Applied Machine learning : Homework 1 Question 2

Name: Yukta Sanjay Muthreja

Student ID: 2001270781

Name of all the attributes: ID Price: price of the care(Target Column) Levy Manufacturer Model Prod. year Category Leather interior Fuel type Engine volume Mileage Cylinders Gear box type Drive wheels Doors Wheel Color Airbags

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
car =pd.read_csv("/Users/yuktamuthreja/Library/CloudStorage/OneDrive-IndianaUn
car.head()
```

Out[2]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume
0	45654403	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Hybrid	3.5
1	44731507	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petrol	3
2	45774419	8467	-	HONDA	FIT	2006	Hatchback	No	Petrol	1.3
3	45769185	3607	862	FORD	Escape	2011	Jeep	Yes	Hybrid	2.5
4	45809263	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petrol	1.3

Question A. Summarize the data. How much data is present? What attributes/features are continuous valued? Which attributes are categorical?

The Length of the dataset:

In [3]:
```
car.shape
```

Out[3]:
```
(19237, 18)
```

General description of the data values, along with it's count and data type

In [4]:
```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19237 entries, 0 to 19236
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                19237 non-null  int64  
 1   Price             19237 non-null  int64  
 2   Levy              19237 non-null  object 
 3   Manufacturer      19237 non-null  object 
 4   Model             19237 non-null  object 
 5   Prod. year        19237 non-null  int64  
 6   Category          19237 non-null  object 
 7   Leather interior  19237 non-null  object 
 8   Fuel type         19237 non-null  object 
 9   Engine volume     19237 non-null  object 
 10  Mileage           19237 non-null  object 
 11  Cylinders         19237 non-null  float64
 12  Gear box type     19237 non-null  object 
 13  Drive wheels      19237 non-null  object 
 14  Doors             19237 non-null  object 
 15  Wheel             19237 non-null  object 
 16  Color             19237 non-null  object 
 17  Airbags           19237 non-null  int64  
dtypes: float64(1), int64(4), object(13)
memory usage: 2.6+ MB
```

Analyzing the above outputs, the the attributes can be classified as follows:

- ID, Price, Levy, Production Year, Mileage, Airbags, Cylinder and Engine Volume are continuous variables.
- Manufacturer, Model, Category, Leather interior, Fuel Type, Gear box type, Drive wheels, Color and Doors are categorical variables

Note: Although the output of the data type and the classification may be different, the variables will be pre-proccessed to be classified accordingly.

Question B: Display the statistical values for each of the attributes, along with visualizations (e.g., histogram) of the distributions for each attribute. Explain noticeable traits for key attributes. Are there any attributes that might require special treatment? If so, what special treatment might they require?

Analyzing the statistical values for each of the numerical attributes

In [5]:
```
car.describe()
```

Out[5]:

	ID	Price	Prod. year	Cylinders	Airbags
count	1.923700e+04	1.923700e+04	19237.000000	19237.000000	19237.000000
mean	4.557654e+07	1.855593e+04	2010.912824	4.582991	6.582627
std	9.365914e+05	1.905813e+05	5.668673	1.199933	4.320168
min	2.074688e+07	1.000000e+00	1939.000000	1.000000	0.000000
25%	4.569837e+07	5.331000e+03	2009.000000	4.000000	4.000000
50%	4.577231e+07	1.317200e+04	2012.000000	4.000000	6.000000
75%	4.580204e+07	2.207500e+04	2015.000000	4.000000	12.000000
max	4.581665e+07	2.630750e+07	2020.000000	16.000000	16.000000

However, we have noticed that there are additional numerical columns that need some pre-processing. Further, we can analyze the frequencies of each attribute.

In [6]:
```
for i in ['ID', 'Price', 'Levy', 'Manufacturer', 'Model', 'Prod. year',
       'Category', 'Leather interior', 'Fuel type', 'Engine volume', 'Mileage',
       'Cylinders', 'Gear box type', 'Drive wheels', 'Doors', 'Wheel', 'Color',
       'Airbags']:
    print(car[i].value_counts())
```

```
45815365    8
45815361    8
45815363    7
45815368    7
45723475    7
           ..
45774312    1
45732621    1
45773011    1
45774019    1
45813273    1
Name: ID, Length: 18924, dtype: int64
15681    280
470      274
14113    244
392      242
314      235
        ...
42601      1
149        1
54349      1
54954      1
22075      1
Name: Price, Length: 2315, dtype: int64
-       5819
765      486
891      461
639      410
640      405
        ...
3156       1
2908       1
1279       1
1719       1
1901       1
Name: Levy, Length: 559, dtype: int64
HYUNDAI          3769
TOYOTA           3662
MERCEDES-BENZ    2076
FORD             1111
CHEVROLET        1069
                 ...
TESLA               1
PONTIAC             1
SATURN              1
ASTON MARTIN        1
GREATWALL           1
Name: Manufacturer, Length: 65, dtype: int64
Prius                     1083
Sonata                    1079
Camry                      938
Elantra                    922
E 350                      542
                          ...
Feroza                       1
C-MAX C-MAX                  1
X1 4X4                       1
Land Cruiser Prado RX        1
Prius C aqua                 1
Name: Model, Length: 1590, dtype: int64
```

