HW 1

Manasa Maganti, Varsha Manju Jayakumar

Due: 11:59 PM CT, September 13

Total points: 75

Your homework should be written in a Python notebook and then exported to a PDF. You may work in groups of two if you wish. Only one student per team needs to submit the assignment on Gradescope. But be sure to include name and UT EID for both students.

Also, please make sure your code runs and the graphics (and anything else) are displayed in your notebook and PDF before submitting. (%matplotlib inline)

Question 1: AWS SageMaker (10 pts):

Read the article titled "10 reasons why Amazon SageMaker is great for machine learning" on Amazon SageMaker and answer the following questions:

1. In two paragraphs and in your own words, highlight some key benefits that a platform like Amazon SageMaker offers for the development of a machine learning (ML) application. (5 pts)
2. Use a large language model (LLM) of your choice to generate an alternative response to the above question. Be sure to include the name of the LLM used and the specific (sequence of) prompt(s) you provided. (3 pts)
3. Compare your explanation with the output from the LLM, and summarize your observations regarding the differences between the two answers. (2 pts)

Answer

Sub-ques 1:

Amazon SageMaker has fully managed services that enables machine learning-based models to be enabled into your applications quickly and easily by making sure that it is flexible, and it is available and accessible always. It also maps real-world use case problems (e.g.: - spam email filtering, Anomaly detection, Topic modeling etc.) with machine learning-based solutions and thereby provide a wide range of algorithms and frameworks. Amazon SageMaker can also be easily integrated with other AWS services like Amazon DynamoDB, Amazon Kinesis etc. thereby supporting our machine learning needs. Its Notebook Instances are fully managed Jupyter notebooks and it helps reduce machine training costs and also time and effort spent on environment setup. Creating a notebook instance, training a model, and deploying the model to production is done with a single click and it helps to focus more on solving the problem than worrying about infrastructure.

The AutoML capabilities of SageMaker helps with building, training and fine-tuning of the best machine learning models for our data type. The several built-in security features keep our data and models secure. Also, provisioned instances need not be paid for as SageMaker automatically scales resources based on the needs of the workload. It also provides monitoring and debugging tools namely CloudWatch logs and SageMaker debugger that supports major machine learning frameworks. Lastly, Amazon SageMaker has a very strong developer and data scientist community and support resources like AWS support, Amazon SageMaker developer forum and so on that aid in the process of developing customized machine learning models.

Sub-ques 2:

Amazon SageMaker offers several benefits that streamline the development of machine learning (ML) applications. One of the key advantages is its fully managed environment, which reduces the complexity of managing underlying infrastructure, such as processing power and data handling. This allows developers to focus on building and training models instead of dealing with the operational aspects of running ML applications. SageMaker supports a wide variety of ML frameworks like TensorFlow, PyTorch, and MXNet, providing flexibility to developers and data scientists to use the tools that best fit their project needs. Additionally, SageMaker offers AutoML capabilities, which can automate model training and optimization, making it easier to create high-quality models without deep expertise in algorithm tuning.

Another important benefit is SageMaker's integration with other AWS services, such as Amazon S3 for data storage and EC2 for compute resources, enabling the creation of robust, end-to-end ML workflows. It also features advanced tools for monitoring and debugging, such as the SageMaker Debugger, which helps detect anomalies during training and scales resources automatically based on workload demands. This

scalability ensures efficient resource usage, lowering costs by provisioning only the necessary resources at any given time. SageMaker's built-in security features, like encryption and role-based access control, make it a secure and compliant platform for enterprises handling sensitive data

LLM Used: ChatGPT

Prompt: "Read the article titled '10 reasons why Amazon SageMaker is great for machine learning' on Amazon SageMaker and in two paragraphs highlight some key benefits that a platform like Amazon SageMaker offers for the development of a machine learning (ML) application."

Sub-ques 3:

The LLM response is more concise and technical, focusing on Amazon SageMaker's core features like infrastructure management, AutoML capabilities, and integration with AWS services such as S3 and EC2. Our explanation, on the other hand, provides additional detail and examples, highlighting real-world applications and specific AWS tools like DynamoDB and Kinesis. It also emphasizes usability (like Jupyter notebooks and single-click deployment) and mentions the developer community and support, making it more relatable and user-focused. Overall, our response gives a more practical, hands-on perspective, while the LLM is more feature-focused.

Question 2: Google Flu Trends (10 pts)

The article "The Parable of Google Flu: Traps in Big Data Analysis" (kept in Canvas --> Modules --> Resources) describes a high-profile (and embarrassingly failed) project done by Google, highlighting the phenomena of data drift and the importance of transparency, among other key issues that an ML project can face.

Read this article and answer the following questions

(i) Briefly describe two important causes of "data drift" in the flu prediction problem that are mentioned in the article (5 pts)

(ii) The last section highlights the importance of transparency. Express in your words, why is transparency important for building data science and AI projects? (You can check this article as a helpful reference: https://hbr.org/2022/06/building-transparency-into-ai-projects or you can look for other sources and cite them in your answer) (5 pts)

Answer

1.

- Big Data Hubris: Over time, the way people search for health information online changes. For example, instead of typing "flu symptoms," they might search for specific symptoms like "fever" or "cough." These shifts in search behavior can throw off predictions because the model is looking for patterns in old search terms that people may not use as much anymore.
- Algorithm Dynamics: Google constantly tweaks its search engine to improve results, like suggesting related searches or providing medical info directly. These updates can change what people end up searching for, which in turn affects the data being used to predict flu trends.

2. Transparency is vital for AI and data science projects because it helps build trust and ensures that everyone involved understands how decisions are made. When companies are open about how data is collected, how models are trained, and how AI systems work, they reduce the risk of mistakes or misuse. Transparency also allows for accountability, as everyone can see who is responsible for each part of the process. Ultimately, it helps people feel respected and informed, especially when AI directly impacts their lives.

