ECE 580 Homework #2

Libo Zhang (lz200)

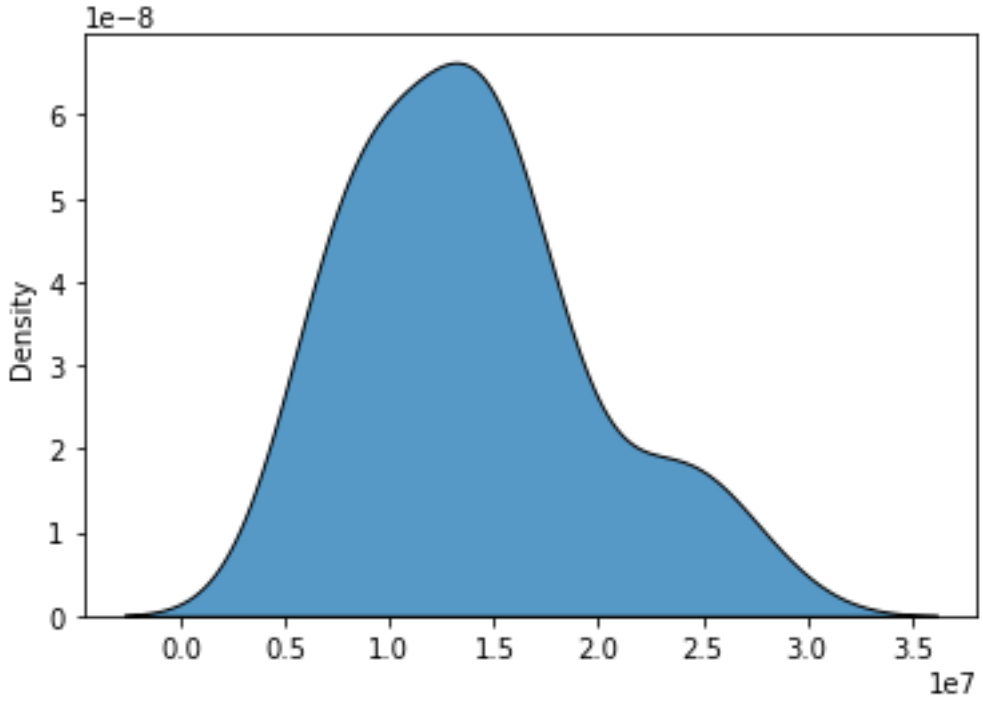
Cross Validation Section

Question 1 (a) Solution:

Sub-question (i) – The equation of my proposed model #1 is shown below.

$$\widehat{price} = \widehat{w_0} + \widehat{w_1}(engine\_size)$$

Sub-question (ii) – The kernel density estimate plot for the mean square error (MSE) is shown below.



$$E\{MSE\} = 13906846.8970 \approx 1.39 \times 10^7$$

Given the kernel density estimate plot for MSE and the value of E{MSE}, I believe E{MSE} is a representative summary statistic (a good approximation) for MSE, because the plot displays a trend of uniform distribution, and the value of E{MSE} is very close to the mean value of this uniform distribution.

Sub-question (iii) – The value of $variance$ for this model is shown below.

$$variance = 2240542.6289 \approx 2.24 \times 10^6$$

Sub-question (iv) – The value of $(bias)^2 + noise\ variance$ for this model is shown below.

$$(bias)^2 + noise\ variance = E\{MSE\} - variance = 11666304.2681 \approx 1.17 \times 10^7$$

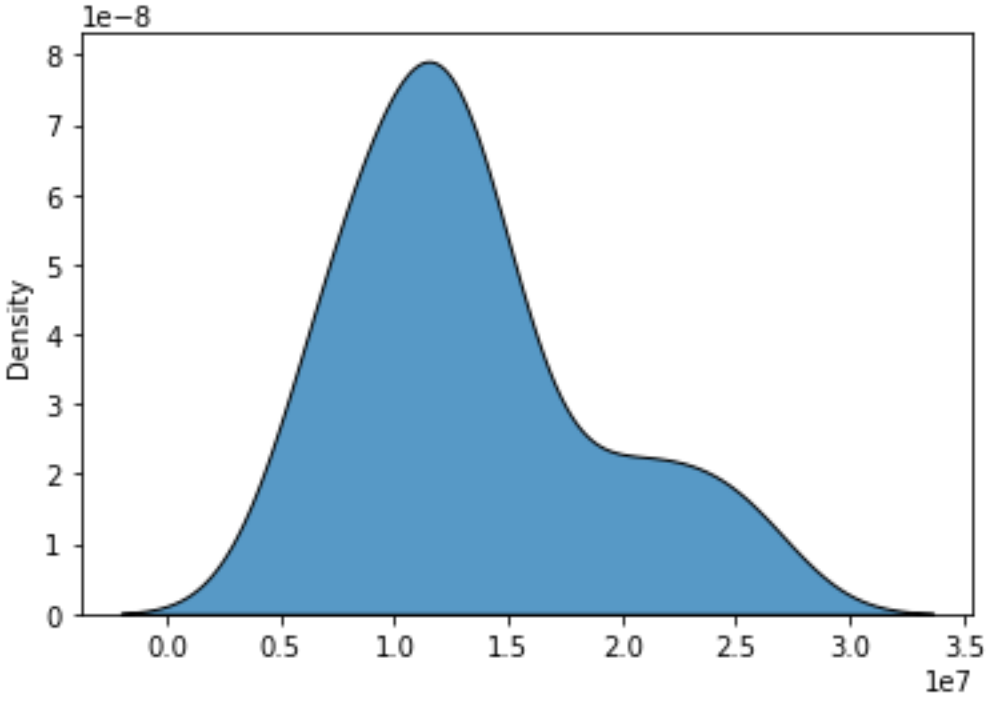
Sub-question (v) – For this model, the $variance$ value is very small while the $(bias)^2 + noise\ variance$ value is very large. Therefore, I think my proposed model #1 appears to have greater consistency and greater systematic error across data sets.

Question 1 (b) Solution:

Sub-question (i) – The equation of my proposed model #2 is shown below.

$$\widehat{price} = \widehat{w_0} + \widehat{w_1}(engine\_size) + \widehat{w_2}(horsepower)$$

Sub-question (ii) – The kernel density estimate plot for the mean square error (MSE) is shown below.



$$E\{MSE\} = 13331364.1512 \approx 1.33 \times 10^7$$

Given the kernel density estimate plot for MSE and the value of E{MSE}, I believe E{MSE} is a representative summary statistic (a good approximation) for MSE, because the plot displays a trend of uniform distribution, and the value of E{MSE} is very close to the mean value of this uniform distribution.