```
2012    2155
2014    2124
2013    1963
2011    1612
2015    1549
2010    1483
2016    1476
2017     959
2008     737
2009     601
2018     500
2007     464
2005     402
2003     367
2004     364
2006     317
2019     306
2002     296
2000     279
2001     254
1998     213
1999     207
1997     151
1996     114
1995     105
2020      47
1994      42
1992      30
1993      23
1990      18
1988      12
1991      10
1986       6
1989       6
1987       5
1984       5
1985       5
1953       4
1983       3
1939       3
1978       2
1980       2
1965       2
1977       2
1974       2
1964       2
1943       1
1976       1
1957       1
1968       1
1947       1
1982       1
1981       1
1973       1
Name: Prod. year, dtype: int64
Sedan          8736
Jeep           5473
Hatchback      2847
Minivan         647
Coupe           532
```

```
Universal       364
Microbus        306
Goods wagon     233
Pickup           52
Cabriolet        36
Limousine        11
Name: Category, dtype: int64
Yes    13954
No      5283
Name: Leather interior, dtype: int64
Petrol            10150
Diesel             4036
Hybrid             3578
LPG                 892
CNG                 494
Plug-in Hybrid       86
Hydrogen              1
Name: Fuel type, dtype: int64
2            3916
2.5          2277
1.8          1760
1.6          1462
1.5          1321
             ...
6.8             1
6.7             1
3.1             1
0.8 Turbo       1
1.1 Turbo       1
Name: Engine volume, Length: 107, dtype: int64
0 km         721
200000 km    183
150000 km    161
160000 km    120
100000 km    119
            ...
63083 km       1
28750 km       1
25077 km       1
77452 km       1
186923 km      1
Name: Mileage, Length: 7687, dtype: int64
4.0     14367
6.0      3462
8.0       991
5.0       169
3.0       107
2.0        42
1.0        38
12.0       38
10.0       12
16.0        5
7.0         4
9.0         1
14.0        1
Name: Cylinders, dtype: int64
Automatic    13514
Tiptronic     3102
Manual        1875
Variator       746
```