Question 3: Maximum Likelihood Estimate (10 pts)

Consider the following probability density function (pdf) of a random variable X that takes value $x \in [0, \infty)$

$$f(x; b) = \frac{x}{Kb^2} \exp(-\frac{x}{b})$$

where b is the parameter of the density function, and K is a suitable normalizing constant. The (unscaled) pdfs with $b = 0.1, 0.5, 1$ are plotted below to help you visualize how this family of pdfs looks like.

Suppose we observe 6 values obtained by sampling i.i.d. from the pdf described above:

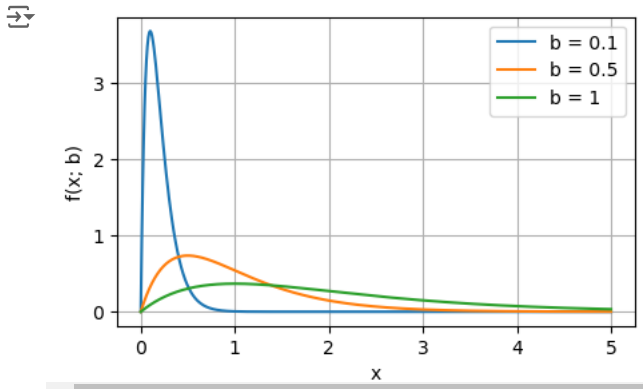
$$\{x_1, x_2, x_3, x_4, x_5, x_6\} = \{0.1, 0.5, 1, 0.4, 2, 1.4\}$$

Answer the following questions:

1. What is the log-likelihood of observing a set of 6 samples drawn i.i.d from the pdf mentioned above (give an algebraic expression that is valid for any set of 6 samples)? (5 points)

2. What is the maximum likelihood estimate for the parameter b? (5 points)

```
import numpy as np
import matplotlib.pyplot as plt

def pdf(x, b):
    return (x / (b**2)) * np.exp(-x / b) # K = 1

b_values = [0.1, 0.5, 1]

x = np.linspace(0, 5, 500)

plt.figure(figsize=(5, 3))
for b in b_values:
    plt.plot(x, pdf(x, b), label=f'b = {b}')

plt.xlabel('x')
plt.ylabel('f(x; b)')
plt.legend()
plt.grid(True)
plt.show()
```



Answer

1. Log-Likelihood Expression

Given the probability density function (pdf):

$$f(x;b) = \frac{x}{Kb^2}\exp\left(-\frac{x}{b}\right)$$

we need to find the log-likelihood of observing a set of 6 samples $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ drawn i.i.d. from this distribution.

1. **Likelihood Function:**

   For a single observation x_i , the likelihood function is:

$$f(x_i;b) = \frac{x_i}{Kb^2}\exp\left(-\frac{x_i}{b}\right)$$

For 6 i.i.d. samples, the likelihood function (L(b)) is:

$$L(b) = \prod_{i=1}^{6} f(x_i;b) = \prod_{i=1}^{6} \left(\frac{x_i}{Kb^2}\exp\left(-\frac{x_i}{b}\right)\right)$$

$$L(b) = \left(\frac{1}{Kb^2}\right)^6 \prod_{i=1}^{6} x_i \exp\left(-\sum_{i=1}^{6}\frac{x_i}{b}\right)$$

2. **Log-Likelihood Function:**

   Taking the logarithm to get the log-likelihood function $\ell(b)$:

$$\ell(b) = \log L(b)$$

$$\ell(b) = \log\left(\frac{1}{K^6 b^{12}} \prod_{i=1}^{6} x_i \exp\left(-\frac{\sum_{i=1}^{6} x_i}{b}\right)\right)$$

$$\ell(b) = \log\left(\frac{1}{K^6 b^{12}}\right) + \log\left(\prod_{i=1}^{6} x_i\right) - \frac{\sum_{i=1}^{6} x_i}{b}$$

$$\ell(b) = -6\log K - 12\log b + \sum_{i=1}^{6} \log x_i - \frac{\sum_{i=1}^{6} x_i}{b}$$

2. Maximum Likelihood Estimate for (b)

To find the maximum likelihood estimate (MLE) for (b), we maximize the log-likelihood function $\ell(b)$.

1. **Differentiate the Log-Likelihood Function:**

$$\frac{\partial \ell(b)}{\partial b} = -12\frac{1}{b} + \frac{\sum_{i=1}^{6} x_i}{b^2}$$

2. **Set the Derivative to Zero:**

$$-12\frac{1}{b} + \frac{\sum_{i=1}^{6} x_i}{b^2} = 0$$

$$\frac{\sum_{i=1}^{6} x_i}{b^2} = 12\frac{1}{b}$$

$$\sum_{i=1}^{6} x_i = 12b$$

$$b = \frac{\sum_{i=1}^{6} x_i}{12}$$

3. **Calculate the MLE:**

   For the sample values ({0.1, 0.5, 1, 0.4, 2, 1.4}):

$$\sum_{i=1}^{6} x_i = 0.1 + 0.5 + 1 + 0.4 + 2 + 1.4 = 5.4$$

$$b_{\text{MLE}} = \frac{5.4}{12} = 0.45$$

Summary:

1. The log-likelihood function is:

$$\ell(b) = -6\log K - 12\log b + \sum_{i=1}^{6} \log x_i - \frac{\sum_{i=1}^{6} x_i}{b}$$

2. The maximum likelihood estimate for (b) is: $b_{\text{MLE}} = 0.45$

Question 4: Linear Regression (10 pts)

1. What is the difference between R-square and adjusted R square and why is it desirable to use the adjusted value? (4 pts)
2. Overfitting usually happens in complex models. Linear Regression is a fairly simple model. Could overfitting happen in Linear Regression? If so, please explain the scenario in which it could happen and how we can tackle it. (6 pts)

Answer

1. Difference between R2 and Adjusted R2:

- R2 and Adjusted R2 are metrics to evaluate the goodness of fit for any regression model.

  R2:

- R2 measures the proportion of the variance in the dependent variable based on the data from independent variables. It generally takes on a value between 0 and 1 and is independent of scale of dependent variable.
- It is defined as R2 = 1-RSS/TSS (RSS - Residual Sum of Squares, TSS - Total sum of squares)
- The explainability of variance in dependent variable increases as it approaches 1. 0 means independent variables do not explain variance in dependent variable and 1 means that they explain all the variance.
- Higher R2 means a better fit of the model to the data.