Sub-question (iii) – The value of $variance$ for this model is shown below.

$$variance = 3633621.0361 \approx 3.63 \times 10^6$$

Sub-question (iv) – The value of $(bias)^2 + noise\ variance$ for this model is shown below.

$$(bias)^2 + noise\ variance = E\{MSE\} - variance = 9697743.1151 \approx 9.70 \times 10^6$$

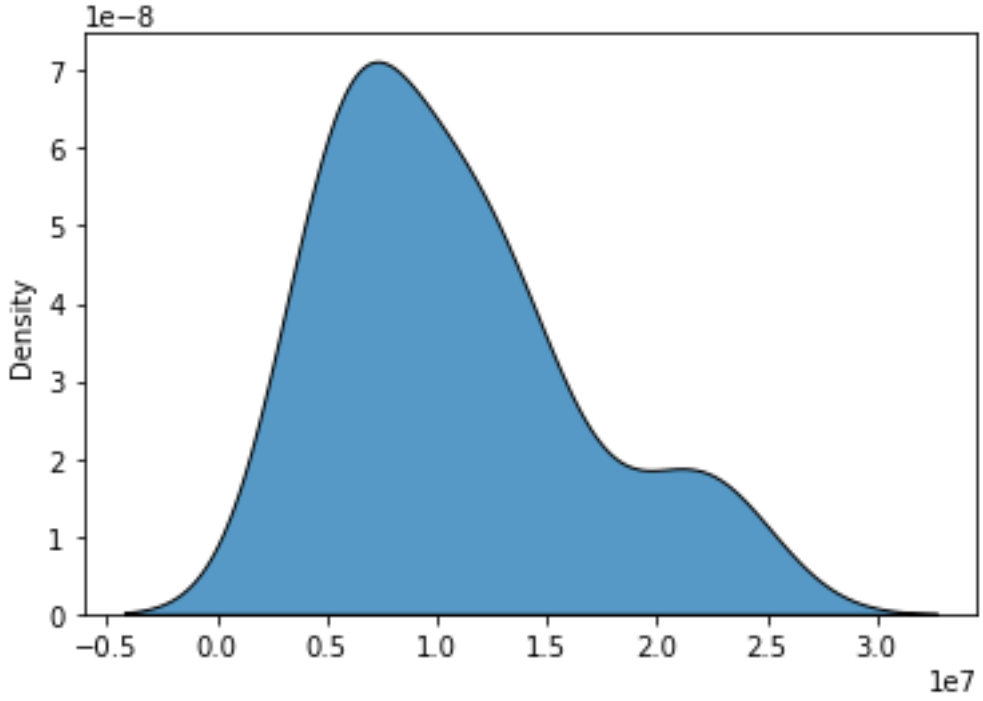
Sub-question (v) – For this model, the $variance$ value is still small while the $(bias)^2 + noise\ variance$ value is still large. Therefore, I think my proposed model #2 appears to have greater consistency and greater systematic error across data sets. However, compared with model #1, I think model #2 does not have such strong extent of the so-called "greater consistency and systematic error", because the difference between bias and variance is smaller, or the distance between bias and variance is closer, which means the model is more balanced in terms of consistency, variability, and systematic error across data sets.

Question 1 (c) Solution:

Sub-question (i) – The equation of my proposed model #3 is shown below.

$$\widehat{price} = \widehat{w_0} + \widehat{w_1}(engine\_size) + \widehat{w_2}(horsepower) + \widehat{w_3}(width) + \widehat{w_4}(width)^2$$

Sub-question (ii) – The kernel density estimate plot for the mean square error (MSE) is shown below.



$$E\{MSE\} = 10787609.9029 \approx 1.08 \times 10^7$$

Given the kernel density estimate plot for MSE and the value of E{MSE}, I believe E{MSE} is a representative summary statistic (a good approximation) for MSE, because the plot displays a trend of uniform distribution, and the value of E{MSE} is very close to the mean value of this uniform distribution.

Sub-question (iii) – The value of $variance$ for this model is shown below.

$$variance = 4183449.9993 \approx 4.18 \times 10^6$$

Sub-question (iv) – The value of $(bias)^2 + noise\ variance$ for this model is shown below.

$$(bias)^2 + noise\ variance = E\{MSE\} - variance = 6604159.9036 \approx 6.60 \times 10^6$$

Sub-question (v) – For this model, the $variance$ value is relatively small while the $(bias)^2 + noise\ variance$ value is relatively large. Although this still indicates that my proposed model #3 appears to have greater consistency and greater systematic error across data sets, I think the so-called "greater" extent is a lot weaker compared with model #1 and model #2, because for model #3 the $variance$ and the $(bias)^2 + noise\ variance$ values are very close to each other. Therefore, I think my proposed model #3 is the most balanced considering the bias-variance trade-off, in terms of model consistency, variability, and systematic error across data sets.

Question 1 (d) Solution:

A summary table for my three proposed models is shown below.

Model Number	$E\{MSE\}$ $(\times 10^6)$	$variance$ $(\times 10^6)$	$[(bias)^2 + noise\ variance]$ $(\times 10^6)$
Model #1	13.94	2.24	11.70
Model #2	13.33	3.63	9.70
Model #3	10.78	4.18	6.60

According to the above table, the order of my 3 models from least complex (high bias, low variance) to most complex (low bias, high variance) can be concluded below ("<" denotes model complexity).

$$model\ \#1 < model\ \#2 < model\ \#3$$

This conclusion also demonstrates my expectation when I initially designed the 3 models, as model #1 uses 1 feature, model #2 uses 2 features, and model #3 uses 3 features. Therefore, without guessing, actually, the model complexity ordering, or the bias-variance trade-off performance of my 3 models should have already been very explicit, and what I am doing here is to prove my expectation and make it more solid.

Question 1 (e) Solution:

I would select model #3 as my ideal model. This is because model #3 has the lowest value of E{MSE} and the lowest value of $(bias)^2 + noise\ variance$, which means that model #3 has the best regression performance and the least systematic error across data sets. Although model #3 has the highest value of $variance$, the variance value of model #3 does not increase significantly compared to model #1 and model #2. According to bias-variance trade-off, it is very reasonable for me to select model #3 as my ideal model.

Here, my conclusion of selecting model #3 is similar to the conclusion I came to without cross-validation in Homework #1. In Homework #1, I also selected model #3 as my ideal model because model #3 had the highest unadjusted R^2 score and the highest adjusted R^2 score, which could tell me that all 3 features in model #3 are meaningful (can provide additional information), and model #3 has the best regression performance.

Question 1 (f) Solution:

Before formally submitting the complete print-out of my code, I want to show that I indeed structure and organize my code according to the requirements to make it modular and extensible.

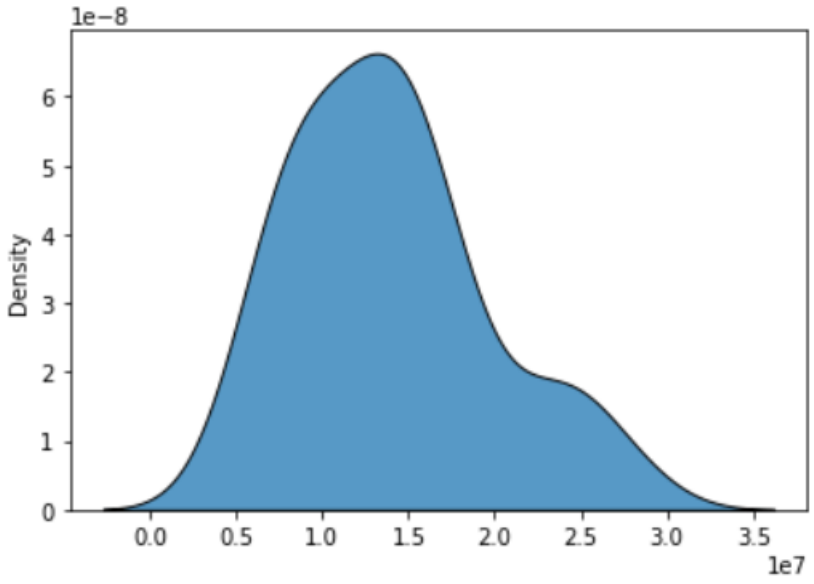
I write a function named “model_summary” integrating all of my previous self-defined functions. This function could generate all of the regression results in Homework 2 using a single code block that accepts:

1. A given data set (X and y for my regression model)
2. The desired norm
3. The desired regularization
4. The desired cross-validation parameters

An example of “a single code block” is shown below.

```
# Cross Validation Section
# Model 1
# Total Model 1
model_summary(X_one, y, alpha = 0, norm = "L2", R = 3, K = 10)
```

```
Model E{MSE}                        = 13906846.8970
Model variance                      = 2240542.6289
Model [(bias)^2 + noise_variance]   = 11666304.2681

Plot a kernel density estimate for MSE of this model below
```



Please continue to the next page for the complete print-out of my code.

