 'state_6', 'MedRent', 'state_39', 'state_25', 'PctPersDenseHous', 'PctVacantBoarded', 'NumStreet', 'TotalPctDiv', 'PctWPubAsst': Selected in one model.

Thus, 'PctKids2Par', 'PctIllleg', 'racePctWhite' can be considered the most critical features, as they remain important regardless of the selection method used.

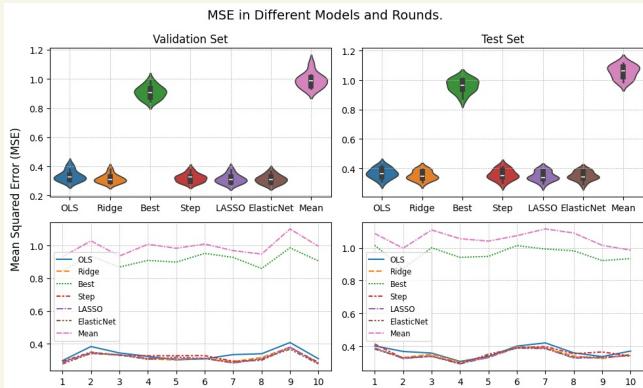


(b)

(i)

	validation	test
OLS	5	0.335173 0.367681
Ridge	3	0.315781 0.352137
Best	6	0.909232 0.962026
Step	4	0.321264 0.358342
LASSO	2	0.314434 0.347623
ElasticNet	1	0.312482 0.349480
Mean	7	0.990844 1.056437

(ii)



(iii)

The MSE of these models based on validation starting at least to highest is:

ElasticNet < Lasso < Ridge < Step < OLS < Best < Mean

The MSE of these models based on test starting at least to highest is:

Lasso < ElasticNet < Ridge < Step < OLS < Best < Mean

So based on these results, we can conclude that:

Elastic Net and LASSO achieve the best prediction error because they effectively combine the advantages of LASSO (feature selection) and Ridge (handling multicollinearity). Additionally, there are 56 highly correlated features in the dataset, where the absolute correlation exceeds 0.9. This makes regularization methods particularly useful in reducing overfitting.

From the graphs, it is difficult to clearly identify overfitting. However, OLS tends to overfit when the number of parameters exceeds the number of records ($p > n$). In this case, where X has shape (1994, 165), this is not an issue.

Since LASSO, Ridge, and Elastic Net include regularization, they cannot perform worse than OLS in terms of generalization. However, Stepwise Selection and Best Subset Selection are more prone to overfitting because they may select too many parameters. Additionally, these methods are computationally expensive, so I limited their result size to ensure efficiency.

Conclusion:
Lasso is the overall best method for prediction on this dataset.

Problem 2 :

(a) (i)

$$y_i = f(x_i) + \epsilon_i = \beta_0 + \sum_{i=1}^p \beta_i x_i + \epsilon_i \quad (\text{ii})$$

$$\bar{y}_i = \beta_0 + \sum_{i=1}^p \beta_i x_i + \epsilon_i$$

$$\frac{\sum y_i}{n} = \frac{\sum f(x_i) + \epsilon_i}{n}$$

$$y_i = \beta_0 x_i + \sum_{i=1}^p \beta_i x_i + \epsilon_i$$

$$\bar{y} = \beta_0 + \sum_{i=1}^p \beta_i \bar{x}_i$$

$$\therefore Y = X\beta + \epsilon$$

$$\therefore y_i - \bar{y} = \sum_{i=1}^p \beta_i (x_i - \bar{x}_i)$$

$$X = \begin{pmatrix} 1_{nx1}, x_1, \dots \end{pmatrix}_{n \times (p+1)} \quad \beta = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_p \end{pmatrix}_{(p+1) \times 1}$$

these models predict the slope in same values.

```
true_beta: [2 3 5 4]
(i)    model1: [0.          4.09753977 5.17005687 9.37841763]
(ii)   model2: [-86.18920724  4.09753977  5.17005687  9.37841763]
MSE of model1: 243404.143
MSE of model2: 243404.143
```

Empirically I test about

$$Y_i = 2 + 3X_{1i} + 4X_{2i} + 5X_{3i} + \epsilon_i$$

$$Y - \bar{Y} = (X - \bar{X})\beta + \epsilon, \text{ as model 1}$$

$$Y = X\beta + \epsilon, \text{ as model 2.}$$

Their results are same.

①

When $p > n$.

There are more variables than the limitations.

Therefore, there are at least $p-n$

random variables to generate $\hat{\beta}$

s.t. $X\hat{\beta} = Y$

As long as, we find such $\hat{\beta}$'s, the training

$$\text{error} = \|Y - X\hat{\beta}\|^2 = 0$$

With $x.shape$ as $n=10$ and $p=16$
There are many feasible beta_hats, one beta is
[[-0.]]

[3.537]
[0.049]
[1.849]
[5.138]
[7.323]
[6.947]
[10.556]
[-0.894]
[-0.697]
[6.799]
[0.247]
[2.604]
[8.995]
[8.919]
[4.436])

Under circumstance of $p > n$, MSE=0.000

② (i)

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

$$\text{Var}(\hat{\beta}) = (X^T X)^{-1} \sigma_e^2$$

if we centralize $X \Rightarrow \hat{X}$, $\hat{\beta}$ unchanged.

$$\text{Then } \text{Cov}(\hat{X}) = \frac{1}{n-1} \hat{X}^T \hat{X}.$$

$$\therefore \hat{X}^T \hat{X} = (n-1) \text{Cov}(\hat{X})$$

$$\begin{aligned} \text{Var}(\hat{\beta}) &= (n-1) \text{Cov}(\hat{X})^{-1} \sigma_e^2 \\ &= \frac{\sigma_e^2}{n-1} \text{Cov}(\hat{X})^{-1} \end{aligned}$$

Correlation Matrix $C = D^T S D$

$$\therefore S = D C D^T \quad D \text{ is diag}(\tau_1, \tau_2, \dots)$$

$$\therefore = \frac{\sigma_e^2}{n-1} (D C D^T)^{-1} = \frac{\sigma_e^2}{n-1} D^{-1} C^{-1} D^{-1}$$

$$\|C^{-1}\| \rightarrow \infty \quad \therefore C^{-1}'s \text{ elements} \uparrow \infty$$

$$\therefore \text{Var}(\hat{\beta}) \uparrow \infty$$

(i): $x.shape=(1000, 6)$
Feature 5 is created by adding a normal(0,3) perturbation to feature 4.

	beta0	beta1	beta2	beta3	beta4	beta5
beta0	1.000000	-0.207558	-0.126775	-0.156165	0.010911	-0.013489
beta1	-0.207558	1.000000	-0.046322	0.036562	0.014034	-0.014058
beta2	-0.126775	-0.046322	1.000000	-0.008151	0.053319	-0.054135
beta3	-0.156165	0.036562	-0.008151	1.000000	-0.009161	0.008840
beta4	0.010911	0.014034	0.053319	-0.009161	1.000000	-0.999870
beta5	-0.013489	-0.014058	-0.054135	0.008840	-0.999870	1.000000

$$\text{Cor}(\text{beta4}, \text{beta5}) = -0.999870$$

Therefore, high variance for correlated features.

$$(ii) \hat{\beta}^r = (X^T X + \lambda I)^{-1} X^T Y$$

if x_i and x_j are highly correlated and centered.

$$\rho_{i,j} = \frac{x_i^T x_j}{\|x_i\| \|x_j\|} \approx 1$$

$$\therefore X^T X = \begin{pmatrix} \|x_1\|^2 & \rho \|x_1\| \|x_2\| \\ \rho \|x_1\| \|x_2\| & \|x_2\|^2 \end{pmatrix}$$

Submatrix.

$$\therefore X^T X + \lambda I = \begin{pmatrix} \|x_1\|^2 + \lambda & \rho \|x_1\| \|x_2\| \\ \rho \|x_1\| \|x_2\| & \|x_2\|^2 + \lambda \end{pmatrix}$$

(ii): $x.shape=(1000, 5)$
 Feature 5 is created by adding a normal(0,3) perturbation to feature 4.
 value
 beta1 178.006782
 beta2 358.254017
 beta3 561.512765
 beta4 822.498310
 beta5 816.541029
 Therefore, ridge would group high correlative features with close betas.

$$\begin{aligned} |X^T X + \lambda I| &= (\|x_1\|^2 + \lambda)(\|x_2\|^2 + \lambda) - (\rho \|x_1\| \|x_2\|)^2 \\ &= (\|x_1\|^2 + \lambda)(\|x_2\|^2 + \lambda) + \lambda^2 - \rho^2 (\|x_1\| \|x_2\|)^2 \\ &= \lambda (\|x_1\|^2 + \|x_2\|^2) + \lambda^2 \end{aligned}$$

$$\therefore (X^T X + \lambda I)^{-1} = \frac{1}{\lambda (\|x_1\|^2 + \|x_2\|^2) + \lambda^2} \begin{pmatrix} \|x_2\|^2 + \lambda & \rho \|x_1\| \|x_2\| \\ \rho \|x_1\| \|x_2\| & \|x_1\|^2 + \lambda \end{pmatrix}$$

for large λ

$$\approx \frac{1}{\lambda} \begin{pmatrix} \lambda & -\rho \\ -\rho & \lambda \end{pmatrix} = \begin{pmatrix} \frac{1}{\lambda} & 0 \\ 0 & \frac{1}{\lambda} \end{pmatrix}$$

$$\therefore \hat{\beta}^r \approx \begin{pmatrix} \frac{1}{\lambda} & 0 \\ 0 & \frac{1}{\lambda} \end{pmatrix} \begin{pmatrix} X_1^T Y \\ X_2^T Y \end{pmatrix}$$

$$\hat{\beta}_1 \approx \frac{X_1^T Y}{\lambda} \quad \hat{\beta}_2 \approx \frac{X_2^T Y}{\lambda}$$

as $X_1^T \approx X_2^T$

$$\therefore \hat{\beta}_1 \approx \hat{\beta}_2$$

(iii)

(iii): $x.shape=(1000, 5)$
 Feature 5 is created by adding a normal(0,3) perturbation to feature 4.
 If lambda big enough (Choosed lambda=0.9), lasso would select only one out of a group of features that are highly correlated.

value
beta1 179.120907
beta2 361.076014
beta3 566.677581
beta4 1646.023194
beta5 0.525146

Appendix

▼ Prework

▼ Import

```
pip install ucimlrepo
```

```
→ Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.11/d
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packag
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7
```

```
import pandas as pd
import numpy as np
import warnings
import seaborn as sns
import matplotlib.pyplot as plt
from ucimlrepo import fetch_ucirepo
```

```
warnings.filterwarnings("ignore")
```

```
# fetch dataset
communities_and_crime = fetch_ucirepo(id=183)

# data (as pandas dataframes)
X = communities_and_crime.data.features
y = communities_and_crime.data.targets

error_list = ['PolicPerPop', 'LemasSwFTPPerPop', 'PctPolicBlack', 'PctPolicHisp', 'Le
communities_and_crime.variables.loc[communities_and_crime.variables.name.isin(error_]

# # metadata
# print(communities_and_crime.metadata)

# # variable information
# print(communities_and_crime.variables)
```