  Adjusted R2:

- Adjusted R2 is another approach for selecting among set of models that contain different number of variables which accounts for the fact that R2 increases as moe predictors are added to the model.
- It is defined as adj R2 = 1 - ((RSS/(n-d-1))/(TSS/ (n-1))) (n - no of observations, d - number of independet variables)
- Adjusted R2 pays a price for the inclusion of unnecessary variables in the model.

  Why use Adjusted R2 over R2?

- R2 value doesn't decrease with more predictors although they do not improve the model accuracy. However, adjusted R2 value decreases if the additional predictors create more noise in the model performance. Hence, its better to use adjusted R2 over R2.

2. Could overfitting happen in Linear Regression?

Yes, overfitting can happen in Linear Regression because of the following reasons:

- When the number of observations in the data is not significantly larger than the number of variables, it can lead to a lot of variance in the model which eventually leads to overfitting.
- The model essentially captures the noise in the training data rather than interpreting the underlying relationship.
- Additionally, if there are multiple features in the model that have strong correlation and are linearly dependent, it leads to collinearity. If there is strong correlation in data, the model may become overly sensitive to small changes, leading to overfitting.
- Lastly, in the case where there are significantly high number of outliers, the model may excessively fit to those extreme values leading to overfitting.

If so, please explain the scenario in which it could happen and how we can tackle it -

We can tackle overfitting in Linear Regression in following ways:

- Shrinkage Methods: Through shrinkage methods, we fit all the p predictors using a technique which constrains or regularizes the coefficient estimates towards zero. In doing so, we are improving the fit by reducing their variance.
- Cross validation: Through cross validation, we are splitting the data into k-folds. We then try to evaluate how the mdel generalizes to new data by training the model on subsets of data and testing on remaining parts. Since we train on each fold, the model tends to generalize the underlying pattern eliminating overfitting.
- Handling outliers: We can remove or treat the outliers in our data through imputation techniques. This will restrain the model from training on extreme data points and hence avoid overfitting
- Dimensionality reduction: By using techniques like Principal Component Analysis (PCA) or manually selecting features based on domain knowledge, we can reduce the dimensionality of the input data and remove redundant or irrelevant features, which can help to prevent overfitting.

Question 5: Ridge/ Lasso Regression (35 pts)

This is a programming question. Please read through each subpart of this question carefully. You are required to add lines of code as specified in the code cells. Please carefully read through the comments in the code cells to identify what code is to be written, where it is to be written and how many lines of code are required. Code is to be added between the **## START CODE** ## and **## END CODE ##** comments and in place of the keyword **None**. In certain cases, the number of lines of code that are to be written will be specified. For example, **## START CODE ## (1 line of code)** specifies that only 1 line of code is to be added between the ## START CODE ## and ## END CODE ## comments. In case there is no information on the required number of lines, you are allowed to add any number of lines of code.

The following question covers a dataset for house cost and linear models in python. The categorical variables and rows with missing variables are removed to make it easier to run the models.

NOTE

- Only use the following code block if you are using Google Colab. If you are using Jupyter Notebook, please ignore this code block. You can directly upload the file to your Jupyter Notebook file systems.
- It will prompt you to select a local file. Click on “Choose Files” then select and upload the file. Wait for the file to be 100% uploaded. You should see the name of the file once Colab has uploaded it.

```
from google.colab import files
uploaded = files.upload()
```

Imports required

```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn.metrics import r2_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
pd.options.mode.chained_assignment = None

df = pd.read_csv('house_cost_data.csv')
X = df.drop(['house_cost'],axis=1)
Y = df['house_cost']


# Show you all the columns in this file
df.columns
```

```
Index(['house_cost', 'num_of_beds', 'num_of_baths', 'living_area', 'sqft_lot',
       'floors', 'condition', 'grade', 'sqft_above', 'sqft_basement',
       'built_year', 'renovation_year', 'latitude', 'longitude',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

```
# Show you the first 5 rows in this file
df.head()
```

	house_cost	num_of_beds	num_of_baths	living_area	sqft_lot	floors	condition	grade	sqft_above	sqft_basement	built_year	renovat
0	221900.0	3	1.00	1180	5650	1.0	3	7	1180	0	1955	
1	538000.0	3	2.25	2570	7242	2.0	3	7	2170	400	1951	
2	180000.0	2	1.00	770	10000	1.0	3	6	770	0	1933	
3	604000.0	4	3.00	1960	5000	1.0	5	7	1050	910	1965	
4	510000.0	3	2.00	1680	8080	1.0	3	8	1680	0	1987	

Part-1: (2 pts)

Split the data into a training set (75% of data) and a test set (25% of data), using the train_test_split function with random_state = 50.

```
##  START CODE  ## (1 line of code)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=50)
##  END CODE    ##
```

Scale the data (not including target) so that each of the independent variables would have zero mean and unit variance. You can use the sklearn.preprocessing.scale function for this.

```
from sklearn.preprocessing import scale
##  START CODE  ## (2 lines of code)
Xscaled_train = scale(X_train)
Xscaled_test = scale(X_test)
##  END CODE    ##
```