Question 1 (f):

Reference the Packages/Toolboxes

I use numpy to process intermediate arraies and matrices.
I use pandas to read and process the original dataset.
I use seaborn to draw the kernel density estimate plot for MSE.
I use RepeatedKFold from sklearn to implement 3-10-folds cross-validation.
I use Ridge from sklearn to implement L2 regression.
I use Lasso from sklearn to implement L1 regression.
(Lasso is not used in HW2, but it is useful as another desired norm option when structuring and organizing my code)
I use mean_squared_error from sklearn to calculate the mean square error (MSE).

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
```

Beginning of Data Preprocessing

Note: Data Preprocessing section has already been completed in Homework 1.
The purpose of this section is to help extract X_one, X_two, X_three, and y for my 3 proposed models.
X and y will be regarded as "a given data set" when I structure and organize my code in Homework 2.

[2]:
```
filename = "imports-85.data"
dataframe = pd.read_csv(filename, header = None)
```

[3]:
```
print("The number of data points in the original data set is: " + str(len(dataframe)))
```

```
The number of data points in the original data set is: 205
```

[4]:
```
# All 26 attributes in the data frame.
all_attributes = ["symboling", "normalized-losses", "make",
                  "fuel-type", "aspiration", "num-of-doors",
                  "body-style", "drive-wheels", "engine-location",
                  "wheel-base", "length", "width",
                  "height", "curb-weight", "engine-type",
                  "num-of-cylinders", "engine-size", "fuel-system",
                  "bore", "stroke", "compression-ratio",
                  "horsepower", "peak-rpm", "city-mpg",
                  "highway-mpg", "price"]
# Restrict ourselves to the 13 continuous predictor variables.
features = ["wheel-base", "length", "width",
            "height", "curb-weight", "engine-size",
            "bore", "stroke", "compression-ratio",
            "horsepower", "peak-rpm", "city-mpg", "highway-mpg"]
# Define the target variable.
target = "price"
# Assign each attribute to each column of the data frame correspondingly.
# This is because the original data set does not contain such information.
dataframe.columns = all_attributes
```

Question 1 (f):

[5]:
```
# Step 1 - Remove the (non-continuous) features that are not of interest.
old_len = len(dataframe)
for attribute in all_attributes :
    if (attribute not in features) and (attribute != target) :
        dataframe.drop(attribute, axis = 1, inplace = True)
    else :
        continue
new_len = len(dataframe)
print("Number of data points in my data frame before Step 1: " + str(old_len))
print("Number of data points in my data frame after Step 1: " + str(new_len))
print("Number of data points removed at Step 1: " + str(old_len - new_len))
```

```
Number of data points in my data frame before Step 1: 205
Number of data points in my data frame after Step 1: 205
Number of data points removed at Step 1: 0
```

[6]:
```
# Step 2 - Remove any data points for which the target variable (price) is unknown.
old_len = len(dataframe)
dataframe = dataframe[dataframe[target] != "?"]
new_len = len(dataframe)
print("Number of data points in my data frame before Step 2: " + str(old_len))
print("Number of data points in my data frame after Step 2: " + str(new_len))
print("Number of data points removed at Step 2: " + str(old_len - new_len))
```

```
Number of data points in my data frame before Step 2: 205
Number of data points in my data frame after Step 2: 201
Number of data points removed at Step 2: 4
```

[7]:
```
# Step 3 - Remove any data points for which the continuous predictor variables are unknown.
old_len = len(dataframe)
for feature in features :
    dataframe = dataframe[dataframe[feature] != "?"]
new_len = len(dataframe)
print("Number of data points in my data frame before Step 3: " + str(old_len))
print("Number of data points in my data frame after Step 3: " + str(new_len))
print("Number of data points removed at Step 3: " + str(old_len - new_len))
```

```
Number of data points in my data frame before Step 3: 201
Number of data points in my data frame after Step 3: 195
Number of data points removed at Step 3: 6
```

[8]:
```
# Do some housekeeping in this block.
df_numpy = dataframe.to_numpy()
for j in range(df_numpy.shape[1]) :
    for i in range(df_numpy.shape[0]) :
        df_numpy[i, j] = float(df_numpy[i, j])
features_matrix = df_numpy[:, 0:13]
target_vector = df_numpy[:, 13]
df_numpy = df_numpy.astype(np.float64)
print(features_matrix.shape)
print(target_vector.shape)
```

```
(195, 13)
(195,)
```

Question 1 (f):

[9]:
```
# Do some housekeeping in this block.
wheelbase  = np.copy(df_numpy[:, 0])
length     = np.copy(df_numpy[:, 1])
width      = np.copy(df_numpy[:, 2])
height     = np.copy(df_numpy[:, 3])
curbweight = np.copy(df_numpy[:, 4])
enginesize = np.copy(df_numpy[:, 5])
bore       = np.copy(df_numpy[:, 6])
stroke     = np.copy(df_numpy[:, 7])
comratio   = np.copy(df_numpy[:, 8])
horsepower = np.copy(df_numpy[:, 9])
peakrpm    = np.copy(df_numpy[:, 10])
citympg    = np.copy(df_numpy[:, 11])
highwaympg = np.copy(df_numpy[:, 12])
```

[10]:
```
# Do some housekeeping in this block.
X_one = enginesize.reshape(-1, 1)

X_two = np.column_stack((enginesize, horsepower))

X_three = np.column_stack((enginesize, horsepower,
                           width, width * width))

y = df_numpy[:, -1]

print(X_one.shape)
print(X_two.shape)
print(X_three.shape)
print(y.shape)
```

```
(195, 1)
(195, 2)
(195, 4)
(195,)
```

Ending of Data Preprocessing

Please continue to the next page for Homework 2 codes.

Question 1 (f):

Beginning of Codes for Homework 2

[11]:
```
def build_model (X, y,            # a given dataset
                 alpha,           # the desired regularization
                 norm = "L2",     # the desired norm
                 R = 3, K = 10): # the desired cross-validation parameters
    rkf = RepeatedKFold(n_splits = K, n_repeats = R,
                        # Pass an int for reproducible output
                        # across multiple function calls
                        random_state = 5)
    count = 0
    total_samples = R * K
    coef_list = np.zeros((total_samples, 1 + X.shape[1]))
    mse_list = np.zeros(total_samples)
    for train_index, test_index in rkf.split(X) :
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        if norm == "L2" :
            model = Ridge(alpha = alpha)
        if norm == "L1" :
            model = Lasso(alpha = alpha)
        model.fit(X_train, y_train)
        coef_list[count, 0] = model.intercept_
        coef_list[count, 1:] = model.coef_
        preds = model.predict(X_test)
        mse_list[count] = mean_squared_error(y_true = y_test,
                                             y_pred = preds)
        count = count + 1
    return coef_list, mse_list
```

[12]:
```
def E_mse (mse_list) :
    mse_sum = 0
    for mse in mse_list :
        mse_sum = mse_sum + mse
    avg_mse = float(mse_sum) / len(mse_list)
    return avg_mse

def plot_kde (mse_list) :
    sns.kdeplot(data = mse_list, multiple = "stack")
    return None

def reconstruct_model (Xn, coef_list, l) :
    yl_Xn = 0
    for i in range(1 + len(Xn)) :
        if i == 0 :
            yl_Xn = yl_Xn + coef_list[l, i]
        else :
            yl_Xn = yl_Xn + coef_list[l, i] * Xn[i - 1]
    return yl_Xn
```

Question 1 (f):