▼ Data Understandign

▼ metadata

```
for i, (key, value) in enumerate(community_and_crime.metadata.items()):  
    print(f"Key-{i}: {key}: {value}")
```

→ Key-0: uci_id: 183
Key-1: name: Communities and Crime
Key-2: repository_url: <https://archive.ics.uci.edu/dataset/183/communities+and+crime>
Key-3: data_url: <https://archive.ics.uci.edu/static/public/183/data.csv>
Key-4: abstract: Communities within the United States. The data combines socio-economic and demographic information from the US Census Bureau.
Key-5: area: Social Science
Key-6: tasks: ['Regression']
Key-7: characteristics: ['Multivariate']
Key-8: num_instances: 1994
Key-9: num_features: 127
Key-10: feature_types: ['Real']
Key-11: demographics: ['Race', 'Age', 'Income', 'Occupation']
Key-12: target_col: ['ViolentCrimesPerPop']
Key-13: index_col: None
Key-14: has_missing_values: yes
Key-15: missing_values_symbol: NaN
Key-16: year_of_dataset_creation: 2002
Key-17: last_updated: Mon Mar 04 2024
Key-18: dataset_doi: 10.24432/C53W3X
Key-19: creators: ['Michael Redmond']
Key-20: intro_paper: {'ID': 405, 'type': 'NATIVE', 'title': 'A data-driven software for community and crime analysis'}
Key-21: additional_info: {'summary': "Many variables are included so that algorithms can be applied directly to the data without extensive preprocessing."}

```
for i, column in enumerate(community_and_crime.variables.columns):  
    if column == 'name':  
        print(f"{i} - {column}: {community_and_crime.variables[column].shape[0]}")  
    else:  
        print(f"{i} - {column}: {community_and_crime.variables[column].value_counts()}")
    print()
```

→ 0 - name: 128 countries

```
1 - role: role  
Feature      127  
Target       1  
Name: count, dtype: int64  
  
2 - type: type  
Continuous    116  
Integer       11  
Categorical   1  
Name: count, dtype: int64
```

```
3 - demographic: demographic
Race          10
Age           4
Income         2
Occupation    2
Name: count, dtype: int64

4 - description: Series([], Name: count, dtype: int64)

5 - units: Series([], Name: count, dtype: int64)

6 - missing_values: missing_values
no      103
yes     25
Name: count, dtype: int64
```

▼ Target

```
communities_and_crime.variables[communities_and_crime.variables.role=="Target"]
```

→	name	role	type	demographic	description	units	missing_v
127	ViolentCrimesPerPop	Target	Continuous			None	None

▼ Type

```
communities_and_crime.variables[communities_and_crime.variables["type"].isin(["Integ
```

index		name	role	type	demographic	description	units
3	3	communityname	Feature	Categorical	None	None	None
0	0	state	Feature	Integer	None	None	None
1	1	county	Feature	Integer	None	None	None
2	2	community	Feature	Integer	None	None	None
4	4	fold	Feature	Integer	None	None	None
5	15	numUrban	Feature	Integer	None	None	None
6	16	pctUrban	Feature	Integer	None	None	None
7	54	NumIlleg	Feature	Integer	None	None	None
8	93	MedOwnCostPctIncNoMtg	Feature	Integer	None	None	None
9	94	NumInShelters	Feature	Integer	None	None	None
10	95	NumStreet	Feature	Integer	None	None	None
11	125	LemasPctOfficDrugUn	Feature	Integer	None	None	None

▼ Demographic data

```
communities_and_crime.variables[~communities_and_crime.variables.demographic.isna()]
```

	index	name	role	type	demographic	description	units	miss
4	11	agePct12t21	Feature	Continuous		Age	None	None
5	12	agePct12t29	Feature	Continuous		Age	None	None
6	13	agePct16t24	Feature	Continuous		Age	None	None
7	14	agePct65up	Feature	Continuous		Age	None	None
8	17	medIncome	Feature	Continuous		Income	None	None
9	18	pctWWage	Feature	Continuous		Income	None	None
16	41	PctOccupManu	Feature	Continuous		Occupation	None	None
17	42	PctOccupMgmtProf	Feature	Continuous		Occupation	None	None
0	7	racePctBlack	Feature	Continuous		Race	None	None
1	8	racePctWhite	Feature	Continuous		Race	None	None
2	9	racePctAsian	Feature	Continuous		Race	None	None
3	10	racePctHisp	Feature	Continuous		Race	None	None
10	26	whitePerCap	Feature	Continuous		Race	None	None
11	27	blackPerCap	Feature	Continuous		Race	None	None
12	28	indianPerCap	Feature	Continuous		Race	None	None
13	29	AsianPerCap	Feature	Continuous		Race	None	None
14	30	OtherPerCap	Feature	Continuous		Race	None	None
15	31	HispPerCap	Feature	Continuous		Race	None	None

▼ Missing_values

```
communities_and_crime.variables[communities_and_crime.variables.missing_values == "y"]
```

	index	name	role	type	demographic	description	units
2	30	OtherPerCap	Feature	Continuous	Race	None	None
3	101	LemasSwornFT	Feature	Continuous	None	None	None
4	102	LemasSwFTPerPop	Feature	Continuous	None	None	None
5	103	LemasSwFTFieldOps	Feature	Continuous	None	None	None
6	104	LemasSwFTFieldPerPop	Feature	Continuous	None	None	None
7	105	LemasTotalReq	Feature	Continuous	None	None	None
8	106	LemasTotReqPerPop	Feature	Continuous	None	None	None
9	107	PolicReqPerOffic	Feature	Continuous	None	None	None
10	108	PolicPerPop	Feature	Continuous	None	None	None
11	109	RacialMatchCommPol	Feature	Continuous	None	None	None
12	110	PctPolicWhite	Feature	Continuous	None	None	None
13	111	PctPolicBlack	Feature	Continuous	None	None	None
14	112	PctPolicHisp	Feature	Continuous	None	None	None
15	113	PctPolicAsian	Feature	Continuous	None	None	None
16	114	PctPolicMinor	Feature	Continuous	None	None	None
17	115	OfficAssgnDrugUnits	Feature	Continuous	None	None	None
18	116	NumKindsDrugsSeiz	Feature	Continuous	None	None	None
19	117	PolicAveOTWorked	Feature	Continuous	None	None	None
20	121	PolicCars	Feature	Continuous	None	None	None
21	122	PolicOperBudg	Feature	Continuous	None	None	None
22	123	LemasPctPolicOnPatr	Feature	Continuous	None	None	None
23	124	LemasGangUnitDeploy	Feature	Continuous	None	None	None
24	126	PolicBudgPerPop	Feature	Continuous	None	None	None
0	1	county	Feature	Integer	None	None	None
1	2	community	Feature	Integer	None	None	None

▼ Data Cleaning

X.shape

→ (1994, 127)

```
missing_features = communities_and_crime.variables[communities_and_crime.variables.m
row_missing_rate = ((X == "?").sum(axis=1) > 0).mean()
```

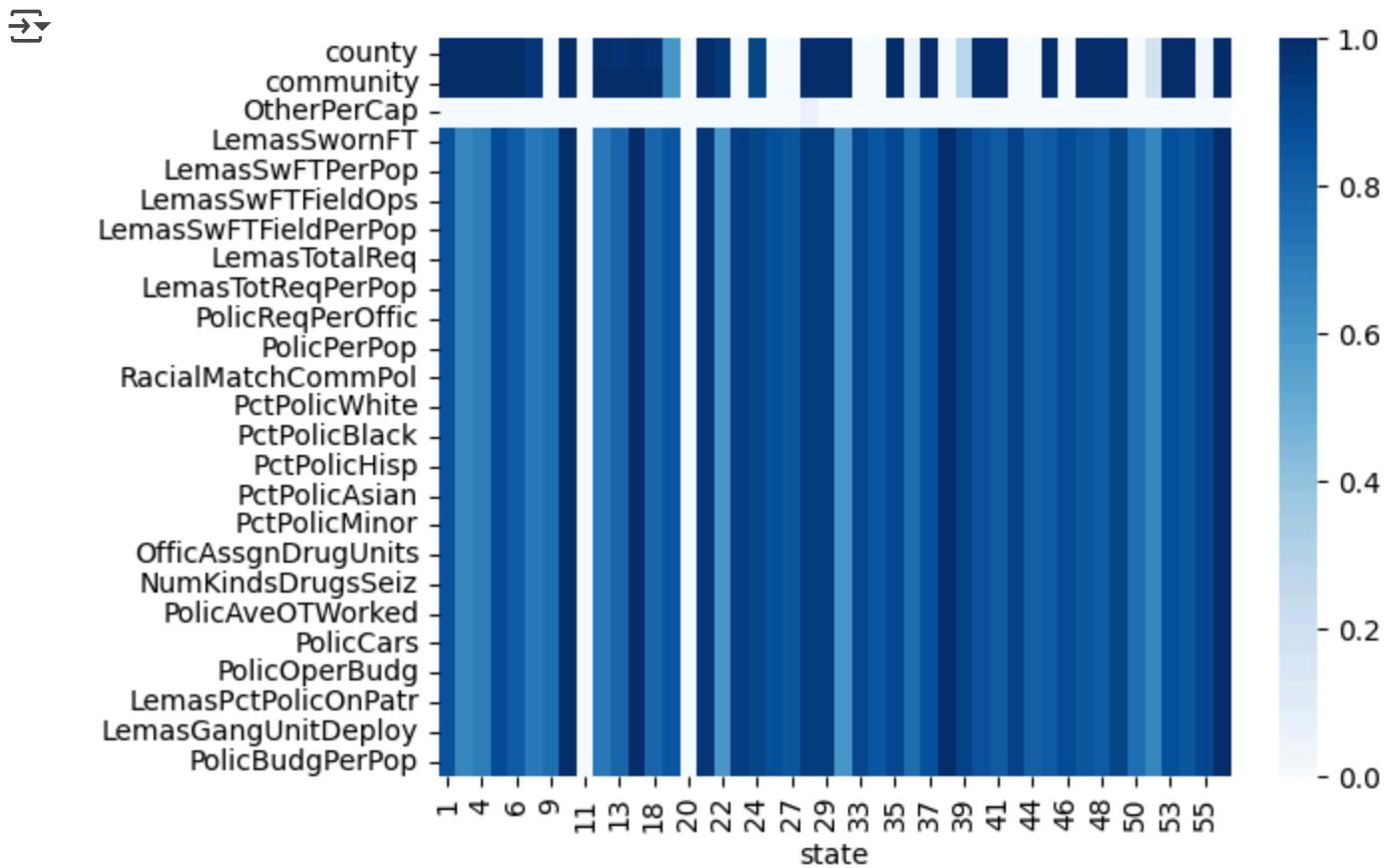
```
for i, column in enumerate(missing_features):
    missing = (X[column] == "?").mean()
    print(f"{i} {column}: {missing:.2f}")
print(f"Total Row missing rate: {row_missing_rate:.2f}")
```