```
y_train = y_train.to_numpy()
y_test = y_test.to_numpy()
```

Print the first 5 rows of the training set after scaling

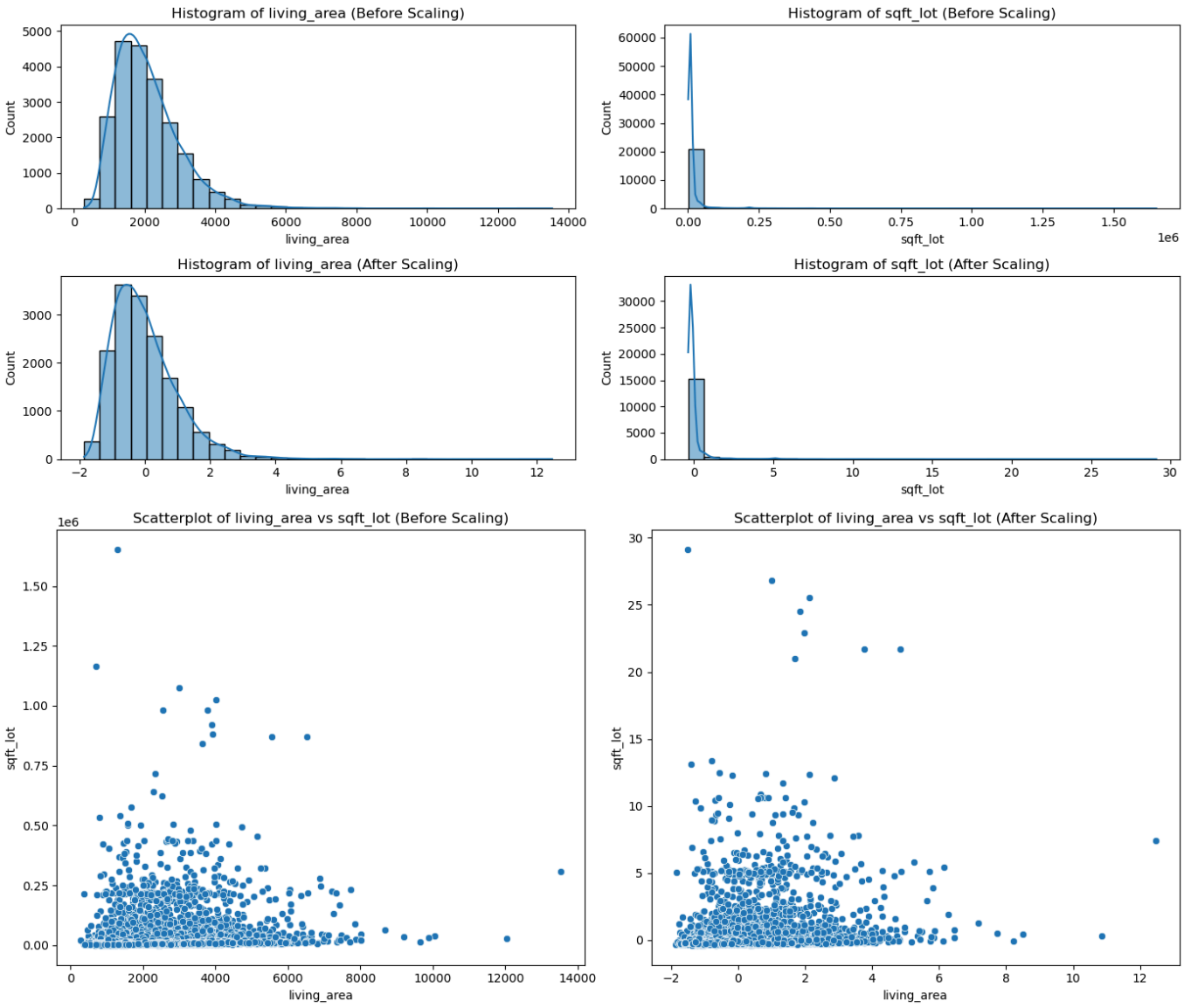
```
##  START CODE  ##
data = pd.DataFrame(Xscaled_train, columns=X_train.columns).head()
data
##  END CODE    ##
```

	num_of_beds	num_of_baths	living_area	sqft_lot	floors	condition	grade	sqft_above	sqft_basement	built_year	renovation_ye
0	-1.461288	-1.450106	-1.362933	-0.225207	-0.920140	0.906949	-1.412493	-1.161718	-0.654608	-0.582459	-0.2078
1	-1.461288	-1.450106	-1.373813	0.028226	-0.920140	0.906949	-2.263327	-1.173788	-0.654608	-1.636973	-0.2078
2	-0.400894	-1.450106	-1.080044	-0.116904	-0.920140	0.906949	-1.412493	-0.847900	-0.654608	-0.718525	-0.2078
3	-1.461288	0.491153	-0.808035	-0.352215	0.936751	-0.631107	0.289175	-1.028949	0.247788	1.254438	-0.2078
4	-0.400894	-0.479476	-0.209616	-0.280688	-0.920140	-0.631107	-0.561659	-0.606502	0.698986	0.097874	-0.2078

Select any two variables. See how their histograms and scatterplots compare before and after scaling.

```
##  START CODE  ##

import matplotlib.pyplot as plt
import seaborn as sns


variable1 = 'living_area'
variable2 = 'sqft_lot'

data_before = df[[variable1, variable2]]
data_after = pd.DataFrame(Xscaled_train, columns=X_train.columns)[[variable1, variable2]]


plt.figure(figsize=(14, 6))

# Histograms before scaling
plt.subplot(2, 2, 1)
sns.histplot(data_before[variable1], bins=30, kde=True)
plt.title(f'Histogram of {variable1} (Before Scaling)')

plt.subplot(2, 2, 2)
sns.histplot(data_before[variable2], bins=30, kde=True)
plt.title(f'Histogram of {variable2} (Before Scaling)')

# Histograms after scaling
plt.subplot(2, 2, 3)
sns.histplot(data_after[variable1], bins=30, kde=True)
plt.title(f'Histogram of {variable1} (After Scaling)')

plt.subplot(2, 2, 4)
sns.histplot(data_after[variable2], bins=30, kde=True)
plt.title(f'Histogram of {variable2} (After Scaling)')

plt.tight_layout()
plt.show()


plt.figure(figsize=(14, 6))

# Scatterplots before scaling
plt.subplot(1, 2, 1)
sns.scatterplot(x=data_before[variable1], y=data_before[variable2])
plt.title(f'Scatterplot of {variable1} vs {variable2} (Before Scaling)')

# Scatterplots after scaling
plt.subplot(1, 2, 2)
sns.scatterplot(x=data_after[variable1], y=data_after[variable2])
plt.title(f'Scatterplot of {variable1} vs {variable2} (After Scaling)')
```

```
    plt.tight_layout()
    plt.show()

## END CODE ##
```



Part-2: (5 pts)

Use `sklearn.linear_model.Lasso` and `sklearn.linear_model.Ridge` classes to do a 5-fold cross validation using sklearn's `KFold`. For the sweep of the regularization parameter, we will look at a grid of values ranging from $\alpha = 10^{-6}$ to $\alpha = 10^{6}$.In Python, you can consider this

range of values as follows: `alpha = 10**numpy.linspace(-6, 6, 100)` so that you can generate 100 uniform values between -6 to 6 as power series.

Fit the 2 regression models (Lasso and Ridge) with scaled data and report the best chosen α based on cross validation as well as the corresponding scoring metric. The cross validation should happen on your training data using MSE as the scoring metric.