```
def calculate_avgpred (Xn, coef_list) :
    L = len(coef_list)
    yn_sum = 0
    yn_avg = 0
    for l in range(L) :
        yn_sum = yn_sum + reconstruct_model(Xn, coef_list, l)
    yn_avg = float(yn_sum) / L
    return yn_avg
```

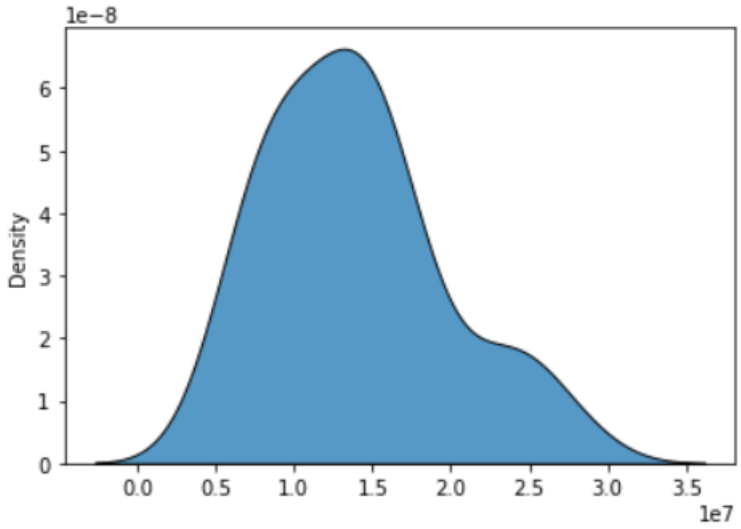
[13]:
```
def calculate_variance (X, coef_list) :
    N = X.shape[0]
    L = len(coef_list)
    sum_inside = 0
    avg_inside = 0
    sum_outside = 0
    avg_outside = 0   # variance = avg_outside
    for n in range(N) :
        Xn = X[n]
        yn_avg = calculate_avgpred(Xn, coef_list)
        for l in range(L) :
            subdiff = reconstruct_model(Xn, coef_list, l) - yn_avg
            sub_sqr = subdiff * subdiff
            sum_inside = sum_inside + sub_sqr
        avg_inside = float(sum_inside) / L
        sum_outside = sum_outside + avg_inside
    avg_outside = float(sum_outside) / N
    return avg_outside

def calculate_bias_noise (X, coef_list, mse_list) :
    # E{MSE} = variance + (bias)^2 + noise_variance
    # (bias)^2 + noise_variance = E{MSE} - variance
    result = E_mse(mse_list) - calculate_variance(X, coef_list)
    return result
```

[14]:
```
# The final "exam" for structuring and organizing my code

def model_summary (X, y,            # a given data set
                   alpha,           # the desired regularization
                   norm = "L2",     # the desired norm
                   R = 3, K = 10): # the desired cross-validation parameters

    coef_list, mse_list = build_model(X = X, y = y, alpha = alpha,
                                      norm = "L2",
                                      R = 3, K = 10)
    avg_mse = E_mse(mse_list)
    variance = calculate_variance(X, coef_list)
    bias_noise = calculate_bias_noise(X, coef_list, mse_list)
    print("Model E{MSE}                        = %.4f" % avg_mse)
    print("Model variance                      = %.4f" % variance)
    print("Model [(bias)^2 + noise_variance] = %.4f" % bias_noise)
    print(" ")
    print("Plot a kernel density estimate for MSE of this model below")
    plot_kde(mse_list)
    return None
```

Question 1 (f):

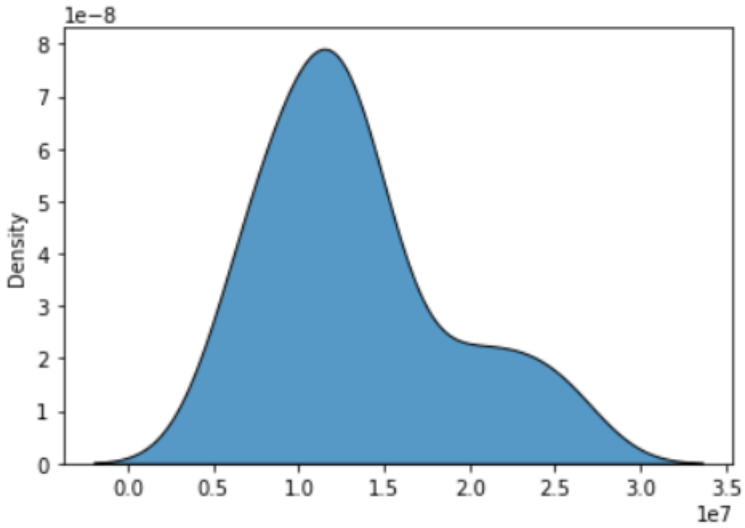
[15]:
```
# Cross Validation Section
# Model 1
# Total Model 1
model_summary(X_one, y, alpha = 0, norm = "L2", R = 3, K = 10)
```

```
Model E{MSE}                        = 13906846.8970
Model variance                      = 2240542.6289
Model [(bias)^2 + noise_variance]   = 11666304.2681

Plot a kernel density estimate for MSE of this model below
```



[16]:
```
# Cross Validation Section
# Model 2
# Total Model 2
model_summary(X_two, y, alpha = 0, norm = "L2", R = 3, K = 10)
```

```
Model E{MSE}                        = 13331364.1512
Model variance                      = 3633621.0361
Model [(bias)^2 + noise_variance]   = 9697743.1151

Plot a kernel density estimate for MSE of this model below
```

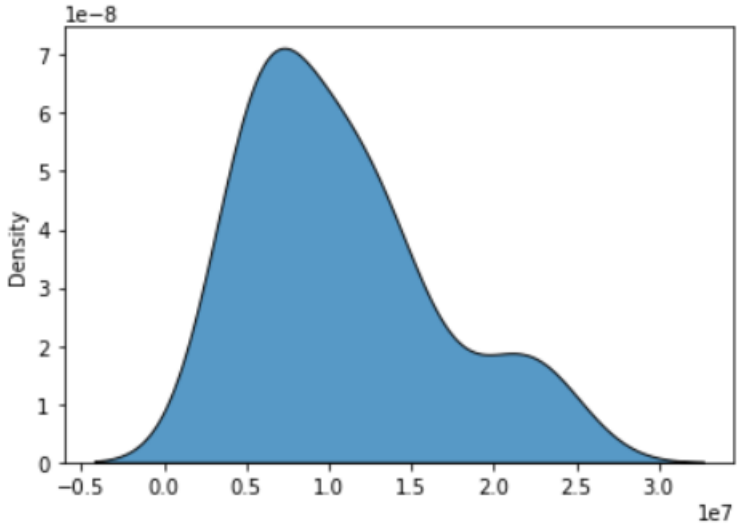


Question 1 (f):

[17]:
```
# Cross Validation Section
# Model 3
# Total Model 3
model_summary(X_three, y, alpha = 0, norm = "L2", R = 3, K = 10)
```