```
Name: Gear box type, dtype: int64
Front    12874
4x4       4058
Rear      2305
Name: Drive wheels, dtype: int64
04-May    18332
02-Mar      777
>5          128
Name: Doors, dtype: int64
Left wheel          17753
Right-hand drive     1484
Name: Wheel, dtype: int64
Black            5033
White            4489
Silver           3792
Grey             2375
Blue             1396
Red               639
Green             322
Orange            253
Brown             187
Carnelian red     179
Golden            145
Beige             134
Sky blue          122
Yellow            106
Purple             39
Pink               26
Name: Color, dtype: int64
4     5823
12    5654
0     2405
8     1608
6     1311
2     1066
10     849
5      104
16      93
7       86
1       76
9       63
3       37
11      33
14      20
15       7
13       2
Name: Airbags, dtype: int64
```

Observing the data from a high-level perspective, we notice the following:

- ID: is a primary key of the dataset (as each car sale has a unique ID) and we won't be adding this to the prediction model. Hence we can drop it.
- Price: this is the value that we would need to predict
- Levy: We can notice that there is data with "-" as the value, we would have to replace that to 0 and will that with the closest prediction.
- Manufacturer, Model and Category: There are a few values which are occuring only once, we will fix the data when we are treating it for outliers.

- Engine volume and Mileage - We would have treat this attribute to read it as a numerical variable
- Color: From the value counts we can see that the data is not equally spread.

Before analyzing the data using histograms, we will clean the data such that we can better analyze the trends and the spread of the data.

In [7]:
```
Mileage_new = [float(value.split(' ')[0]) for value in car['Mileage']]
car['Mileage'] = Mileage_new # replacing mileage data from the original dataset
```

In [8]:
```
Enginevol_new = [float(value.split(' ')[0]) for value in car['Engine volume']]
car['Engine volume'] = Enginevol_new # replacing engine volume data from the or
```

In [9]:
```
car["Levy"] = np.where(car["Levy"] == "-" ,0 , car["Levy"]).astype(int) #Replac
```

In [10]:
```
car.head()
```

Out[10]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume
0	45654403	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Hybrid	3.5
1	44731507	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petrol	3.0
2	45774419	8467	0	HONDA	FIT	2006	Hatchback	No	Petrol	1.3
3	45769185	3607	862	FORD	Escape	2011	Jeep	Yes	Hybrid	2.5
4	45809263	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petrol	1.3

In [11]:
```
car.info()
```

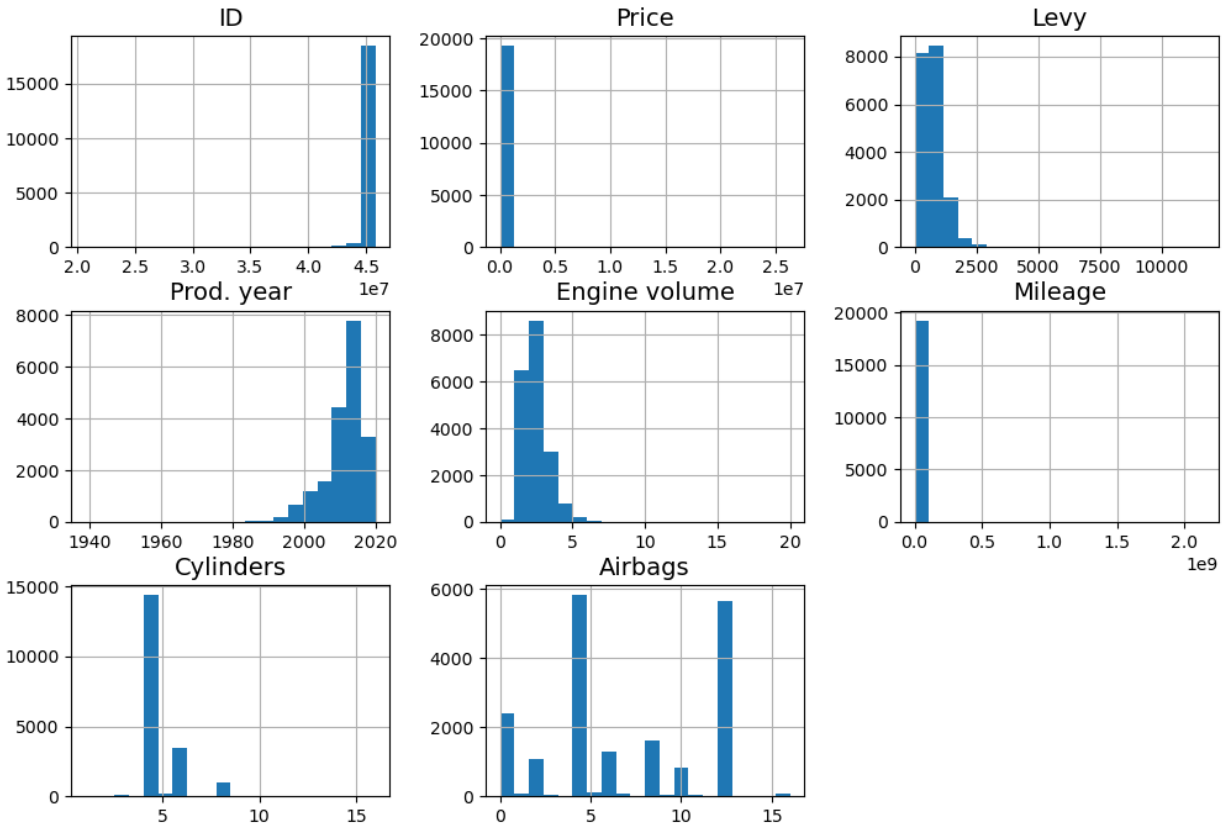
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19237 entries, 0 to 19236
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                19237 non-null  int64  
 1   Price             19237 non-null  int64  
 2   Levy              19237 non-null  int64  
 3   Manufacturer      19237 non-null  object 
 4   Model             19237 non-null  object 
 5   Prod. year        19237 non-null  int64  
 6   Category          19237 non-null  object 
 7   Leather interior  19237 non-null  object 
 8   Fuel type         19237 non-null  object 
 9   Engine volume     19237 non-null  float64
 10  Mileage           19237 non-null  float64
 11  Cylinders         19237 non-null  float64
 12  Gear box type     19237 non-null  object 
 13  Drive wheels      19237 non-null  object 
 14  Doors             19237 non-null  object 
 15  Wheel             19237 non-null  object 
 16  Color             19237 non-null  object 
 17  Airbags           19237 non-null  int64  
dtypes: float64(3), int64(5), object(10)
memory usage: 2.6+ MB
```

In [12]:
```
car.describe() #Reviewing the statistical values for each of the cleaned numeri
```

Out[12]:

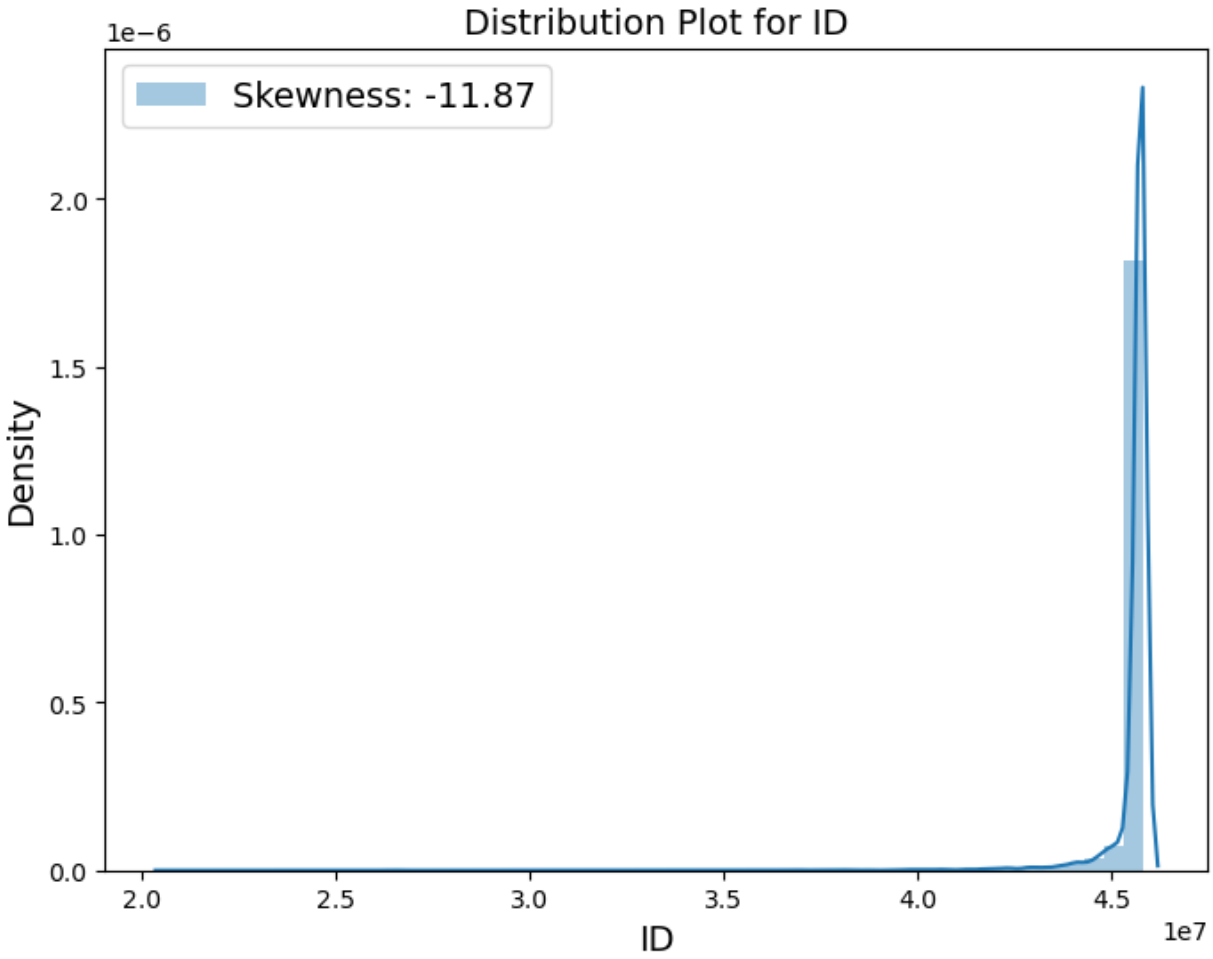
	ID	Price	Levy	Prod. year	Engine volume	Mileage
count	1.923700e+04	1.923700e+04	19237.000000	19237.000000	19237.000000	1.923700e+04
mean	4.557654e+07	1.855593e+04	632.528669	2010.912824	2.307990	1.532236e+06
std	9.365914e+05	1.905813e+05	567.721688	5.668673	0.877805	4.840387e+07
min	2.074688e+07	1.000000e+00	0.000000	1939.000000	0.000000	0.000000e+00
25%	4.569837e+07	5.331000e+03	0.000000	2009.000000	1.800000	7.013900e+04
50%	4.577231e+07	1.317200e+04	642.000000	2012.000000	2.000000	1.260000e+05
75%	4.580204e+07	2.207500e+04	917.000000	2015.000000	2.500000	1.888880e+05
max	4.581665e+07	2.630750e+07	11714.000000	2020.000000	20.000000	2.147484e+09