```
# Define number of folds
##  START CODE  ## (1 line of code)
n_folds = 5
##  END CODE  ##



# Create KFold from sklearn
from sklearn.model_selection import KFold
##  START CODE  ## (1 line of code)
k_fold = KFold(n_splits=n_folds, shuffle=True, random_state=50)
##  END CODE    ##



#Define the alphas as defined in the question
##  START CODE  ## (1 line of code)
alphas = 10 ** np.linspace(-6, 6, 100)
##  END CODE    ##

lasso_avg_mse = {}
ridge_avg_mse = {}


#For each value of alpha and each fold compute the mean square error
#---------NOTE: This might take a while to run, so please be patient---------------#
#---------NOTE: There will be some convergence warning for smaller alphas, but you can ignore it---------------#
from sklearn.linear_model import Lasso, Ridge
from sklearn.metrics import mean_squared_error, r2_score

for alpha in alphas:

  #Instantiate a lasso model with the current alpha
  ##  START CODE  ## (1 line of code)
    lasso_model = Lasso(alpha=alpha)
  ##  END CODE    ##

    avg_mse = 0
    for k, (train, test) in enumerate(k_fold.split(Xscaled_train, y_train)):

    #Fit the scaled training data to the lasso model
    ## START CODE ## (1 line of code)
        lasso_model.fit(Xscaled_train[train], y_train[train])
    ## END CODE ##

    #Calculate the average mean sqaured error
    ##  START CODE  ## (1 line of code)
        avg_mse += mean_squared_error(y_train[test], lasso_model.predict(Xscaled_train[test]))
    ##  END CODE    ##

  # Take the average mean squared error as metric
    lasso_avg_mse[alpha] = avg_mse / n_folds
```

```
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
```

```python
# Find the best value for alpha with minimum mean squared error
##  START CODE  ## (1 line of code)
best_alpha_lasso = min(lasso_avg_mse, key=lasso_avg_mse.get)
##  END CODE    ##

print("Best lasso alpha: {}".format(best_alpha_lasso))
```

```
Best lasso alpha: 533.6699231206302
```

```python
#For each value of alpha and each fold compute the mean square error
for alpha in alphas:

  #Instantiate a ridge model with the current alpha
  ##  START CODE  ## (1 line of code)
    ridge_model = Ridge(alpha=alpha)
  ##  END CODE    ##

    avg_mse = 0

    for k, (train, test) in enumerate(k_fold.split(Xscaled_train, y_train)):

    #Fit the scaled training data to the ridge model
    ## START CODE ## (1 line of code)
        ridge_model.fit(Xscaled_train[train], y_train[train])
    ## END CODE ##

    #Calculate the average mean sqaured error
    ##  START CODE  ## (1 line of code)
        avg_mse += mean_squared_error(y_train[test], ridge_model.predict(Xscaled_train[test]))
    ##  END CODE    ##

  # Take the average mean squared error as metric
    ridge_avg_mse[alpha] = avg_mse / n_folds


# Find the best value for alpha with minimum mean squared error
##  START CODE  ## (1 line of code)
best_alpha_ridge = min(ridge_avg_mse, key=ridge_avg_mse.get)
##  END CODE    ##
```

```
print("Best Ridge alpha: {}".format(best_alpha_ridge))
```

```
Best Ridge alpha: 57.223676593502205
```

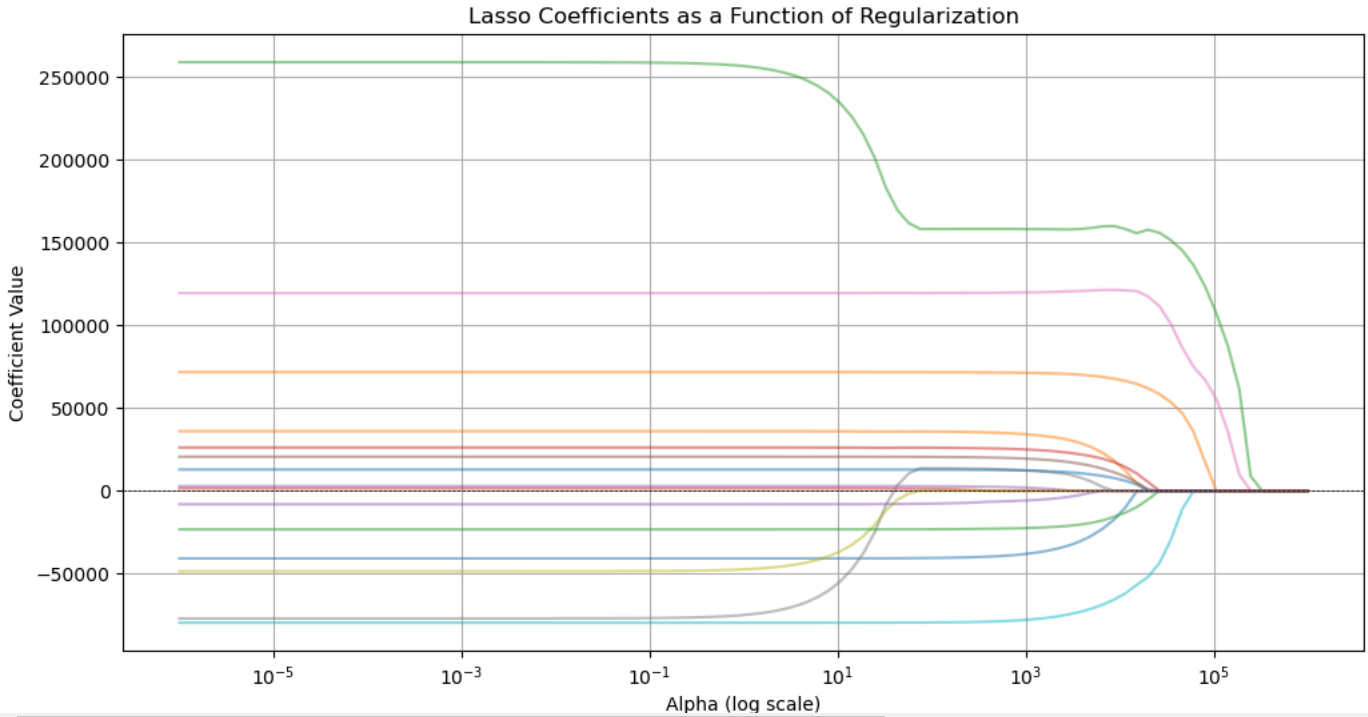
Part-3: (7 pts)

Run ridge and lasso regression for all of the α specified above (on training data), and plot the coefficients learned for each of them - there should be one plot each for lasso and ridge, so a total of two plots; different features' weights of each model should be on the same plot with different colors.