```
Model E{MSE}                        = 10787609.9029
Model variance                      = 4183449.9993
Model [(bias)^2 + noise_variance]   = 6604159.9036
```

Plot a kernel density estimate for MSE of this model below



Regularization Section

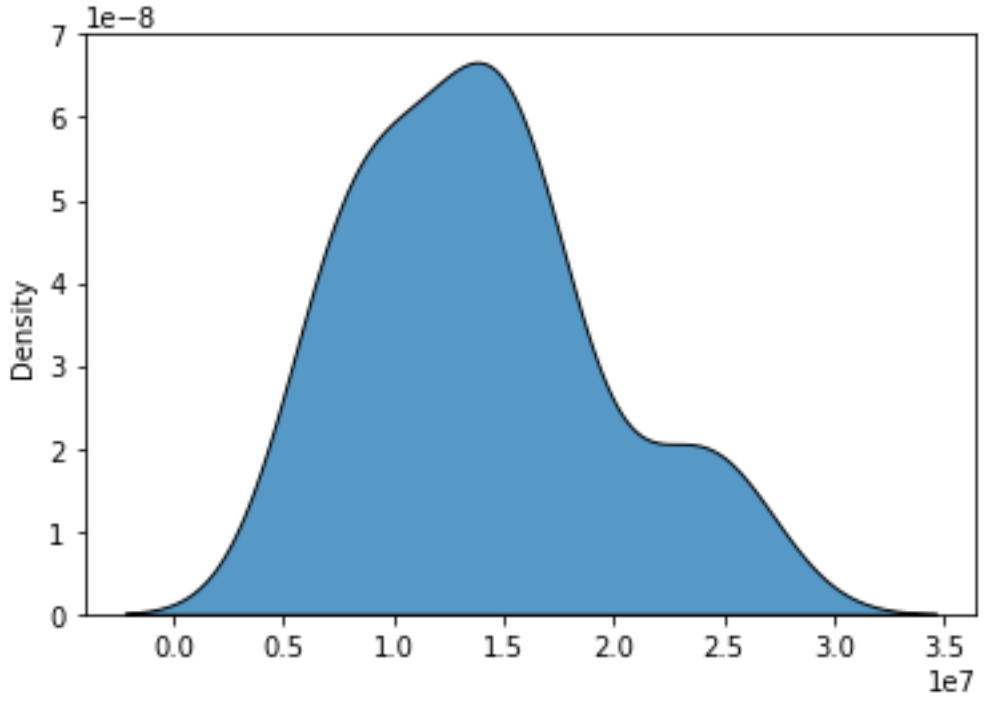
Question 2 (a) Solution:

Note: When choosing a weighting for the regularization term (denoted as α or alpha), my experiments show that a small alpha value does not affect my model #1 much, in terms of E{MSE}, variance, and bias. However, if the alpha value is too large, then my model #1's regression performance will degrade significantly. Therefore, considering such trade-off, I choose $\alpha = 10000 = 10^4$ for my proposed model #1.

Sub-question (i) – The equation of my proposed model #1 is shown below.

$$\widehat{price} = \widehat{w_0} + \widehat{w_1}(engine\_size)$$

Sub-question (ii) – The kernel density estimate plot for the mean square error (MSE) is shown below.



$$E\{MSE\} = 13932504.3158 \approx 1.39 \times 10^7$$

Given the kernel density estimate plot for MSE and the value of E{MSE}, I believe E{MSE} is a representative summary statistic (a good approximation) for MSE, because the plot displays a trend of uniform distribution, and the value of E{MSE} is very close to the mean value of this uniform distribution.

Sub-question (iii) – The value of $variance$ for this model is shown below.

$$variance = 2005667.0544 \approx 2.00 \times 10^6$$

Sub-question (iv) – The value of $(bias)^2 + noise\ variance$ is shown below.

$$(bias)^2 + noise\ variance = E\{MSE\} - variance = 11926837.2615 \approx 1.19 \times 10^7$$

Sub-question (v) – For this model, the $variance$ value is very small while the $(bias)^2 + noise\ variance$ value is very large. Therefore, I think this model appears to have greater consistency and greater systematic error across data sets.

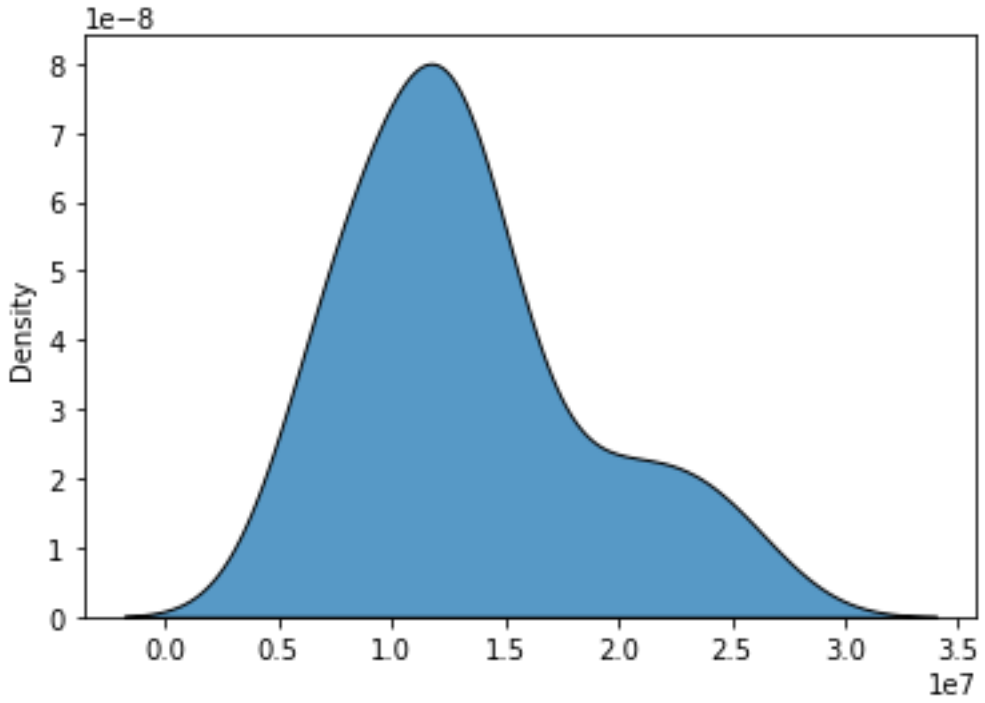
Question 2 (b) Solution:

Note: When choosing a weighting for the regularization term (denoted as α or alpha), my experiments show that a small alpha value does not affect my model #2 much, in terms of E{MSE}, variance, and bias. However, if the alpha value is too large, then my model #2's regression performance will degrade significantly. Therefore, considering such trade-off, I choose $\alpha = 10000 = 10^4$ for my proposed model #2.

Sub-question (i) – The equation of my proposed model #2 is shown below.

$$\widehat{price} = \widehat{w_0} + \widehat{w_1}(engine\_size) + \widehat{w_2}(horsepower)$$

Sub-question (ii) – The kernel density estimate plot for the mean square error (MSE) is shown below.



$$E\{MSE\} = 13345028.5078 \approx 1.33 \times 10^7$$

Given the kernel density estimate plot for MSE and the value of E{MSE}, I believe E{MSE} is a representative summary statistic (a good approximation) for MSE, because the plot displays a trend of uniform distribution, and the value of E{MSE} is very close to the mean value of this uniform distribution.

Sub-question (iii) – The value of $variance$ for this model is shown below.

$$variance = 3256561.7754 \approx 3.26 \times 10^6$$

Sub-question (iv) – The value of $(bias)^2 + noise\ variance$ for this model is shown below.

$$(bias)^2 + noise\ variance = E\{MSE\} - variance = 10088466.7324 \approx 1.01 \times 10^7$$

Sub-question (v) – For this model, the $variance$ value is still very small while the $(bias)^2 + noise\ variance$ value is still very large. Therefore, I think this model appears to have greater consistency and greater systematic error across data sets.

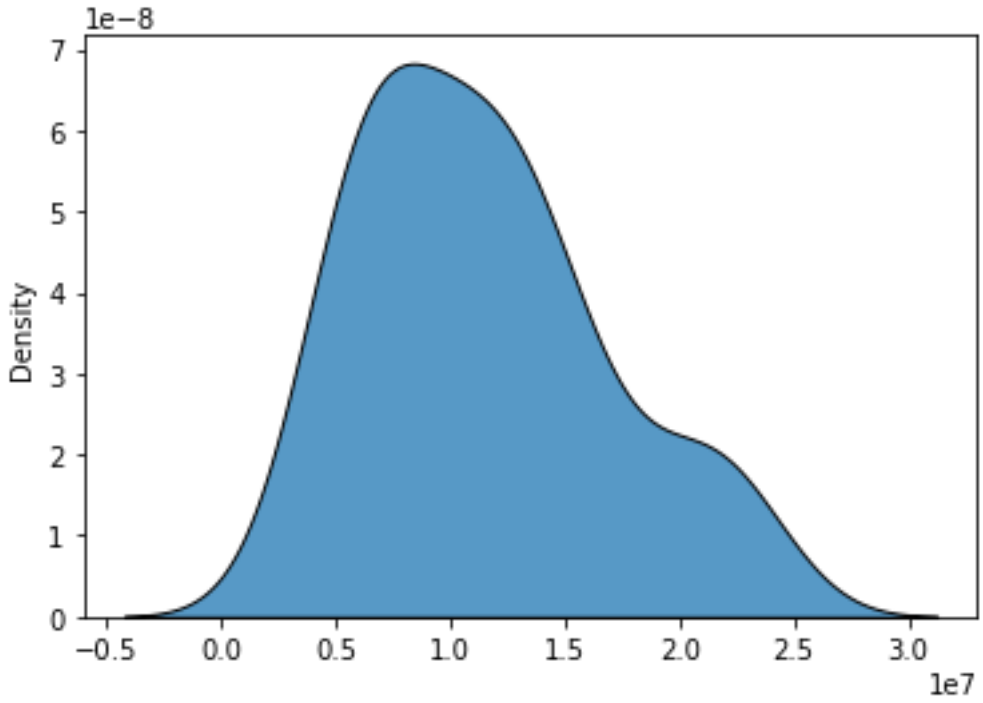
Question 2 (c) Solution:

Note: When choosing a weighting for the regularization term (denoted as α or alpha), my experiments show that a small alpha value could clearly affect my proposed model #3, such as increasing E{MSE} and bias, while decreasing variance. If the alpha value is too large, then my model #3's regression performance will degrade significantly. Therefore, I choose $\alpha = 1$ for my proposed model #3.

Sub-question (i) – The equation of my proposed model #3 is shown below.

$$\widehat{price} = \widehat{w_0} + \widehat{w_1}(engine\_size) + \widehat{w_2}(horsepower) + \widehat{w_3}(width) + \widehat{w_4}(width)^2$$