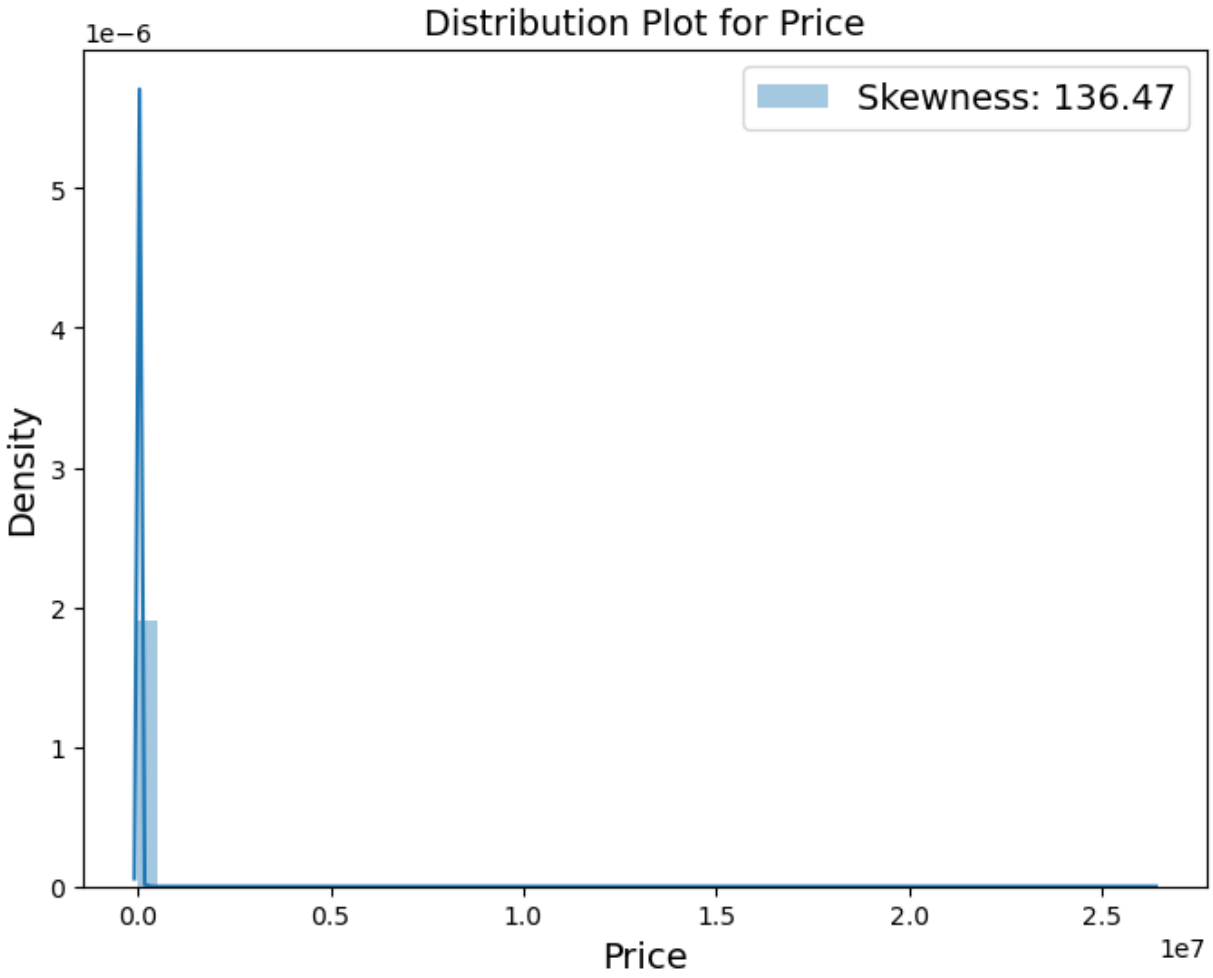
In [13]:
```
plt.rc('font', size=14)
plt.rc('axes', labelsize=14, titlesize=14)
plt.rc('legend', fontsize=14)
plt.rc('xtick', labelsize=10)
plt.rc('ytick', labelsize=10)

car.hist(bins=20, figsize=(12, 8))
plt.show()

#Ref: Code from chapter 2 of Hands-on Machine Learning with Scikit-Learn, Keras
```