→ 0 county: 0.59
1 community: 0.59
2 OtherPerCap: 0.00
3 LemassSwornFT: 0.84
4 LemassSwFTPPerPop: 0.84
5 LemassSwFTFieldOps: 0.84
6 LemassSwFTFieldPerPop: 0.84
7 LemassTotalReq: 0.84
8 LemastTotReqPerPop: 0.84
9 PolicReqPerOffic: 0.84
10 PolicPerPop: 0.84
11 RacialMatchCommPol: 0.84
12 PctPolicWhite: 0.84
13 PctPolicBlack: 0.84
14 PctPolicHisp: 0.84
15 PctPolicAsian: 0.84
16 PctPolicMinor: 0.84
17 OfficAssgnDrugUnits: 0.84
18 NumKindsDrugsSeiz: 0.84
19 PolicAveOTWorked: 0.84
20 PolicCars: 0.84
21 PolicOperBudg: 0.84
22 LemasPctPolicOnPatr: 0.84
23 LemasGangUnitDeploy: 0.84
24 PolicBudgPerPop: 0.84
Total Row missing rate: 0.94

```
(X.loc[:, missing_features].isna().sum(axis=1) == len(missing_features)).mean()
# There is not a record that miss all 25 features
```

→ 0.0

```
df_miss_feature_and_location = X.groupby("state").apply(lambda subdf: pd.Series(
    {key : round((subdf[key] == "?").mean(), 3) for key in missing_features}
))
sns.heatmap(df_miss_feature_and_location.T, cmap="Blues")
plt.show()
```



```

drop_X = X.drop(["community", "OtherPerCap", "county", "communityname", "fold"], axis=1)
nan_columns = drop_X.select_dtypes(include="object").columns
drop_X[nan_columns] = drop_X[nan_columns].replace("?", np.nan).astype(float)
drop_X[nan_columns] = drop_X[nan_columns].fillna(drop_X[nan_columns].median())

dummies = pd.get_dummies(drop_X.iloc[:, 0], drop_first=True, prefix="state").astype(bool)
n = dummies.shape[1]
fine_X = pd.concat([dummies, drop_X.iloc[:, 1:]], axis=1)

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
import statsmodels as sm
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from statsmodels.api import OLS
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS

```

▼ Problem 1

▼ Question A

- (30 points) What are the most important features? Compare and contrast the top features as determined by:
 - Statistical significance via Least Squares.
 - Best Subsets.
 - Step-wise approaches (and/or Recursive Feature Elimination).
 - Lasso.
 - Elastic Net.

▼ OLS Statistics Significant

```
column_name = fine_X.columns[n+1:]
centered_Y = pd.DataFrame(StandardScaler(with_mean=True).fit_transform(y), columns=[stand_X = pd.concat((fine_X.iloc[:, :n], pd.DataFrame(StandardScaler(with_mean=True,
stand_X
```

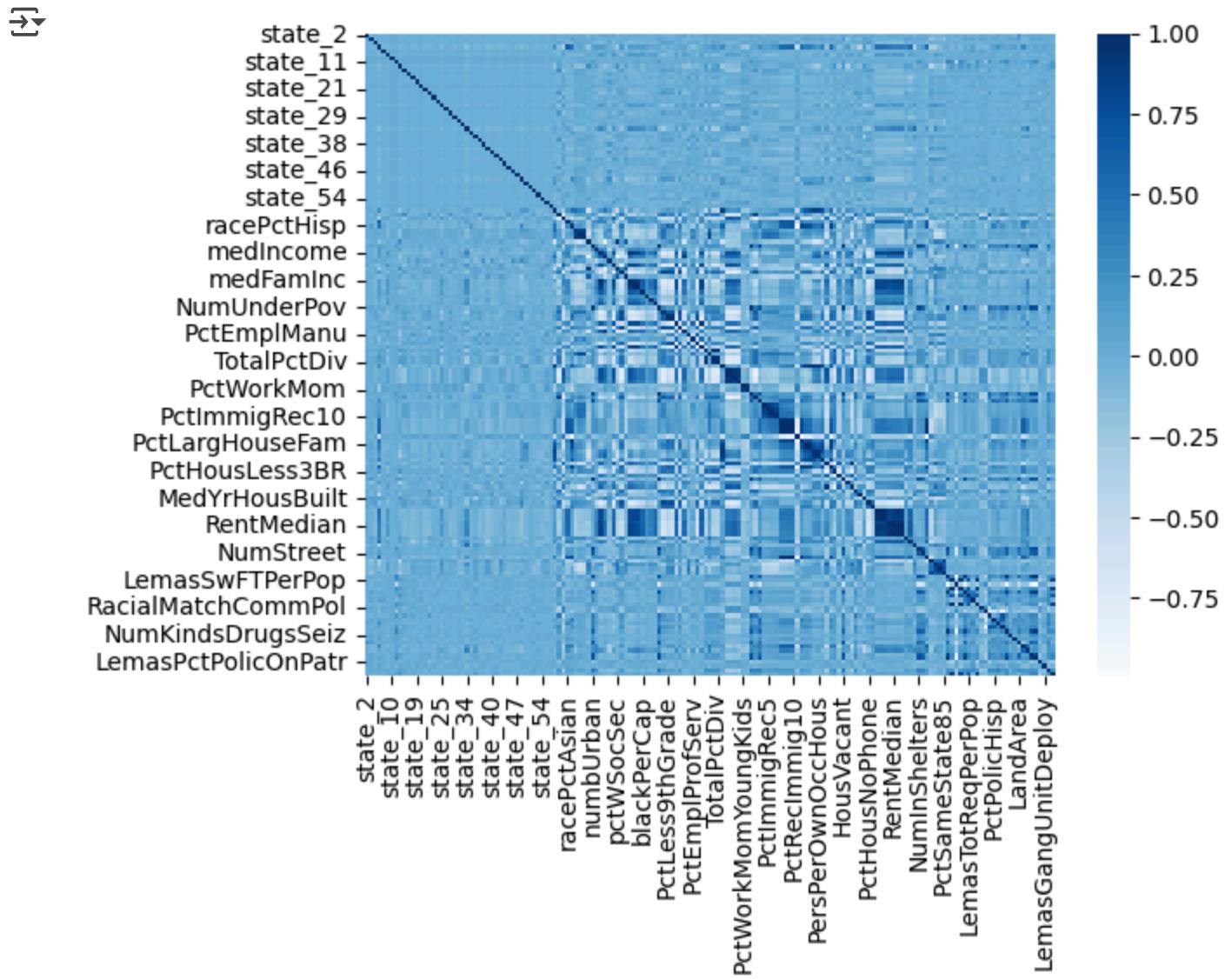
	state_2	state_4	state_5	state_6	state_8	state_9	state_10	state_11	sta
0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
...
1989	0	0	0	0	0	0	0	0	0
1990	0	0	0	1	0	0	0	0	0
1991	0	0	0	0	0	1	0	0	0
1992	0	0	0	0	0	0	0	0	0
1993	0	0	0	1	0	0	0	0	0

1994 rows × 165 columns

```
import seaborn as sns

corr = stand_X.corr()
```

```
sns.heatmap(corr, cmap="Blues")
plt.show()
```



```
np.fill_diagonal(corr.values, np.nan)
lower_tri_indices = np.tril_indices_from(corr.values, k=-1)
lower_corr_values = corr.values[lower_tri_indices]
selected_pairs = pd.DataFrame({
    "Feature 1": corr.index[lower_tri_indices[0]],
    "Feature 2": corr.columns[lower_tri_indices[1]],
    "Correlation": lower_corr_values
})

# Filter correlations > 0.75
selected_pairs = selected_pairs[selected_pairs["Correlation"].abs() > 0.9].res
selected_pairs.shape
```

(56, 3)

```

correlations = np.zeros((stand_X.shape[1], 1))
for i, column in enumerate(stand_X.columns):
    correlations[i, 0] = stand_X[column].corr(centered_Y.iloc[:, 0])
cor_df = pd.DataFrame(correlations, index=stand_X.columns, columns=["correlation"])
top_cor = abs(cor_df).sort_values(by="correlation", ascending=False).head(10)
last_cor = abs(cor_df).sort_values(by="correlation", ascending=False).tail(10)

intercept = pd.DataFrame(np.ones((stand_X.shape[0], 1)), columns=["intercept"])
# not top_cor_features
lsmodel_not = OLS(centered_Y, stand_X).fit()
result_ls_not = np.column_stack((lsmodel_not.params.index, lsmodel_not.params.values))
result_ls_not = result_ls_not[result_ls_not[:, 2].argsort()][:10, :2]

# top_10_cor_features
lsmodel_yes = OLS(centered_Y, stand_X.loc[:, top_cor.index]).fit()
result_ls_yes = np.column_stack((lsmodel_yes.params.index, lsmodel_yes.params.values))
result_ls_yes = result_ls_yes[result_ls_yes[:, 2].argsort()][:, :2]

results_ls = np.column_stack((result_ls_not, result_ls_yes))
params_ls = pd.DataFrame(results_ls, columns=["Name_Not", "Params_Not", "Name_Yes", "Params_Yes"])
print("Top Cor Params:\n", list(top_cor.index))
params_ls

```

→ Top Cor Params:

['PctKids2Par', 'PctIlleg', 'PctFam2Par', 'racePctWhite', 'PctYoungKids2Par', '...

	Name_Not	Params_Not	Name_Yes	Params_Yes	
0	state_28	-0.978944	PctIlleg	0.297512	
1	state_12	0.582829	racePctWhite	-0.215177	
2	state_13	-0.545023	FemalePctDiv	0.162579	
3	state_51	-0.540313	PctKids2Par	-0.660482	
4	state_42	-0.3721	PctFam2Par	0.379547	
5	state_6	0.376383	pctWInvInc	0.035502	
6	pctUrban	0.071043	racepctblack	-0.033854	
7	MedRent	0.423081	PctYoungKids2Par	0.022763	
8	PctWorkMom	-0.134557	pctWPubAsst	0.011648	
9	state_39	-0.227767	PctTeen2Par	0.014326	

后续步骤：

[使用 params_ls 生成代码](#)

[查看推荐的图表](#)

[New interactive sheet](#)

Best Subsets

```
model_linear = LinearRegression(fit_intercept=False)
efs = EFS(
    model_linear,
    min_features=1,
    max_features=10,
    scoring="neg_mean_squared_error", #we use negative mse since the sfs method maxi
    cv=0
)
```

efs.fit(stand_X.loc[:, top_cor.index], centered_Y)

efs_summary = pd.DataFrame.from_dict(efs.get_metric_dict()).T[["feature_idx", "avg_s

→ Features: 1023/1023

```
efs_size = np.zeros(len(efs_summary))
for i in range(len(efs_summary)):
    efs_size[i] = len(efs_summary["feature_idx"][i])
efs_summary['model size'] = efs_size
efs_summary.rename(columns={'avg_score': 'neg_mse'}, inplace=True)
best_features = efs_summary.sort_values(by="neg_mse", ascending=False).iloc[0]
result_efs = top_cor.iloc[list(best_features["feature_idx"])]
params_best = result_efs.reset_index().rename(columns={"index": "Name_Yes", "correla
params_best
```