Sub-question (ii) – The kernel density estimate plot for the mean square error (MSE) is shown below.



$$E\{MSE\} = 11450997.9716 \approx 1.14 \times 10^7$$

Given the kernel density estimate plot for MSE and the value of E{MSE}, I believe E{MSE} is a representative summary statistic (a good approximation) for MSE, because the plot displays a trend of uniform distribution, and the value of E{MSE} is very close to the mean value of this uniform distribution.

Sub-question (iii) – The value of $variance$ for this model is shown below.

$$variance = 3829208.8159 \approx 3.83 \times 10^6$$

Sub-question (iv) – The value of $(bias)^2 + noise\ variance$ for this model is shown below.

$$(bias)^2 + noise\ variance = E\{MSE\} - variance = 7621789.1558 \approx 7.62 \times 10^6$$

Sub-question (v) – For this model, the $variance$ value is relatively small while the $(bias)^2 + noise\ variance$ value is relatively large. Although this still indicates that my proposed model #3 appears to have greater consistency and greater systematic error across data sets, I think such "greater" extent is much weaker compared with my model #1 and model #2, because for model #3 the variance and bias values are relatively close to each other. Therefore, I think model #3 is still the most balanced considering the bias-variance trade-off.

Question 2 (d) Solution:

A summary table for my 3 proposed models with/without regularization is shown below.

Model (# / Total #)	L_2 Regularization (α)	$E\{MSE\}$ ($\times 10^6$)	$variance$ ($\times 10^6$)	$[(bias)^2 + noise\ variance]$ ($\times 10^6$)
#1 (#1)	0	13.94	2.24	11.70
#1 (#4)	10000	13.93	2.00	11.93
#2 (#2)	0	13.33	3.63	9.70
#2 (#5)	10000	13.35	3.26	10.09
#3 (#3)	0	10.78	4.18	6.60
#3 (#6)	1	11.45	3.83	7.62

According to the above table, the order of my 3 models (considering both L2 regularization and cross-validation) from least complex (high bias, low variance) to most complex (low bias, high variance) can be concluded below ("<" denotes model complexity).

$$model\ \#1 < model\ \#2 < model\ \#3$$

According to the above table, we can see that after adding the appropriate L2 regularization term, the E{MSE} tends to be stable or to increase a little bit, the variance tends to decrease a bit, and the bias tends to increase a bit, but all changes are not significant. Therefore, we can also conclude that the results using L2 regularization (Ridge regression) are similar to the results using no regularization.

Question 2 (e) Solution:

I would select the proposed model #3 under no regularization as my ideal model. This model is also #3 among the Total # of models listed in the above summary table.

As for the reason, firstly, this model has the lowest E{MSE} and the lowest bias among all 6 models, which means that this model has the best regression performance and the least systematic error across data sets.

Secondly, although this model has the highest variance, its variance is not significantly higher than the other 5 models, and in fact, this model has the smallest difference (or the closest distance) between bias and variance, which means this model is the most balanced one among all 6 models, according to bias-variance trade-off.

Thirdly, during my experiments with Ridge regression, I find that neither a small L2 regularization term nor a large L2 regularization term could help my 3 models improve regression performance. Using Ridge regression in my models would always tend to increase E{MSE} and to increase the difference between bias and variance, making my models less balanced according to bias-variance trade-off.

Question 2 (f) Solution:

Before formally submitting the complete print-out of my code, I want to show that I indeed structure and organize my code according to the requirements to make it modular and extensible.

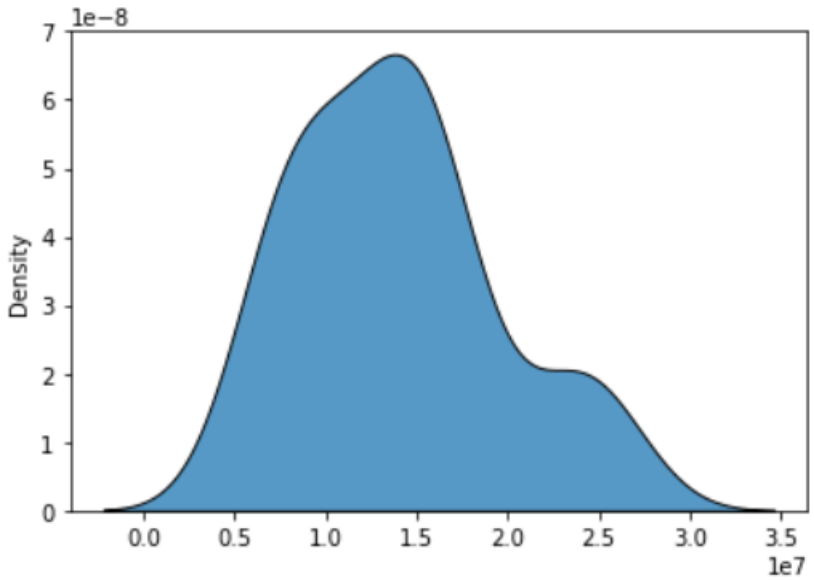
I write a function named “model_summary” integrating all of my previous self-defined functions. This function could generate all of the regression results in Homework 2 using a single code block that accepts:

5. A given data set (X and y for my regression model)
6. The desired norm
7. The desired regularization
8. The desired cross-validation parameters

An example of “a single code block” is shown below.