```
# Lasso Regression

alphas = 10**np.linspace(6,-6,100)

lasso = linear_model.Lasso(alpha=alpha)
coefs = []

for a in alphas:
  #Specify current alpha as parameter for the lasso model
  ## START CODE ## (1 line of code)
    lasso = Lasso(alpha=a)
  ## END CODE ##

  #Fit the training data to the lasso model
  ## START CODE ## (1 line of code)
    lasso.fit(Xscaled_train, y_train)
  ## END CODE ##

  #Store learned coefficients in the coef variable
  ## START CODE ## (1 line of code)
    coefs.append(lasso.coef_)
  ## END CODE ##
```

```
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
```

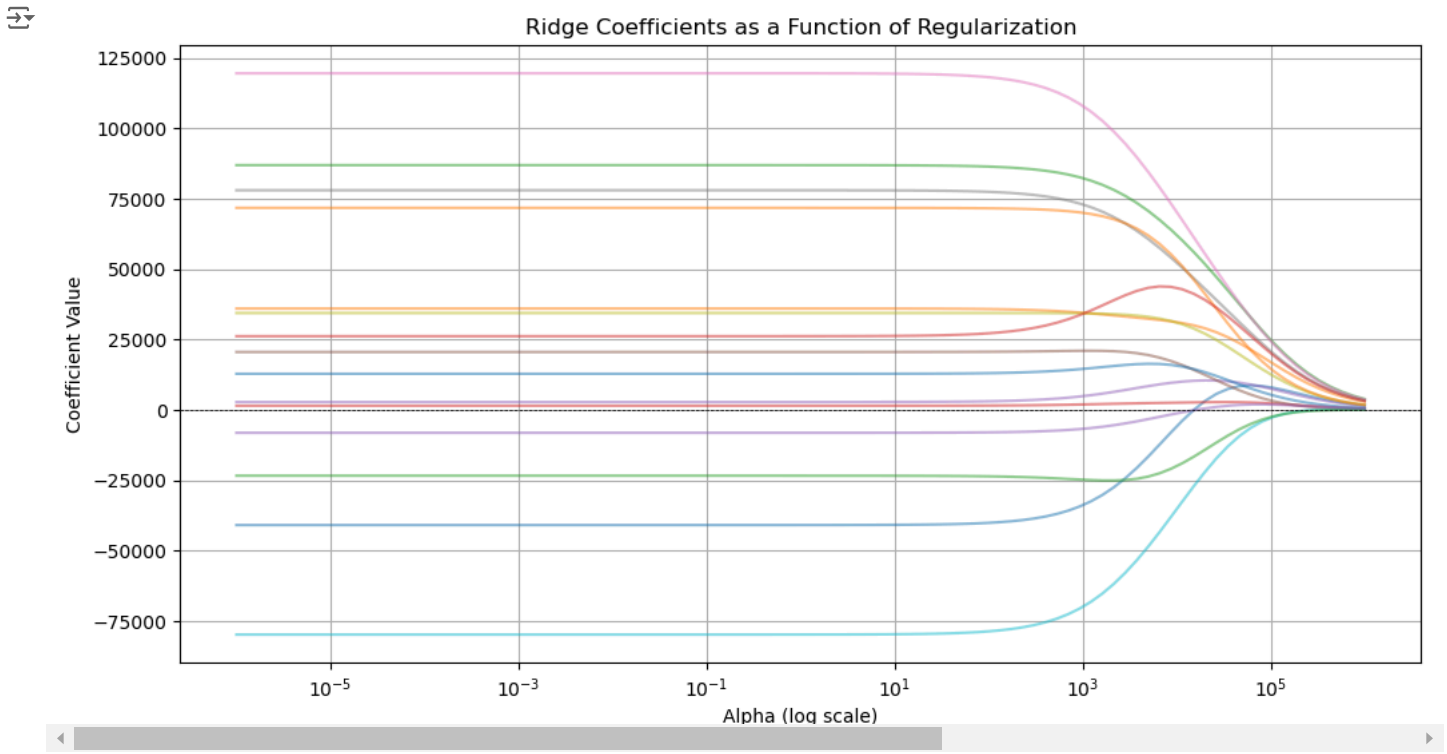
```
# Write the code to make the plot for coefficients learned from lasso
## START CODE ##
plt.figure(figsize=(12, 6))
for coef in np.array(coefs).T:  # Transpose for features
    plt.plot(alphas, coef, alpha=0.5)
plt.xscale('log')
plt.title('Lasso Coefficients as a Function of Regularization')
plt.xlabel('Alpha (log scale)')
plt.ylabel('Coefficient Value')
plt.grid(True)
plt.axhline(0, color='black', lw=0.5, ls='--')
plt.show()
## END CODE ##
```