	Name_Yes	value_Yes(Correlation)	
0	PctKids2Par	0.738424	
1	PctIlleg	0.737957	
2	PctFam2Par	0.706667	
3	racePctWhite	0.684770	
4	PctYoungKids2Par	0.666059	
5	PctTeen2Par	0.661582	
6	racepctblack	0.631264	
7	pctWInvInc	0.576324	
8	pctWPubAsst	0.574665	
9	FemalePctDiv	0.556032	

后续步骤:

[使用 params_best 生成代码](#)

[查看推荐的图表](#)

[New interactive sheet](#)

✓ Step-Wise Approaches

```
#defining aic evaluation functions compatible with mlxtend.feature_selection.SequentialFeatureSelection
def calculate_aic(estimator, X, y):
    """
    Custom AIC scorer for SequentialFeatureSelector.

    Args:
        estimator: A fitted sklearn-compatible estimator.
        X: Features (numpy array).
        y: Target variable (numpy array).

    Returns:
        Negative AIC value for compatibility with SFS (higher is better).
    """
    n, k = X.shape # n: number of samples, k: number of predictors
    y_pred = estimator.predict(X)
    residual_sum_of_squares = np.sum((y - y_pred) ** 2)
    aic = n * np.log(residual_sum_of_squares / n) + 2 * k
    return -aic # SFS maximizes the score

def aic_scorer_wrapper(estimator, X, y):
    estimator.fit(X, y)
    return calculate_aic(estimator, X, y)

sfs = SFS(
    estimator=LinearRegression(fit_intercept=False),
    k_features=(1, 10),
    forward=True,
    floating=False,
    scoring=aic_scorer_wrapper,
    cv=0
)

sfs.fit(stand_X.loc[:, :], centered_Y)
results_Not = max([value for _, value in sfs.get_metric_dict().items()], key=lambda
results_Not = np.array(results_Not)

sfs.fit(stand_X.loc[:, top_cor.index], centered_Y)
results_Yes = max([value for _, value in sfs.get_metric_dict().items()], key=lambda
results_Yes = {
    "Name_Yes": pd.Series(results_Yes)
}

params_step = pd.concat((pd.DataFrame(results_Yes), pd.DataFrame(results_Not, columns=[0]))[0], axis=1)
```

	Name_Yes	Name_Not	
0	PctKids2Par	state_12	
1	PctIlleg	state_13	
2	PctFam2Par	state_25	
3	racePctWhite	state_28	
4	FemalePctDiv	racePctWhite	
5	NaN	MalePctDivorce	
6	NaN	PctKids2Par	
7	NaN	PctWorkMom	
8	NaN	PctIlleg	
9	NaN	HousVacant	

后续步骤:

[使用 params_step 生成代码](#)[查看推荐的图表](#)[New interactive sheet](#)

▼ LASSO

```

lasso = Lasso(fit_intercept=False, alpha=0.07)
lasso.fit(stand_X.loc[:, :], centered_Y.iloc[:, 0])
# print(pd.DataFrame(np.stack((fine_X.columns.values, fit0.coef_, lasso.coef_), axis
index_lasso_not = np.where(lasso.coef_ != 0)
result_lasso_not = np.column_stack((stand_X.columns[index_lasso_not], lasso.coef_[in
temp1 = pd.DataFrame(result_lasso_not, columns=["Name_Not", "Value_Not"], index=rang

lasso.fit(stand_X.loc[:, top_cor.index], centered_Y.iloc[:, 0])
index_lasso_yes = np.where(lasso.coef_ != 0)
result_lasso_yes = {
    "Name_Yes": pd.Series(top_cor.index[index_lasso_yes]),
    "Value_Yes": pd.Series(lasso.coef_[index_lasso_yes])
}

params_lasso = pd.concat((pd.DataFrame(result_lasso_yes), temp1), axis=1)
params_lasso

```

	Name_Yes	Value_Yes	Name_Not	Value_Not	
0	PctKids2Par	-0.266173	racePctWhite	-0.185912	
1	PctIlleg	0.224178	pctUrban	0.006777	
2	racePctWhite	-0.212139	MalePctDivorce	0.064956	
3	FemalePctDiv	0.079820	PctKids2Par	-0.266554	
4	NaN	NaN	PctWorkMom	-0.004336	
5	NaN	NaN	PctIlleg	0.192499	
6	NaN	NaN	PctPersDenseHous	0.044448	
7	NaN	NaN	HousVacant	0.08634	
8	NaN	NaN	PctVacantBoarded	0.005694	
9	NaN	NaN	NumStreet	0.035718	

后续步骤:

[使用 params_lasso 生成代码](#)[查看推荐的图表](#)[New interactive sheet](#)

▼ Elastic Net

```

elasticnet = ElasticNet(alpha=1, l1_ratio = 0.29, fit_intercept=False)
elasticnet.fit(stand_X.loc[:, :], centered_Y.iloc[:, 0])
index_en_not = np.where(~np.isclose(elasticnet.coef_, 0))
result_en_not = np.column_stack((stand_X.columns.values[index_en_not], elasticnet.co

elasticnet.fit(stand_X.loc[:, top_cor.index], centered_Y.iloc[:, 0])
index_en_yes = np.where(~np.isclose(elasticnet.coef_, 0))
result_en_yes = {
    "Name_Yes": pd.Series(top_cor.index[index_en_yes]),
    "Value_Yes": pd.Series(elasticnet.coef_[index_en_yes])
}
params_en = pd.concat((pd.DataFrame(result_en_yes), pd.DataFrame(result_en_not, colu
params_en

```

	Name_Yes	Value_Yes	Name_Not	Value_Not	
0	PctKids2Par	-0.095074	racepctblack	0.018048	
1	PctIlleg	0.102120	racePctWhite	-0.089296	
2	PctFam2Par	-0.063209	pctWInvInc	-0.001308	
3	racePctWhite	-0.089257	FemalePctDiv	0.003679	
4	PctYoungKids2Par	-0.030703	TotalPctDiv	0.001234	
5	PctTeen2Par	-0.028707	PctFam2Par	-0.06309	
6	racepctblack	0.018050	PctKids2Par	-0.094948	
7	pctWInvInc	-0.001408	PctYoungKids2Par	-0.030612	
8	FemalePctDiv	0.004179	PctTeen2Par	-0.0286	
9	NaN	NaN	PctIlleg	0.102142	

后续步骤:

[使用 params_en 生成代码](#)[查看推荐的图表](#)[New interactive sheet](#)

▼ Results:

```

params_dfs = [params_ls, params_best, params_step, params_lasso, params_en]

params_ls.columns = pd.MultiIndex.from_product([["Least_Square"], params_ls.columns])
params_best.columns = pd.MultiIndex.from_product([["Best_Subsets"], params_best.columns])
params_step.columns = pd.MultiIndex.from_product([["Step_Wise"], params_step.columns])
params_lasso.columns = pd.MultiIndex.from_product([["Lasso"], params_lasso.columns])
params_en.columns = pd.MultiIndex.from_product([["Elastic_Net"], params_en.columns])
print("'Not' means 'selected features from all', 'Yes' means 'selected features from"
result = pd.concat([params_ls, params_best, params_step, params_lasso, params_en], axis=1)
a
result

```



后续步骤:

[使用 result 生成代码](#)[查看推荐的图表](#)[New interactive sheet](#)

✓ (ii) Fit and visualize regularization paths for the following methods:

- Lasso
- Elastic Net (at two separate α 's).
- Ridge.

```
def model_features_returner(column):  
    i, j = column  
    model, feature = None, None  
    if i == "Least_Square":  
        return LinearRegression, result[column].dropna().values  
    elif i == "Best_Subsets":  
        return LinearRegression, result[column].dropna().values  
    elif i == "Step_Wise":  
        return LinearRegression, result[column].dropna().values  
    elif i == "Lasso":  
        return Lasso, result[column].dropna().values  
    elif i == "Elastic_Net":  
        return ElasticNet, result[column].dropna().values  
    else:  
        return Ridge, top_cor.index
```

```
def regularization_path(X, Y, model, features, l1_ratio=0.1):
```

```
lambdas = np.exp(np.linspace(np.log(0.01), np.log(10000), 500))
betasl = np.zeros((len(lambdas), X.loc[:, features].shape[1]))
for i, lamb in enumerate(lambdas):
    if model == ElasticNet:
        m = model(alpha = lamb, l1_ratio = l1_ratio, fit_intercept=False)
    else:
        m = model(alpha = lamb)
    m.fit(X.loc[:, features], Y)
    betasl[i, :] = m.coef_

# Plot Lasso paths (log-scale)
plt.figure(figsize=(10, 6))
for j in range(X.loc[:, features].shape[1]): #for each variable
    plt.plot(lambdas, betasl[:, j], label=f"Variable {j+1}")
plt.xscale("log")
plt.xlabel("Lambda")
plt.ylabel("Coefficients")
plt.title(f"{type(m).__name__} Regression Coefficient Paths")
if len(features) <= 10:
    plt.legend(features)
plt.axhline(y=0, color="black", linewidth=0.5)
plt.show()

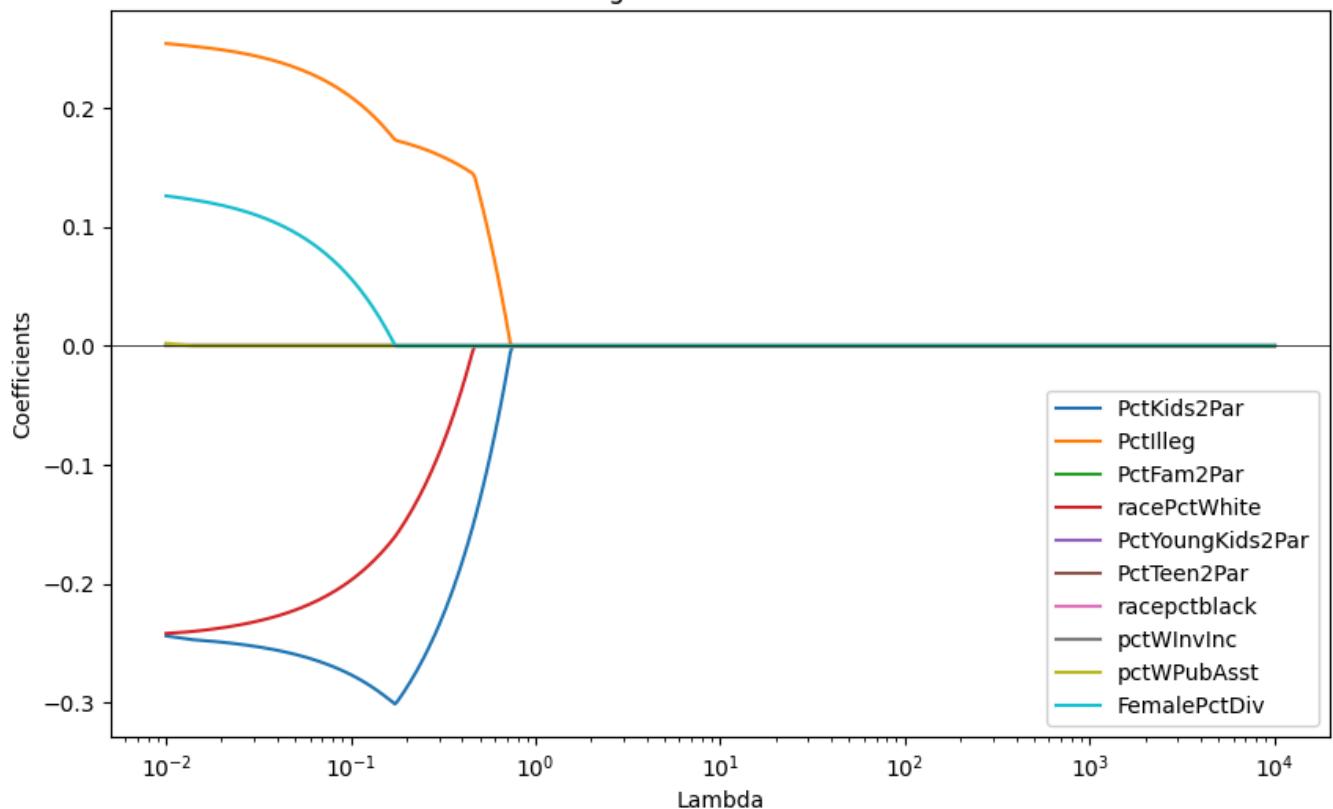
# columns = list(filter(lambda x: (x[0] in ["Lasso", "Elastic_Net"]) and (x[1] in [
# for column in columns:
#     model, features = model_features_returner(column)
#     regularization_path(stand_X, centered_Y, model, features)

regularization_path(stand_X, centered_Y, Lasso, top_cor.index)
regularization_path(stand_X, centered_Y, ElasticNet, top_cor.index, l1_ratio=0.2)
regularization_path(stand_X, centered_Y, Ridge, top_cor.index)

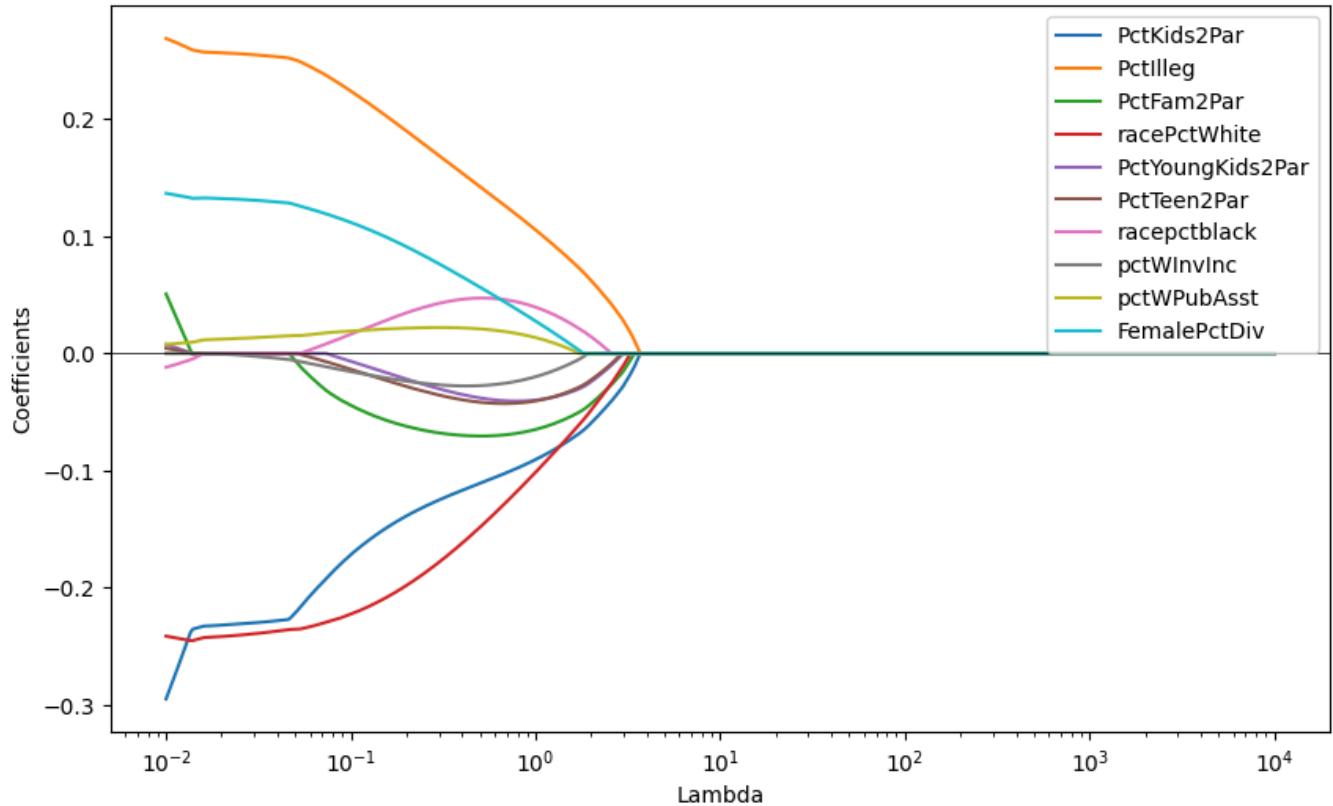
regularization_path(stand_X, centered_Y, Lasso, stand_X.columns)
regularization_path(stand_X, centered_Y, ElasticNet, stand_X.columns, l1_ratio=0.2)
regularization_path(stand_X, centered_Y, Ridge, stand_X.columns)
```



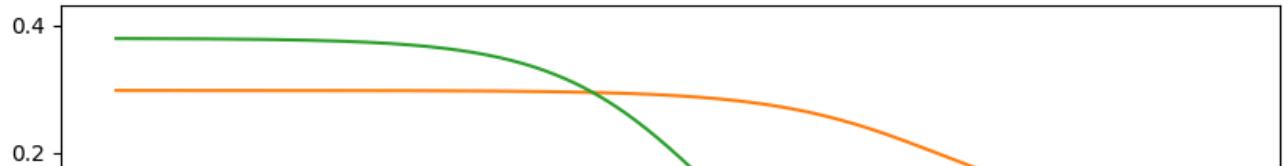
Lasso Regression Coefficient Paths

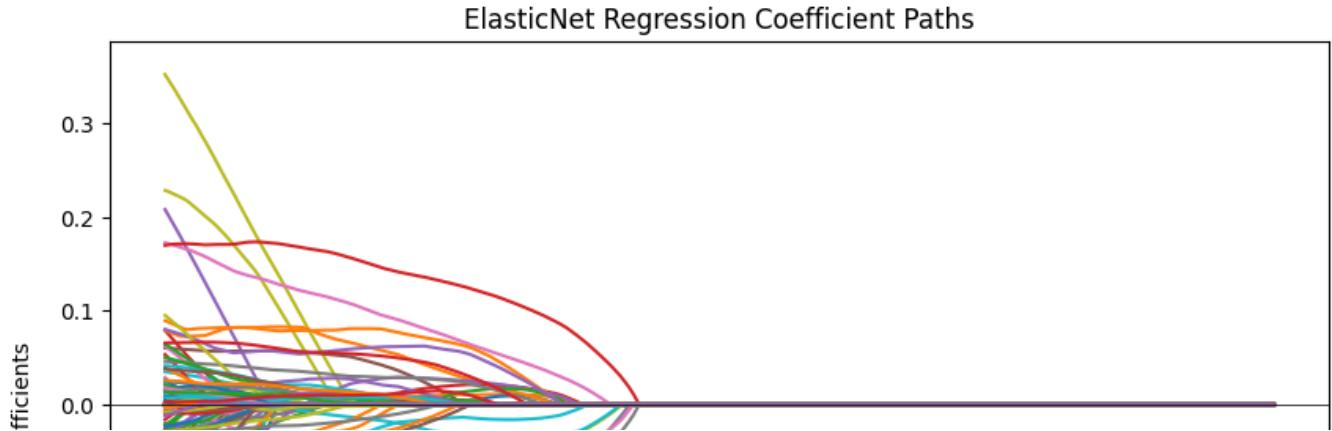
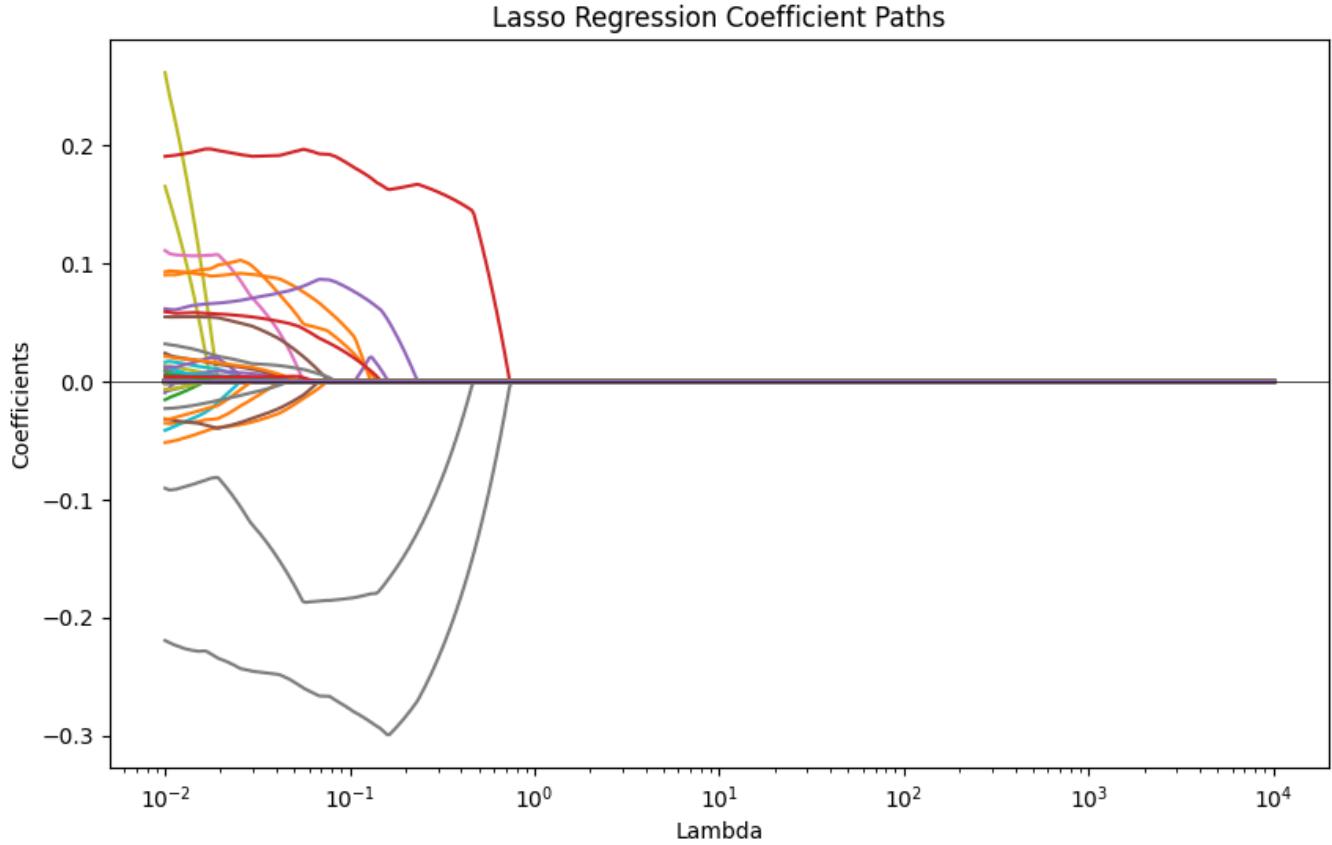
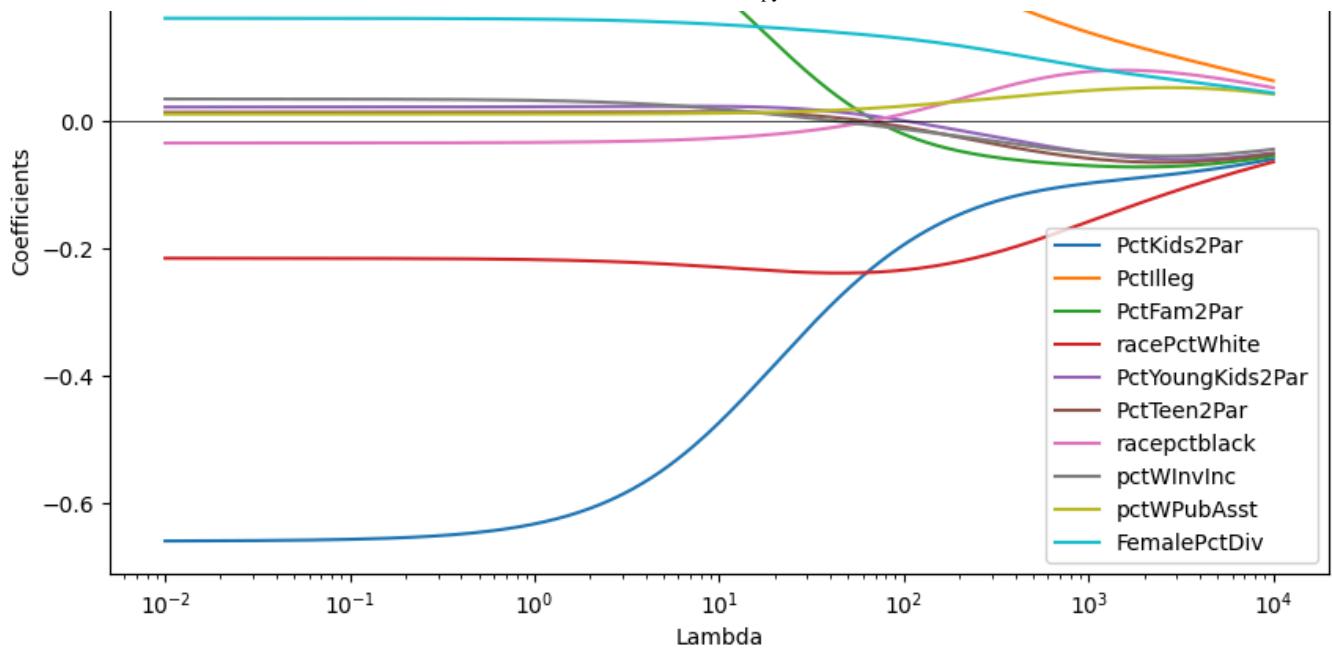