```
[18]: # Regularization Section
      # Model 1
      # Total model 4
      model_summary(X_one, y, alpha = 10000, norm = "L2", R = 3, K = 10)
```

```
Model E{MSE}                        = 13932504.3158
Model variance                      = 2005667.0544
Model [(bias)^2 + noise_variance]   = 11926837.2615

Plot a kernel density estimate for MSE of this model below
```



Please continue to the next page for the complete print-out of my code.

Question 2 (f):

Reference the Packages/Toolboxes

I use numpy to process intermediate arraies and matrices.
I use pandas to read and process the original dataset.
I use seaborn to draw the kernel density estimate plot for MSE.
I use RepeatedKFold from sklearn to implement 3-10-folds cross-validation.
I use Ridge from sklearn to implement L2 regression.
I use Lasso from sklearn to implement L1 regression.
(Lasso is not used in HW2, but it is useful as another desired norm option when structuring and organizing my code)
I use mean_squared_error from sklearn to calculate the mean square error (MSE).

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
```

Beginning of Data Preprocessing

Note: Data Preprocessing section has already been completed in Homework 1.
The purpose of this section is to help extract X_one, X_two, X_three, and y for my 3 proposed models.
X and y will be regarded as "a given data set" when I structure and organize my code in Homework 2.

[2]:
```
filename = "imports-85.data"
dataframe = pd.read_csv(filename, header = None)
```

[3]:
```
print("The number of data points in the original data set is: " + str(len(dataframe)))
```

```
The number of data points in the original data set is: 205
```

[4]:
```
# All 26 attributes in the data frame.
all_attributes = ["symboling", "normalized-losses", "make",
                  "fuel-type", "aspiration", "num-of-doors",
                  "body-style", "drive-wheels", "engine-location",
                  "wheel-base", "length", "width",
                  "height", "curb-weight", "engine-type",
                  "num-of-cylinders", "engine-size", "fuel-system",
                  "bore", "stroke", "compression-ratio",
                  "horsepower", "peak-rpm", "city-mpg",
                  "highway-mpg", "price"]
# Restrict ourselves to the 13 continuous predictor variables.
features = ["wheel-base", "length", "width",
            "height", "curb-weight", "engine-size",
            "bore", "stroke", "compression-ratio",
            "horsepower", "peak-rpm", "city-mpg", "highway-mpg"]
# Define the target variable.
target = "price"
# Assign each attribute to each column of the data frame correspondingly.
# This is because the original data set does not contain such information.
dataframe.columns = all_attributes
```

Question 2 (f):

[5]:
```
# Step 1 - Remove the (non-continuous) features that are not of interest.
old_len = len(dataframe)
for attribute in all_attributes :
    if (attribute not in features) and (attribute != target) :
        dataframe.drop(attribute, axis = 1, inplace = True)
    else :
        continue
new_len = len(dataframe)
print("Number of data points in my data frame before Step 1: " + str(old_len))
print("Number of data points in my data frame after Step 1: " + str(new_len))
print("Number of data points removed at Step 1: " + str(old_len - new_len))
```

```
Number of data points in my data frame before Step 1: 205
Number of data points in my data frame after Step 1: 205
Number of data points removed at Step 1: 0
```

[6]:
```
# Step 2 - Remove any data points for which the target variable (price) is unknown.
old_len = len(dataframe)
dataframe = dataframe[dataframe[target] != "?"]
new_len = len(dataframe)
print("Number of data points in my data frame before Step 2: " + str(old_len))
print("Number of data points in my data frame after Step 2: " + str(new_len))
print("Number of data points removed at Step 2: " + str(old_len - new_len))
```

```
Number of data points in my data frame before Step 2: 205
Number of data points in my data frame after Step 2: 201
Number of data points removed at Step 2: 4
```

[7]:
```
# Step 3 - Remove any data points for which the continuous predictor variables are unknown.
old_len = len(dataframe)
for feature in features :
    dataframe = dataframe[dataframe[feature] != "?"]
new_len = len(dataframe)
print("Number of data points in my data frame before Step 3: " + str(old_len))
print("Number of data points in my data frame after Step 3: " + str(new_len))
print("Number of data points removed at Step 3: " + str(old_len - new_len))
```

```
Number of data points in my data frame before Step 3: 201
Number of data points in my data frame after Step 3: 195
Number of data points removed at Step 3: 6
```

[8]:
```
# Do some housekeeping in this block.
df_numpy = dataframe.to_numpy()
for j in range(df_numpy.shape[1]) :
    for i in range(df_numpy.shape[0]) :
        df_numpy[i, j] = float(df_numpy[i, j])
features_matrix = df_numpy[:, 0:13]
target_vector = df_numpy[:, 13]
df_numpy = df_numpy.astype(np.float64)
print(features_matrix.shape)
print(target_vector.shape)
```

```
(195, 13)
(195,)
```

Question 2 (f):

[9]:
```
# Do some housekeeping in this block.
wheelbase  = np.copy(df_numpy[:, 0])
length     = np.copy(df_numpy[:, 1])
width      = np.copy(df_numpy[:, 2])
height     = np.copy(df_numpy[:, 3])
curbweight = np.copy(df_numpy[:, 4])
enginesize = np.copy(df_numpy[:, 5])
bore       = np.copy(df_numpy[:, 6])
stroke     = np.copy(df_numpy[:, 7])
comratio   = np.copy(df_numpy[:, 8])
horsepower = np.copy(df_numpy[:, 9])
peakrpm    = np.copy(df_numpy[:, 10])
citympg    = np.copy(df_numpy[:, 11])
highwaympg = np.copy(df_numpy[:, 12])
```

[10]:
```
# Do some housekeeping in this block.
X_one = enginesize.reshape(-1, 1)

X_two = np.column_stack((enginesize, horsepower))

X_three = np.column_stack((enginesize, horsepower,
                           width, width * width))

y = df_numpy[:, -1]

print(X_one.shape)
print(X_two.shape)
print(X_three.shape)
print(y.shape)
```

```
(195, 1)
(195, 2)
(195, 4)
(195,)
```

Ending of Data Preprocessing

Please continue to the next page for Homework 2 codes.

Question 2 (f):

Beginning of Codes for Homework 2

[11]:
```
def build_model (X, y,            # a given dataset
                 alpha,           # the desired regularization
                 norm = "L2",     # the desired norm
                 R = 3, K = 10): # the desired cross-validation parameters
    rkf = RepeatedKFold(n_splits = K, n_repeats = R,
                        # Pass an int for reproducible output
                        # across multiple function calls
                        random_state = 5)
    count = 0
    total_samples = R * K
    coef_list = np.zeros((total_samples, 1 + X.shape[1]))
    mse_list = np.zeros(total_samples)
    for train_index, test_index in rkf.split(X) :
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        if norm == "L2" :
            model = Ridge(alpha = alpha)
        if norm == "L1" :
            model = Lasso(alpha = alpha)
        model.fit(X_train, y_train)
        coef_list[count, 0] = model.intercept_
        coef_list[count, 1:] = model.coef_
        preds = model.predict(X_test)
        mse_list[count] = mean_squared_error(y_true = y_test,
                                             y_pred = preds)
        count = count + 1
    return coef_list, mse_list
```

[12]:
```
def E_mse (mse_list) :
    mse_sum = 0
    for mse in mse_list :
        mse_sum = mse_sum + mse
    avg_mse = float(mse_sum) / len(mse_list)
    return avg_mse

def plot_kde (mse_list) :
    sns.kdeplot(data = mse_list, multiple = "stack")
    return None

def reconstruct_model (Xn, coef_list, l) :
    yl_Xn = 0
    for i in range(1 + len(Xn)) :
        if i == 0 :
            yl_Xn = yl_Xn + coef_list[l, i]
        else :
            yl_Xn = yl_Xn + coef_list[l, i] * Xn[i - 1]
    return yl_Xn
```