```
# Ridge Regression

alphas = 10**np.linspace(6,-6,100)

ridge = linear_model.Ridge(alpha=alpha)
coefs = []

for a in alphas:
  #Specify current alpha as parameter for the lasso model
  ## START CODE ## (1 line of code)
    ridge = Ridge(alpha=a)
  ## END CODE ##

  #Fit the training data to the lasso model
  ## START CODE ## (1 line of code)
    ridge.fit(Xscaled_train, y_train)
  ## END CODE ##
```

```
  #Store learned coefficients in the coef variable
  ## START CODE ## (1 line of code)
    coefs.append(ridge.coef_)
  ## END CODE ##
```

```
# Write the code to make the plot for coefficients learned from ridge
## START CODE ##
plt.figure(figsize=(12, 6))
for coef in np.array(coefs).T:  # Transpose for features
    plt.plot(alphas, coef, alpha=0.5)
plt.xscale('log')
plt.title('Ridge Coefficients as a Function of Regularization')
plt.xlabel('Alpha (log scale)')
plt.ylabel('Coefficient Value')
plt.grid(True)
plt.axhline(0, color='black', lw=0.5, ls='--')
plt.show()
## END CODE ##
```



What do you qualitatively observe when the value of the regularization parameter changes?

When the regularization parameter(alpha) increases in Lasso regression, the model becomes more selective, pushing some feature coefficients to zero, effectively ignoring certain features. This helps simplify the model by focusing only on the most important features. As alpha decreases, more features are retained, and the coefficients increase, making the model more flexible. However, with too little regularization, the model might overfit, capturing noise in the data.

In Ridge regression, increasing alpha shrinks all the feature coefficients toward zero, but unlike Lasso, it doesn't eliminate any features completely. This makes the model simpler and less sensitive to the specific training data, improving generalization. As alpha decreases, the model becomes more complex, with larger coefficients, potentially leading to overfitting if regularization is too weak.

Answer

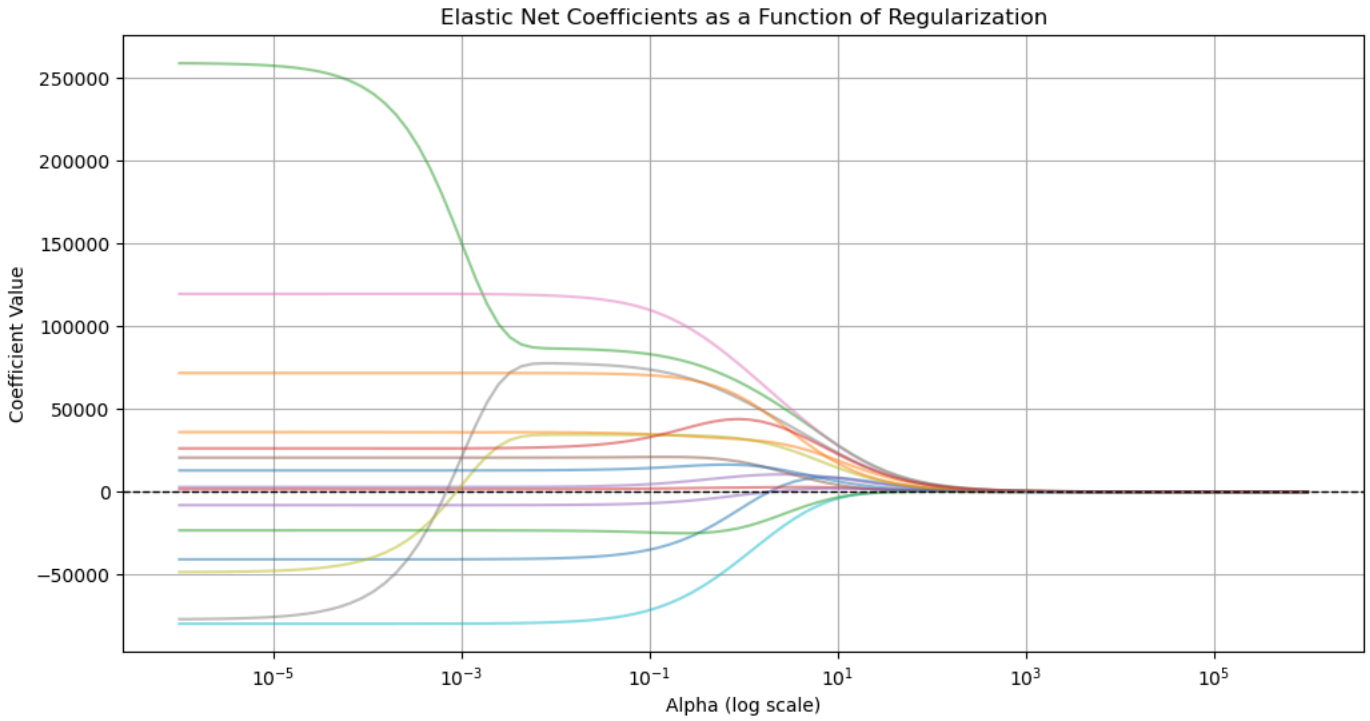
Part-4: (5 pts)

Similarly, use `sklearn.linear_model.ElasticNet` to do linear regression with different α values, and plot the coefficients learned for each of them

```
# Ridge Regression
from sklearn.linear_model import ElasticNet

alphas = 10**np.linspace(6,-6,100)

ElastNet = linear_model.ElasticNet(alpha=alpha)
coefs = []

for a in alphas:
  #Specify current alpha as parameter for the lasso model
  ## START CODE ## (1 line of code)
    elastic_net = ElasticNet(alpha=a)
  ## END CODE ##

  #Fit the training data to the lasso model
  ## START CODE ## (1 line of code)
    elastic_net.fit(Xscaled_train, y_train)
  ## END CODE ##

  #Store learned coefficients in the coef variable
  ## START CODE ## (1 line of code)
    coefs.append(elastic_net.coef_)
  ## END CODE ##
```

```
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
  model = cd_fast.enet_coordinate_descent(
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not con
```