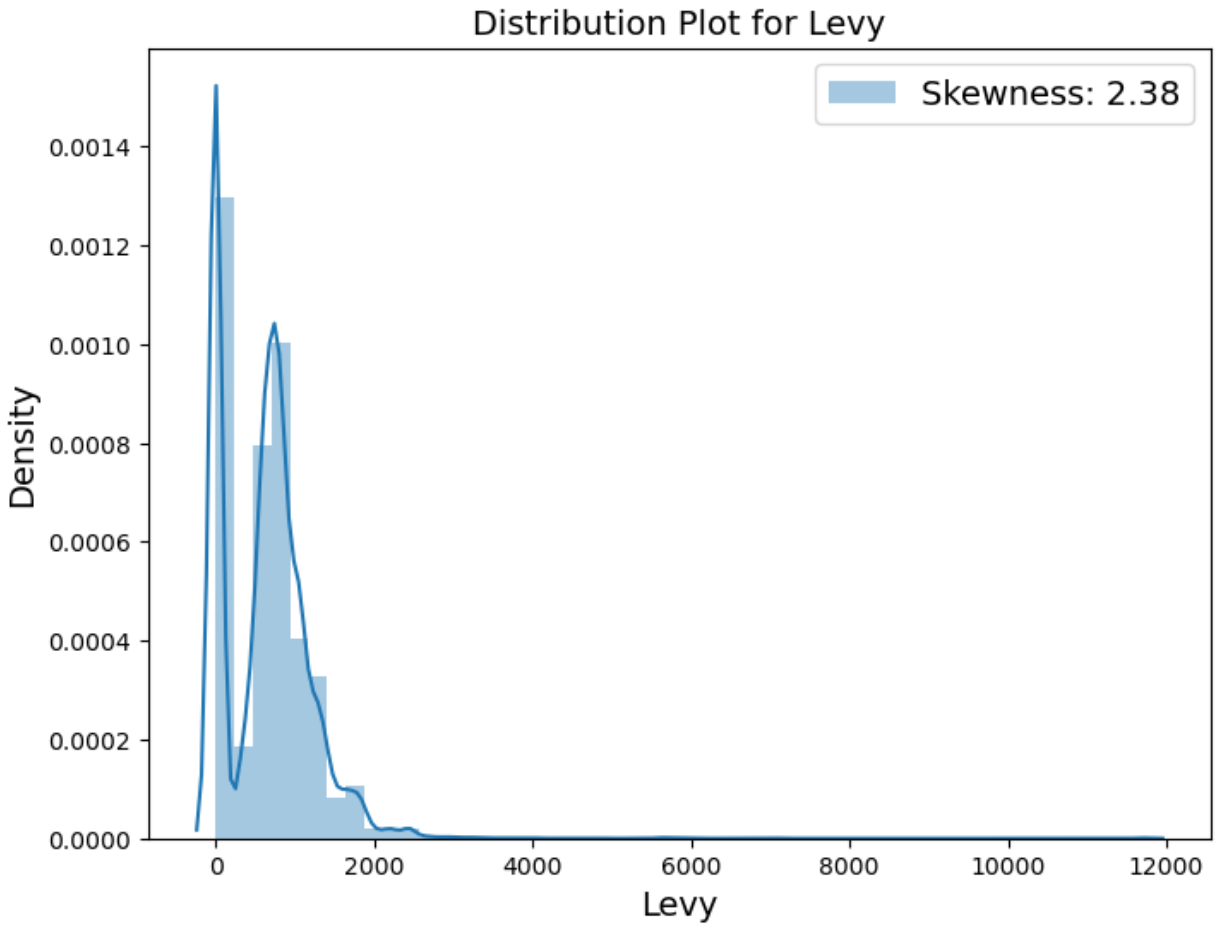


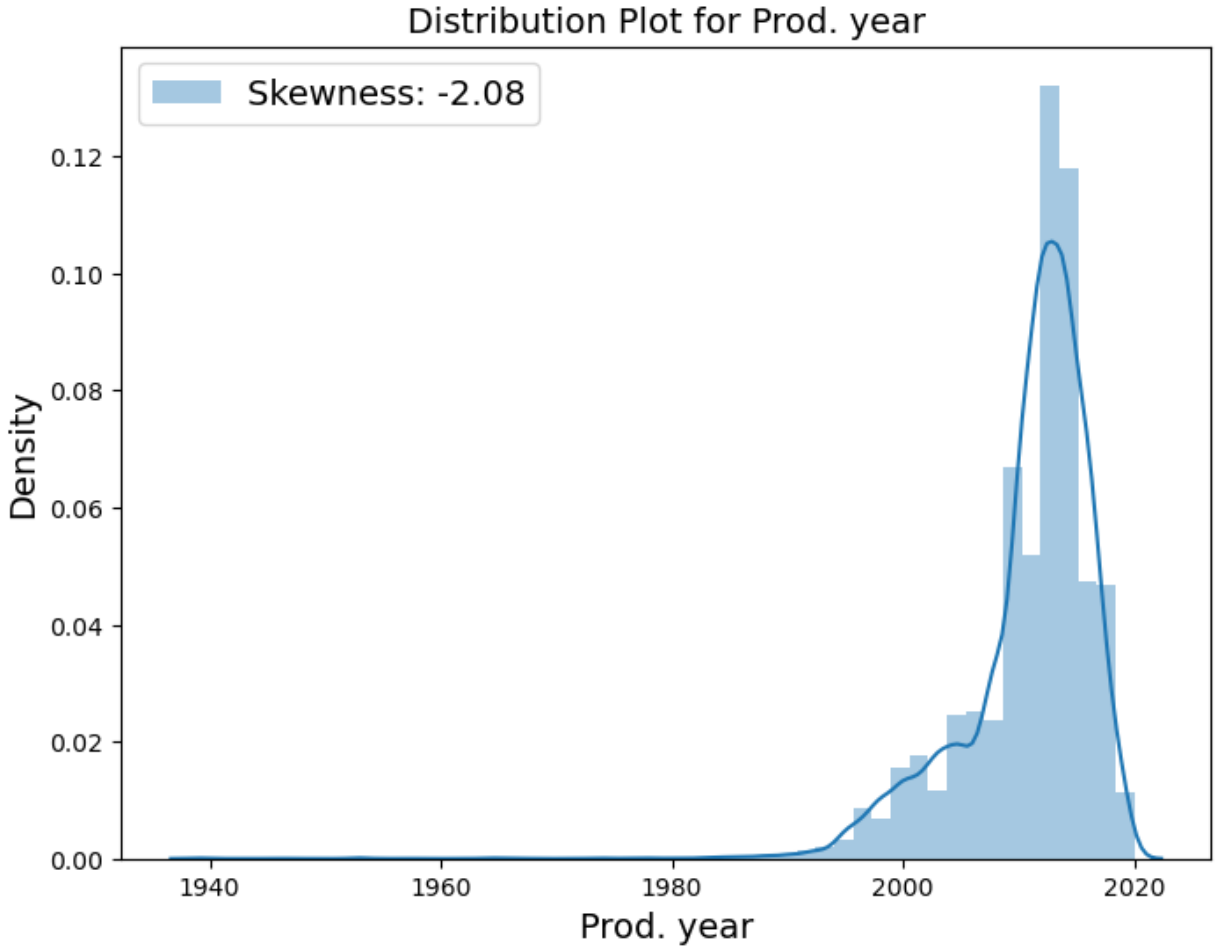
In [14]:
```
car_numerical = car.select_dtypes(include=np.number)
```