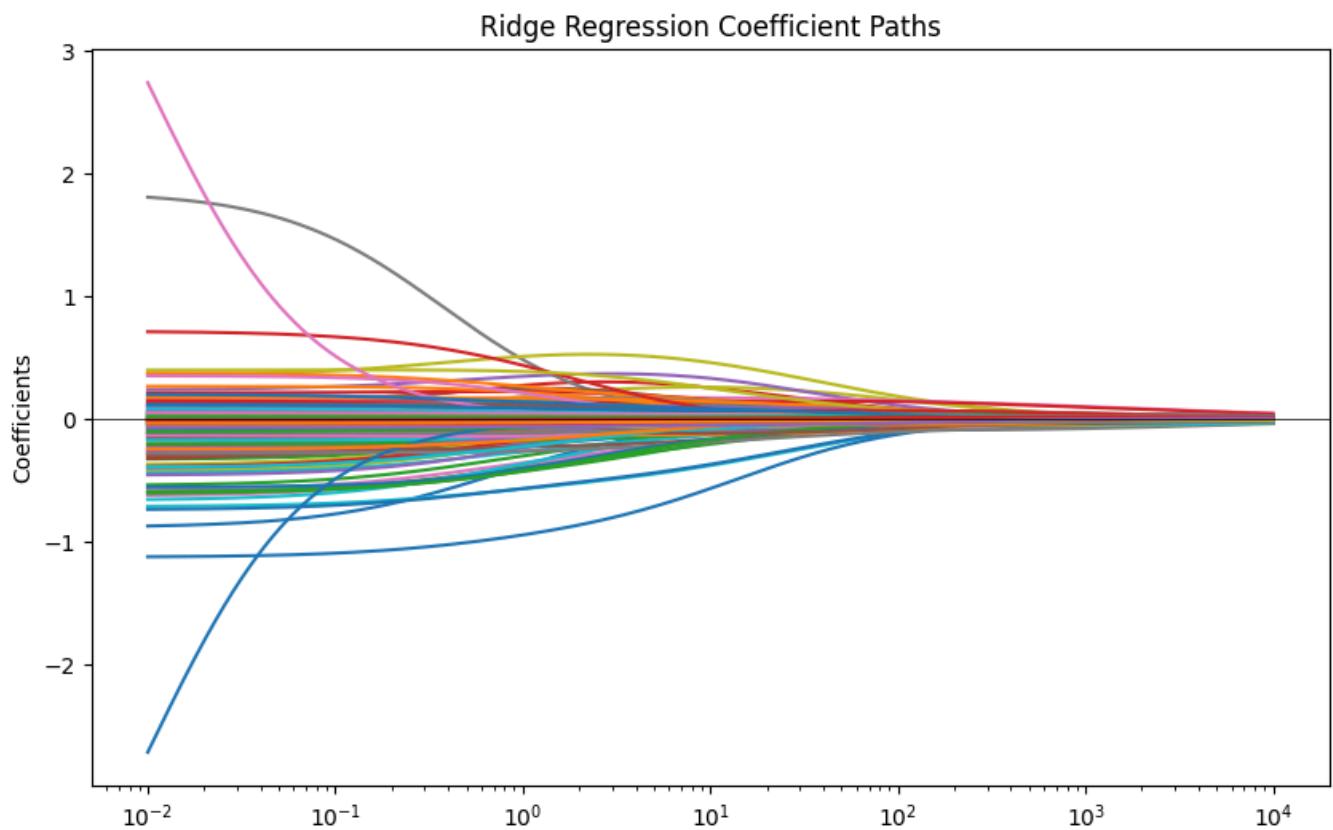
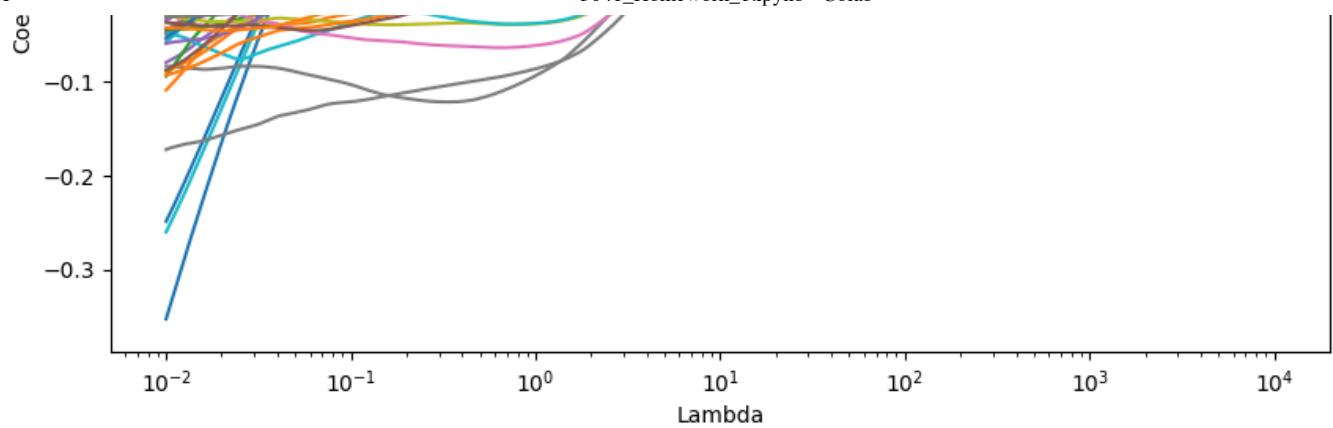
ElasticNet Regression Coefficient Paths



Ridge Regression Coefficient Paths







```
stand_X["FemalePctDiv"].corr(stand_X["PctKids2Par"])
```

→ -0.7299217835356265

▼ (iii) Reflect on these results.

- Are the top features different for each method? Why or why not? If there are tuning parameters, how did you determine these? Do different tuning parameters yield different important features? Are there any features consistently selected by all methods? What are the most important features and how did you determine this? Explain and expand on your responses.

result

→

	Least_Square			Best_Subsets		
	Name_Not	Params_Not	Name_Yes	Params_Yes	Name_Yes	value_Yes(
0	state_28	-0.978944	PctIlleg	0.297512	PctKids2Par	
1	state_12	0.582829	racePctWhite	-0.215177	PctIlleg	
2	state_13	-0.545023	FemalePctDiv	0.162579	PctFam2Par	
3	state_51	-0.540313	PctKids2Par	-0.660482	racePctWhite	
4	state_42	-0.3721	PctFam2Par	0.379547	PctYoungKids2Par	
5	state_6	0.376383	pctWInvInc	0.035502	PctTeen2Par	
6	pctUrban	0.071043	racepctblack	-0.033854	racepctblack	
7	MedRent	0.423081	PctYoungKids2Par	0.022763	pctWInvInc	
8	PctWorkMom	-0.134557	pctWPubAsst	0.011648	pctWPubAsst	
9	state_39	-0.227767	PctTeen2Par	0.014326	FemalePctDiv	

后续步骤:

[使用 result 生成代码](#)

[查看推荐的图表](#)

[New interactive sheet](#)

```
count = dict()
for i in result:
    if i[1] != "Name_Not":
        continue
    for j in range(10):
        if result.loc[j, i] not in count.keys():
            count[result.loc[j, i]] = 1
        else:
```

```

count[result.loc[j, i]] += 1
# for i in top_cor.index:
#     if i not in count.keys():
#         count[i] = 1
#     else:
#         count[i] += 1

r = sorted(count.items(), key=lambda x: x[1], reverse=True)
final = pd.DataFrame(r, columns=["Feature", "Count"])

```

Reflecting on the feature selection results, we observe that different methods prioritize different features due to their unique selection criteria.