```
# Write the code to make the plot for coefficients learned from ElasticNet
## START CODE ##
plt.figure(figsize=(12, 6))
for coef in np.array(coefs).T:  # Transpose for features
    plt.plot(alphas, coef, alpha=0.5)
plt.xscale('log')
plt.title('Elastic Net Coefficients as a Function of Regularization')
plt.xlabel('Alpha (log scale)')
plt.ylabel('Coefficient Value')
plt.grid(True)
plt.axhline(0, color='black', lw=1, ls='--')
plt.show()
## END CODE ##
```



Observe the plot, then explain the pros and cons of ridge, lasso and Elastic Net models.

Ridge Regression:

Pros:

- Reduces model complexity by shrinking coefficients, helping to prevent overfitting.
- Works well when many small/medium-sized effects are present (when all features are relevant).
- Handles multicollinearity well since it includes all features in the model.

Cons:

- Does not perform feature selection (all coefficients are shrunk but remain non-zero).
- May include irrelevant features, which can lead to less interpretable models.

Lasso Regression:

Pros:

- Performs automatic feature selection by driving some coefficients to zero, which simplifies the model.
- Can lead to more interpretable models by excluding less important features.
- Useful when you suspect that only a few features are important.

Cons:

- Can be sensitive to small changes in the data, particularly when features are highly correlated.
- May struggle when many features contribute similarly to the outcome, as it tends to select one and ignore others.

Elastic Net:

Pros:

- Combines the strengths of both Ridge and Lasso by including both L1 and L2 penalties.
- Performs well in situations where the number of features is greater than the number of observations or when several features are correlated.
- Can lead to better predictive performance than using Lasso or Ridge alone.

Cons:

- More complex due to the introduction of two hyperparameters (alpha and the mixing parameter).
- Requires careful tuning of parameters to achieve optimal performance.

Elastic Net is often preferred in practice when dealing with high-dimensional datasets or when there is a belief that many features might be relevant, as it balances the trade-offs of Ridge and Lasso regression.

Answer

Part-5: (10 pts)

Run the following three regression models with MSE loss on the training data:

(a) linear regression without regularization

(b) linear regression with ridge regularization

(c) linear regression with lasso regularization

For part (b) and (c), use only the best regularization parameters. Report the MSE and R2 on the test data for each of the models.

```
## START CODE ##
from sklearn.linear_model import LinearRegression
# Linear Regression without regularization

linear_model = LinearRegression()
linear_model.fit(Xscaled_train, y_train)
y_pred_linear = linear_model.predict(Xscaled_test)

# Calculate MSE and R2 for Linear Regression
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)



# Linear Regression with Ridge regularization using the best alpha
ridge_model = Ridge(alpha=best_alpha_ridge)
ridge_model.fit(Xscaled_train, y_train)
y_pred_ridge = ridge_model.predict(Xscaled_test)

# Calculate MSE and R2 for Ridge Regression
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)


# Linear Regression with Lasso regularization using the best alpha
lasso_model = Lasso(alpha=best_alpha_lasso)
lasso_model.fit(Xscaled_train, y_train)
y_pred_lasso = lasso_model.predict(Xscaled_test)

# Calculate MSE and R2 for Lasso Regression
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)


print(f"Linear Regression MSE: {mse_linear:.4f}, R²: {r2_linear:.4f}")
print(f"Ridge Regression MSE: {mse_ridge:.4f}, R²: {r2_ridge:.4f}")
print(f"Lasso Regression MSE: {mse_lasso:.4f}, R²: {r2_lasso:.4f}")
## END CODE ##
```

```
Linear Regression MSE: 48827883407.8462, R²: 0.6513
Ridge Regression MSE: 48828862496.5081, R²: 0.6512
```

```
Lasso Regression MSE: 48866301676.5811, R²: 0.6510
```

Part-6: (3 pts)

Train the 3 models and report metrics with the original data without scaling

```
# (a) Linear Regression without regularization on original data
linear_model_original = LinearRegression()
linear_model_original.fit(X_train, y_train)
y_pred_linear_original = linear_model_original.predict(X_test)

mse_linear_original = mean_squared_error(y_test, y_pred_linear_original)
r2_linear_original = r2_score(y_test, y_pred_linear_original)

# (b) Linear Regression with Ridge regularization on original data using the best alpha
ridge_model_original = Ridge(alpha=best_alpha_ridge)
ridge_model_original.fit(X_train, y_train)
y_pred_ridge_original = ridge_model_original.predict(X_test)

mse_ridge_original = mean_squared_error(y_test, y_pred_ridge_original)
r2_ridge_original = r2_score(y_test, y_pred_ridge_original)

# (c) Linear Regression with Lasso regularization on original data using the best alpha
lasso_model_original = Lasso(alpha=best_alpha_lasso)
lasso_model_original.fit(X_train, y_train)
y_pred_lasso_original = lasso_model_original.predict(X_test)

mse_lasso_original = mean_squared_error(y_test, y_pred_lasso_original)
r2_lasso_original = r2_score(y_test, y_pred_lasso_original)

## START CODE ##
print(f"Linear Regression MSE (Original Data): {mse_linear_original:.4f}, R²: {r2_linear_original:.4f}")
print(f"Ridge Regression MSE (Original Data): {mse_ridge_original:.4f}, R²: {r2_ridge_original:.4f}")
print(f"Lasso Regression MSE (Original Data): {mse_lasso_original:.4f}, R²: {r2_lasso_original:.4f}")
## END CODE ##
```

```
Linear Regression MSE (Original Data): 48852207456.8430, R²: 0.6511
Ridge Regression MSE (Original Data): 49078487402.6339, R²: 0.6495
Lasso Regression MSE (Original Data): 48934697060.7969, R²: 0.6505
C:\Users\manas\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning: Objective did not conver
  model = cd_fast.enet_coordinate_descent(
```

Part-7: (3 pts)

Why did we have to scale the data in ridge and lasso regression?

Answer

In Lasso regression, the L1 penalty term drives some coefficients to zero. In Ridge regression, the L2 penalty term shrinks the coefficients. If the features are not on the same scale, the regularization will disproportionately affect features with larger ranges. Scaling the data ensures that regularization impacts all features uniformly.