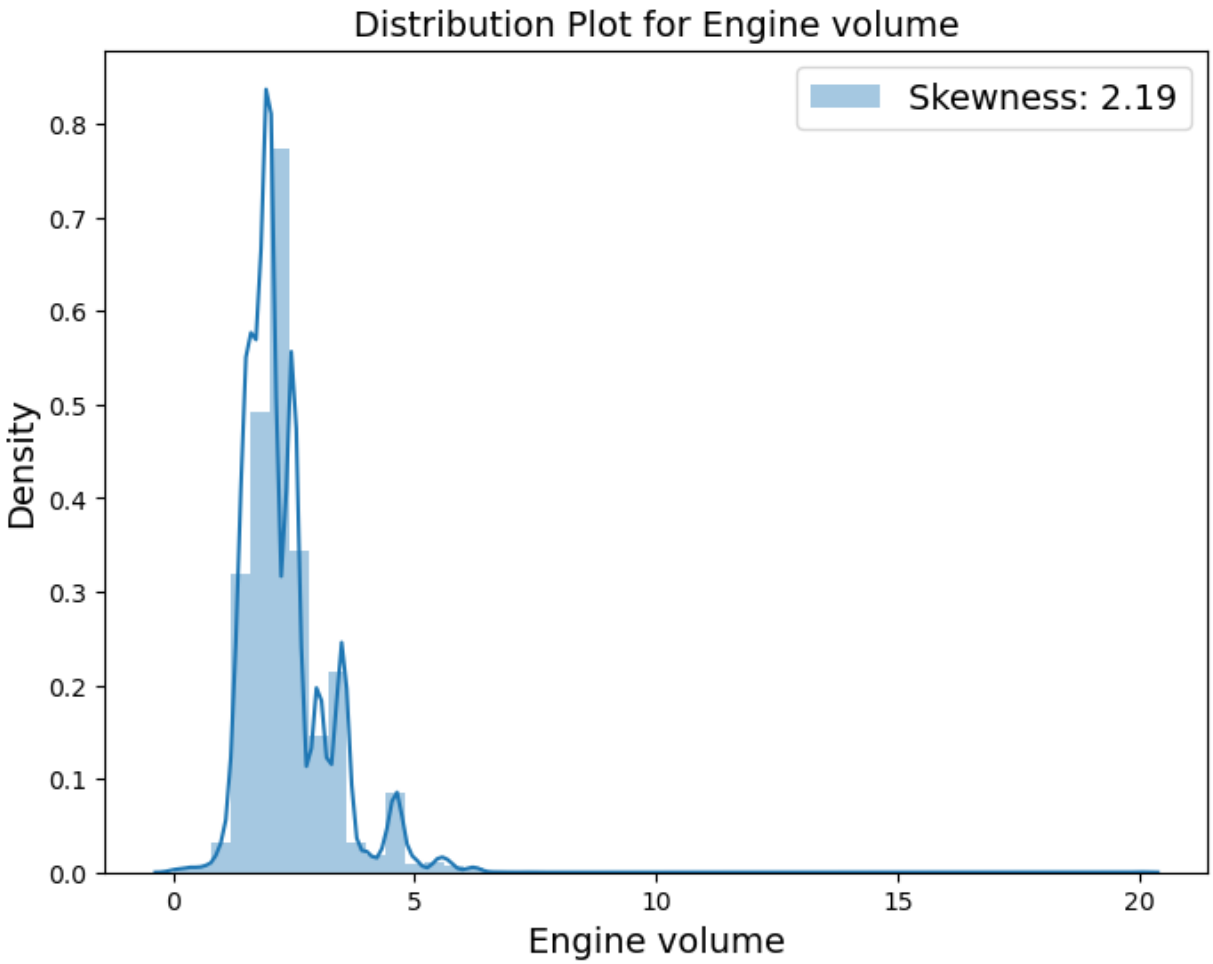
In [15]:
```
import warnings

warnings.filterwarnings("ignore")

for i in car_numerical.columns:
    plt.figure(figsize=(8, 6))  # Adjust the figure size as needed
    sns.distplot(car[i], label='Skewness: %.2f' % (car[i].skew()))
    plt.legend()
    plt.title(f'Distribution Plot for {i}')
    plt.show()

warnings.resetwarnings()
```