- **OLS identifies features based on statistical significance**, selecting those with low p-values. In my selected params, most are state dummies. However, other models do not quite find state dummies significant, except for step-wise model. And LS model does not choose features ('PctKids2Par', 'PctIlleg', 'racePctWhite') which are chosen by all other models.
- **LASSO selects a sparse set of features**, dropping those with high collinearity. Based on the regularization paths with top correlated features, it shows that "PctKids2Par" increases as "FemalePctDiv" decreases to zero, indicating their high correlation. Finally, all betas shrink to zero. Different lambda to be set will lead to different number and value of betas.
- **Ridge retains all features but shrinks them differently depending on multicollinearity**. Unlike Lasso and Elastic_Net, Ridge won't make betas ultimately to zeros. Instead, it forces betas to converge into several groups.
- **Best Subsets chooses a feature set optimizing predictive accuracy**. It compares all combinations of betas, which takes much longer time.
- **Stepwise approaches iteratively adding features to improve model fit**. It returns the results much faster than the Best Subset model, but it won't check all possible combinations.
- **Elastic Net balances LASSO and Ridge behaviors, selecting features based on both sparsity and correlation handling**. Because it contains norm 1 penalty, it did shrink betas to zero finally, but due to the l1_ratio it shrunk betas with slow pace.

Different tuning parameters led to different selected features, especially in **LASSO and Elastic Net**, where higher regularization forces more coefficients to zero. However, some features were **consistently selected across all methods**, indicating their strong predictive power.

Most Important Features: After comparing all methods and getting 27 significant features. The most consistently selected and impactful features are:

- '**PctKids2Par', 'PctIlleg', 'racePctWhite**': Selected in four models.
- '**PctWorkMom**': Selected in three models.

- 'state_28', 'state_12', 'state_13', 'pctUrban', 'MalePctDivorce', 'HousVacant', 'racepctblack', 'pctWlInvInc', 'FemalePctDiv', 'PctFam2Par', 'PctYoungKids2Par', 'PctTeen2Par': Selected in two models.
- 'state_51', 'state_42', 'state_6', 'MedRent', 'state_39', 'state_25', 'PctPersDenseHous', 'PctVacantBoarded', 'NumStreet', 'TotalPctDiv', 'pctWPubAsst': Selected in one model.

Thus, "PctKids2Par", "PctIlleg", "racePctWhite can be considered the most critical features, as they remain important regardless of the selection method used.

▼ Problem b

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV, ElasticNetCV
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
import numpy as np

# Split data into 60% train, 20% validation, 20% test
def split_data(X, y):
    X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
    return X_train, X_val, X_test, y_train, y_val, y_test

# Number of repetitions
n_repeats = 10
test_errors = {
    "OLS": {"validation": [], "test": []},
    "Ridge": {"validation": [], "test": []},
    "Best": {"validation": [], "test": []},
    "Step": {"validation": [], "test": []},
    "LASSO": {"validation": [], "test": []},
    "ElasticNet": {"validation": [], "test": []},
    "Mean": {"validation": [], "test": []}
}

for _ in range(n_repeats):
    # Split the data
    X_train, X_val, X_test, y_train, y_val, y_test = split_data(stand_X, centered_Y)

    # Mean
    test_errors["Mean"]["validation"].append(mean_squared_error(y_val, np.mean(y_train)))
    test_errors["Mean"]["test"].append(mean_squared_error(y_test, np.mean(y_train)) * 2)

    # OLS
    ols = LinearRegression().fit(X_train, y_train)
    test_errors["OLS"]["validation"].append(mean_squared_error(y_val, ols.predict(X_val)))
    test_errors["OLS"]["test"].append(mean_squared_error(y_test, ols.predict(X_test)))
```

```
# Ridge (chooses best alpha using validation set)
ridge = RidgeCV(alphas=np.logspace(-3, 3, 10), store_cv_values=True).fit(X_train)
test_errors["Ridge"]["validation"].append(mean_squared_error(y_val, ridge.predict(X_val)))
test_errors["Ridge"]["test"].append(mean_squared_error(y_test, ridge.predict(X_test)))

# # LASSO (chooses best alpha using validation set)
lasso = LassoCV(alphas=np.logspace(-3, 3, 10), cv=5).fit(X_train, y_train)
test_errors["LASSO"]["validation"].append(mean_squared_error(y_val, lasso.predict(X_val)))
test_errors["LASSO"]["test"].append(mean_squared_error(y_test, lasso.predict(X_test)))

# # Elastic Net (chooses best alpha and L1 ratio using validation set)
elastic_net = ElasticNetCV(l1_ratio=[0.1, 0.5, 0.9], alphas=np.logspace(-3, 3, 10)).fit(X_train, y_train)
test_errors["ElasticNet"]["validation"].append(mean_squared_error(y_val, elastic_net.predict(X_val)))
test_errors["ElasticNet"]["test"].append(mean_squared_error(y_test, elastic_net.predict(X_test)))

# # Step Wise
sfs.fit(X_train, y_train)
best_features = list(sfs.k_feature_idx_)
step = LinearRegression().fit(X_train.iloc[:, best_features], y_train)
test_errors["Step"]["validation"].append(mean_squared_error(y_val, step.predict(X_val)))
test_errors["Step"]["test"].append(mean_squared_error(y_test, step.predict(X_test)))

# # Best Subsets
efs.fit(X_train.loc[:, top_cor.index], y_train)
best_features = list(efs.best_idx_)
best = LinearRegression().fit(X_train.iloc[:, best_features], y_train)
test_errors["Best"]["validation"].append(mean_squared_error(y_val, best.predict(X_val)))
test_errors["Best"]["test"].append(mean_squared_error(y_test, best.predict(X_test)))
print(f"\nRound-{n+1} Done.")

# Compute average test error for each method
avg_test_errors = {
    model: {types: np.mean(values) for types, values in items.items()}
    for model, items in test_errors.items()
}
avg_test_errors = pd.DataFrame.from_dict(avg_test_errors).T
```

→ Features: 1023/1023
 Round-1 Done.
 Features: 1023/1023
 Round-2 Done.
 Features: 1023/1023
 Round-3 Done.
 Features: 1023/1023
 Round-4 Done.
 Features: 1023/1023
 Round-5 Done.
 Features: 1023/1023
 Round-6 Done.
 Features: 1023/1023
 Round-7 Done.
 Features: 1023/1023
 Round-8 Done.
 Features: 1023/1023
 Round-9 Done.
 Features: 1023/1023
 Round-10 Done.

	validation	test	
OLS	0.335173	0.367681	
Ridge	0.315781	0.352137	
Best	0.909232	0.962026	
Step	0.321264	0.358342	
LASSO	0.314434	0.347623	
ElasticNet	0.312482	0.349480	
Mean	0.990844	1.056437	

后续步骤:

[使用 avg_test_errors 生成代码](#)[查看推荐的图表](#)[New interactive sheet](#)

```
avg_test_errors.sort_values(by="test", ascending=False)
```

	validation	test	
Mean	0.990844	1.056437	
Best	0.909232	0.962026	
OLS	0.335173	0.367681	
Step	0.321264	0.358342	
Ridge	0.315781	0.352137	
ElasticNet	0.312482	0.349480	
LASSO	0.314434	0.347623	

```
avg_test_errors.sort_values(by="validation", ascending=False)
```

	validation	test	grid
Mean	0.990844	1.056437	
Best	0.909232	0.962026	
OLS	0.335173	0.367681	
Step	0.321264	0.358342	
Ridge	0.315781	0.352137	
LASSO	0.314434	0.347623	
ElasticNet	0.312482	0.349480	

```
import matplotlib.pyplot as plt
import seaborn as sns

validation_sets = {model: values["validation"] for model, values in test_errors.items()}
test_sets = {model: values["test"] for model, values in test_errors.items()}
df_va = pd.DataFrame.from_dict(validation_sets)
df_te = pd.DataFrame.from_dict(test_sets)

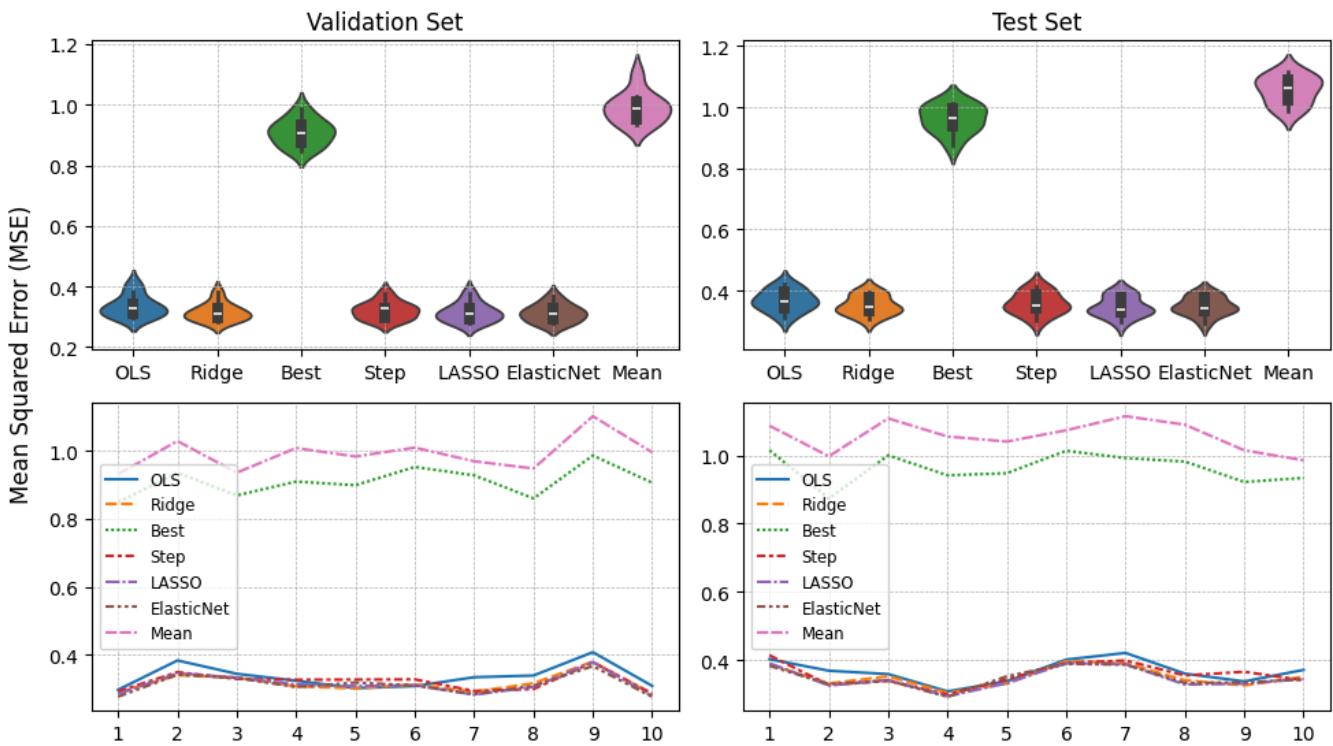
fig, ax = plt.subplots(2, 2, figsize=(10, 6))
fig.suptitle("MSE in Different Models and Rounds.", fontsize=14)
fig.supylabel("Mean Squared Error (MSE)")

sns.violinplot(data=df_va, ax=ax[0, 0])
sns.violinplot(data=df_te, ax=ax[0, 1])
sns.lineplot(data=df_va, ax=ax[1, 0])
sns.lineplot(data=df_te, ax=ax[1, 1])

ax[0, 0].set_title("Validation Set")
ax[0, 1].set_title("Test Set")
ax[1, 1].legend(loc="center left", fontsize="small")
ax[1, 0].legend(loc="center left", fontsize="small")
ax[1, 0].set_xticks(range(0, 10)) # Keep the original tick positions
ax[1, 0].set_xticklabels(range(1, 11)) # Change labels to start from 1
ax[1, 1].set_xticks(range(0, 10)) # Keep the original tick positions
ax[1, 1].set_xticklabels(range(1, 11)) # Change labels to start from 1
for i in range(2):
    for j in range(2):
        ax[i, j].grid(linestyle="--", linewidth=0.5)
plt.tight_layout()
plt.show()
```



MSE in Different Models and Rounds.