Question 2 (f):

```
def calculate_avgpred (Xn, coef_list) :
    L = len(coef_list)
    yn_sum = 0
    yn_avg = 0
    for l in range(L) :
        yn_sum = yn_sum + reconstruct_model(Xn, coef_list, l)
    yn_avg = float(yn_sum) / L
    return yn_avg
```

[13]:
```
def calculate_variance (X, coef_list) :
    N = X.shape[0]
    L = len(coef_list)
    sum_inside = 0
    avg_inside = 0
    sum_outside = 0
    avg_outside = 0   # variance = avg_outside
    for n in range(N) :
        Xn = X[n]
        yn_avg = calculate_avgpred(Xn, coef_list)
        for l in range(L) :
            subdiff = reconstruct_model(Xn, coef_list, l) - yn_avg
            sub_sqr = subdiff * subdiff
            sum_inside = sum_inside + sub_sqr
        avg_inside = float(sum_inside) / L
        sum_outside = sum_outside + avg_inside
    avg_outside = float(sum_outside) / N
    return avg_outside

def calculate_bias_noise (X, coef_list, mse_list) :
    # E{MSE} = variance + (bias)^2 + noise_variance
    # (bias)^2 + noise_variance = E{MSE} - variance
    result = E_mse(mse_list) - calculate_variance(X, coef_list)
    return result
```

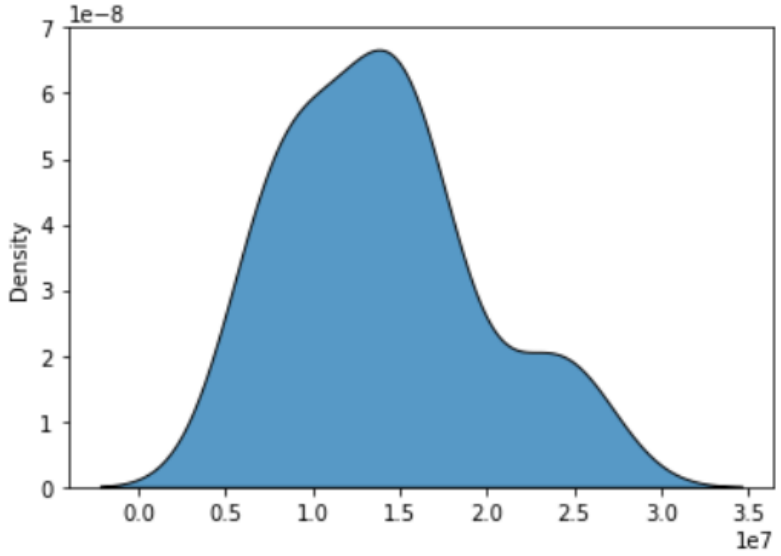
[14]:
```
# The final "exam" for structuring and organizing my code

def model_summary (X, y,            # a given data set
                   alpha,           # the desired regularization
                   norm = "L2",     # the desired norm
                   R = 3, K = 10): # the desired cross-validation parameters

    coef_list, mse_list = build_model(X = X, y = y, alpha = alpha,
                                      norm = "L2",
                                      R = 3, K = 10)
    avg_mse = E_mse(mse_list)
    variance = calculate_variance(X, coef_list)
    bias_noise = calculate_bias_noise(X, coef_list, mse_list)
    print("Model E{MSE}                        = %.4f" % avg_mse)
    print("Model variance                      = %.4f" % variance)
    print("Model [(bias)^2 + noise_variance] = %.4f" % bias_noise)
    print(" ")
    print("Plot a kernel density estimate for MSE of this model below")
    plot_kde(mse_list)
    return None
```

Question 2 (f):

[18]:
```
# Regularization Section
# Model 1
# Total model 4
model_summary(X_one, y, alpha = 10000, norm = "L2", R = 3, K = 10)
```

```
Model E{MSE}                        = 13932504.3158
Model variance                      = 2005667.0544
Model [(bias)^2 + noise_variance]   = 11926837.2615
```

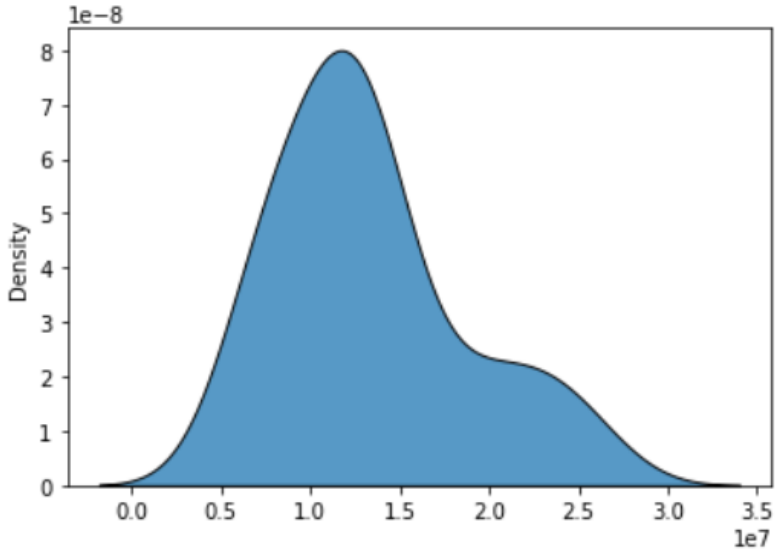
Plot a kernel density estimate for MSE of this model below



[19]:
```
# Regularization Section
# Model 2
# Total Model 5
model_summary(X_two, y, alpha = 10000, norm = "L2", R = 3, K = 10)
```

```
Model E{MSE}                        = 13345028.5078
Model variance                      = 3256561.7754
Model [(bias)^2 + noise_variance]   = 10088466.7324
```

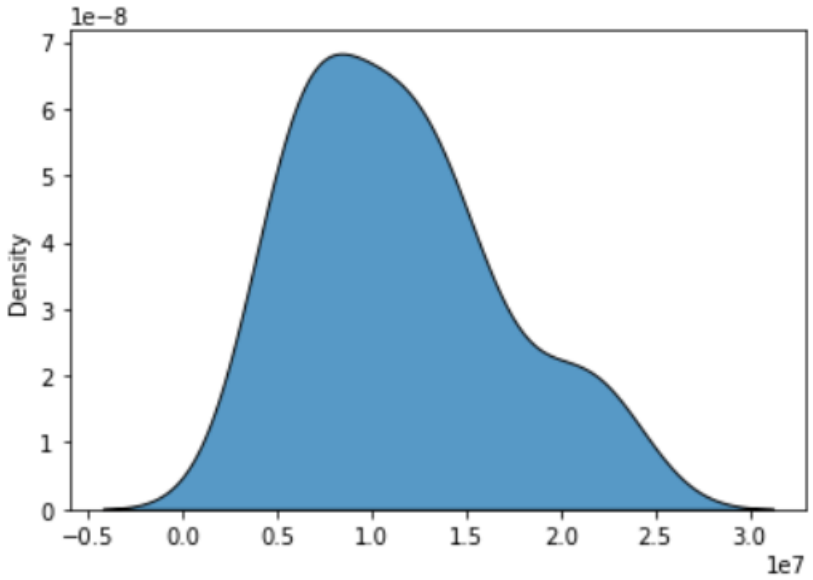
Plot a kernel density estimate for MSE of this model below



Question 2 (f):

[20]:
```
# Regularization Section
# Model 3
# Total Model 6
model_summary(X_three, y, alpha = 1, norm = "L2", R = 3, K = 10)
```

```
Model E{MSE}                        = 11450997.9716
Model variance                      = 3829208.8159
Model [(bias)^2 + noise_variance]   = 7621789.1558

Plot a kernel density estimate for MSE of this model below
```



Ending of Codes for Homework 2