```
print(f"""
The MSE of these models based on validation starting at least to highest is:
ElasticNet < Lasso < Ridge < Step < OLS < Best < Mean
The MSE of these models based on test starting at least to highest is:
Lasso < ElasticNet < Ridge < Step < OLS < Best < Mean
```

So based on these results, we can conclude that:
 Elastic Net and LASSO achieve the best prediction error because they effectively combine the advantages of LASSO (feature selection) and Ridge (handling multicollinearity). Additionally, there are {selected_pairs.shape[0]} highly correlated features in the dataset, where the absolute correlation exceeds 0.9. This makes regularization methods particularly useful in reducing overfitting.

From the graphs, it is difficult to clearly identify overfitting. However, OLS tends to overfit when the number of parameters exceeds the number of records ($p > n$). In this case, where X has shape {stand_X.shape}, this is not an issue.

Since LASSO, Ridge, and Elastic Net include regularization, they cannot perform worse than OLS in terms of generalization. However, Stepwise Selection and Best Subset Selection are more prone to overfitting because they may select too many parameters. Additionally, these methods are computationally expensive, so I limited their result size to ensure efficiency.

Conclusion:

Lasso is the overall best method for prediction on this dataset.
 """)

- The MSE of these models based on validation starting at least to highest is:
 ElasticNet < Lasso < Ridge < Step < OLS < Best < Mean
 The MSE of these models based on test starting at least to highest is:
 Lasso < ElasticNet < Ridge < Step < OLS < Best < Mean

So based on these results, we can conclude that:

Elastic Net and LASSO achieve the best prediction error because they effectively combine the advantages of LASSO (feature selection) and Ridge (handling multicollinearity). Additionally, there are 56 highly correlated features in the dataset, where the absolute correlation exceeds 0.9. This makes regularization methods particularly useful in reducing overfitting.

From the graphs, it is difficult to clearly identify overfitting. However, OLS tends to overfit when the number of parameters exceeds the number of records ($p > n$). In this case, where X has shape (1994, 165), this is not an issue.

Since LASSO, Ridge, and Elastic Net include regularization, they cannot perform worse than OLS in terms of generalization. However, Stepwise Selection and Best Subset Selection are more prone to overfitting because they may select too many parameters. Additionally, these methods are computationally expensive, so I limited their result size to ensure efficiency.

Conclusion:

Lasso is the overall best method for prediction on this dataset.

▼ Question 2

- ▼ (a) (10 points) Empirically demonstrate or mathematically show that fitting linear regression with an intercept term is equivalent to

- (i) fitting linear regression when centering Y and centering the columns of X, and
- (ii) fitting linear regression when adding a column of ones to X.

```
import numpy as np
import pandas as pandas
import statsmodels.api as sm
import sklearn.metrics as metric

np.random.seed(89)
beta_true = np.array([2, 3, 5, 4]).reshape(-1, 1)
x_emperical = np.column_stack((np.random.uniform(0, 100, 1000), np.random.norm(x_emperical = sm.add_constant(x_emperical)
y_emperical = x_emperical @ beta_true + np.random.normal(0, 500, size=(1000, 1
y_emperical_center = y_emperical - y_emperical.mean()
```

```

x_emperical_center = (x_emperical - x_emperical.mean(axis=0))

model1 = sm.OLS(y_emperical, x_emperical)
result1 = model1.fit()

model2 = sm.OLS(y_emperical_center, x_emperical_center)
result2 = model2.fit()

result2.params, result1.params

y_emperical_pred_1 = x_emperical @ result1.params.reshape(-1, 1)
y_emperical_pred_2 = x_emperical_center @ result2.params.reshape(-1, 1)

if np.isclose((result1.params[1:] - result2.params[1:]), 0).mean() == 1:
    print("these models predict the slope in same values.\n")
else:
    print("these models dont predict the slope in same values.\n")
print(f"true_beta: {beta_true.flatten()}")
print(f"(i)\tmodel1: {result2.params}")
print(f"(ii)\tmodel2: {result1.params}")

print(f"MSE of model1: {metric.mean_squared_error(y_emperical, y_emperical_pre
print(f"MSE of model2: {metric.mean_squared_error(y_emperical_center, y_emperi

```

→ these models predict the slope in same values.

```

true_beta: [2 3 5 4]
(i)      model1: [0.          4.09753977 5.17005687 9.37841763]
(ii)     model2: [-86.18920724   4.09753977   5.17005687   9.37841763]
MSE of model1: 243404.143
MSE of model2: 243404.143

```

- ✓ (b) (10 points) Empirically demonstrate or mathematically show that the least squares solution has zero training error when $p > n$.

```

import random

np.random.seed(89)
x_emperical = np.random.normal(random.random()*100, random.random()*1000, size
for i in range(14):
    x_emperical = np.column_stack((x_emperical, np.random.normal(random.random()

beta_true2 = np.random.randint(5, 10, size=(16, 1))
x_emperical = sm.add_constant(x_emperical)
y_emperical = x_emperical @ beta_true2 + np.random.normal(0, 50, size=(10, 1))
y_emperical_center = y_emperical - y_emperical.mean()
x_emperical_center = x_emperical - x_emperical.mean(axis=0)

```

- ▼ (b) (10 points) Empirically demonstrate or mathematically show that the least squares solution has zero training error when $p > n$.

```
[125] import random

np.random.seed(89)
x_emperical = np.random.normal(random.random()*100, random.random()*1000, size=14)
for i in range(14):
    x_emperical = np.column_stack((x_emperical, np.random.normal(random.random())))
beta_true2 = np.random.randint(5, 10, size=(16, 1))
x_emperical = sm.add_constant(x_emperical)
y_emperical = x_emperical @ beta_true2 + np.random.normal(0, 50, size=(10, 1))
y_emperical_center = y_emperical - y_emperical.mean()
x_emperical_center = x_emperical - x_emperical.mean(axis=0)

model3 = sm.OLS(y_emperical_center, x_emperical_center)
result3 = model3.fit()
y_emperical_pred = x_emperical_center @ result3.params.reshape(-1, 1)
mse = metric.mean_squared_error(y_emperical_pred, y_emperical_center)
print(f"""
With x.shape as n={x_emperical.shape[0]} and p={x_emperical.shape[1]}
There are many feasible beta_hats, one beta is \n{np.round(result3.params.reshape(-1, 1), 3)}
Under circumstance of p > n, MSE={mse:.3f}
""")
```

- ▼ (c) (30 points) Empirically demonstrate or mathematically show properties of methods with correlated features.

- i. Empirically demonstrate or mathematically show that the least squares estimate has high variance for correlated features.
- ii. Empirically demonstrate or mathematically show that ridge regression groups highly correlated features for sufficiently large λ .
- iii. Empirically demonstrate or mathematically show that lasso regression tends to select only one out of a group of features that are highly correlated.

```
[126] betas = np.zeros((6, 1000))
for i in range(1000):
    x_emperical = np.random.normal(30, 180, size=(1000, 4))
    x_emperical = np.column_stack((x_emperical, x_emperical[:, 3] + np.random.normal(0, 30)))
    x_emperical_constant = sm.add_constant(x_emperical)
    beta_true = np.arange(1, 6).reshape(-1, 1)
    y_emperical = x_emperical @ beta_true + np.random.normal(0, 100, size=(1000, 1))

    model_test = sm.OLS(y_emperical, x_emperical_constant)
    betas[:, i] = model_test.fit().params
center_betas = betas - betas.mean(axis=1).reshape(-1, 1)
cov_betas = (center_betas @ center_betas.T) / 1000
diag = np.diag((1 / np.diag(cov_betas)) ** 0.5)
correlation_matrix = diag @ cov_betas @ diag
result4 = pd.DataFrame(correlation_matrix,
                       columns=[f"beta{i}" for i in range(6)],
                       index=[f"beta{i}" for i in range(6)])
value = result4.loc["beta5", "beta4"]
print(f"""
(i): x.shape={x_emperical_constant.shape}
Feature 5 is created by adding a normal(0,3) perturbation to feature 4.
(result4.to_string())
Cor(beta4, beta5) = {value:.6f}
Therefore, high variance for correlated features.
""")
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge

x_emperical_stand = StandardScaler().fit_transform(x_emperical)
y_emperical_center = y_emperical - y_emperical.mean()
model_test = Ridge(alpha=10, fit_intercept=False)
model_test.fit(x_emperical_stand, y_emperical_center)
betas = model_test.coef_
results = pd.DataFrame(betas, columns=["value"], index=[f"beta{i}" for i in range(5)])
print(f"""
(ii): x.shape={x_emperical_stand.shape}
Feature 5 is created by adding a normal(0,3) perturbation to feature 4.
{results.to_string()}
Therefore, ridge would group high correlative features with close betas.
""")

(ii): x.shape=(1000, 5)
Feature 5 is created by adding a normal(0,3) perturbation to feature 4.
      value
beta1  178.006782
beta2  358.254017
beta3  561.512765
beta4  822.498310
beta5  816.541029
Therefore, ridge would group high correlative features with close betas.
```

```
[139] from sklearn.linear_model import Lasso

lam = 0.9
model_test = Lasso(alpha=lam, fit_intercept=False)
model_test.fit(x_emperical_stand, y_emperical_center)
result = model_test.coef_.reshape(-1, 1)
print(f"""
(iii): x.shape={x_emperical_stand.shape}
Feature 5 is created by adding a normal(0,3) perturbation to feature 4.
If lambda big enough (Choosed lambda={lam}), lasso would select only one
out of a group of features that are highly correlated.
{pd.DataFrame(result, columns=["value"], index=[f"beta{i}" for i in range(1, 6)])}
""")

→ (iii): x.shape=(1000, 5)
Feature 5 is created by adding a normal(0,3) perturbation to feature 4.
If lambda big enough (Choosed lambda=0.9), lasso would select only one
out of a group of features that are highly correlated.
      value
beta1  179.120907
beta2  361.076014
beta3  566.677581
beta4  1646.023194
beta5   0.525146
```