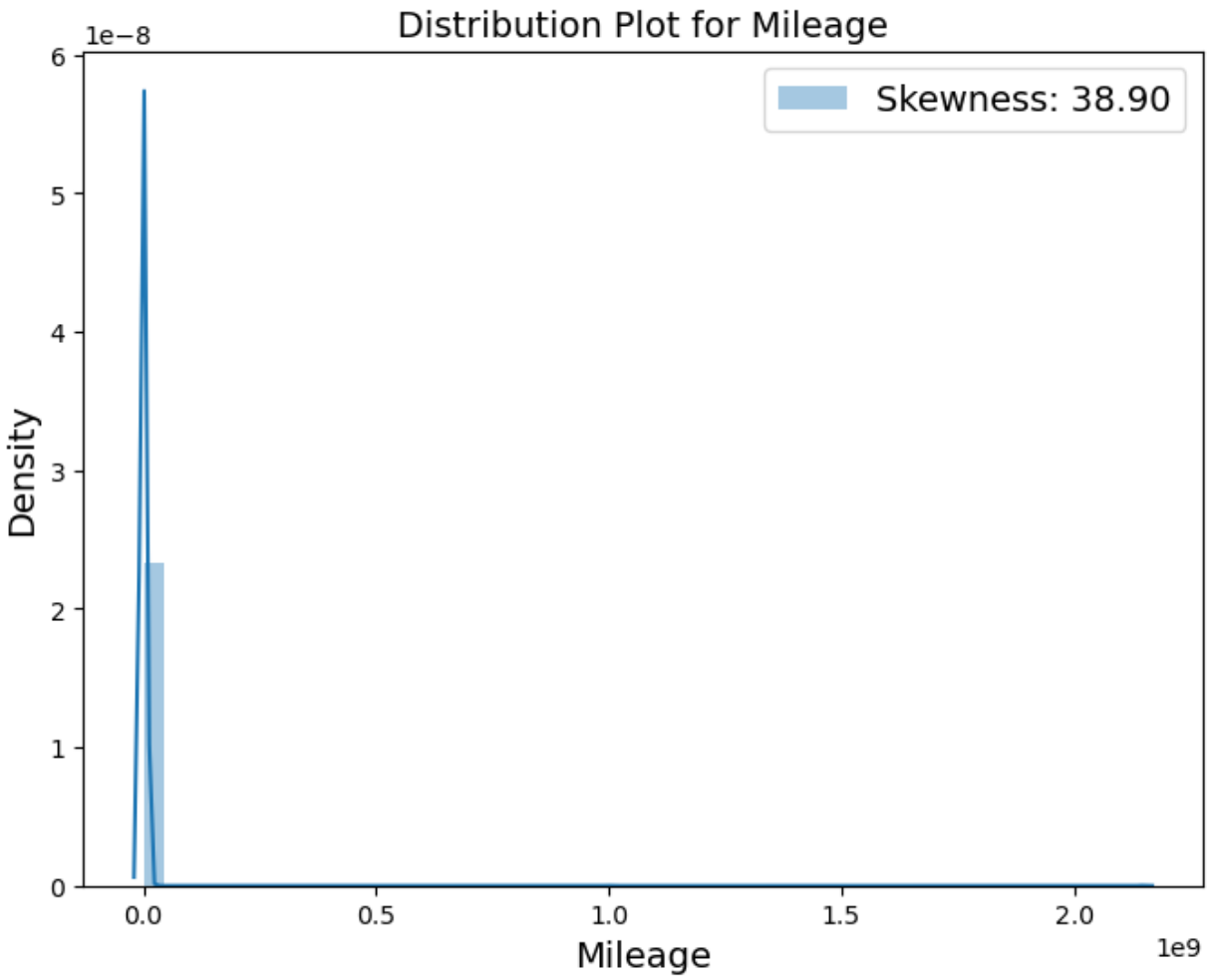

Distribution Plot for ID
1e−6
Skewness: -11.87
Density
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5
1e7
ID

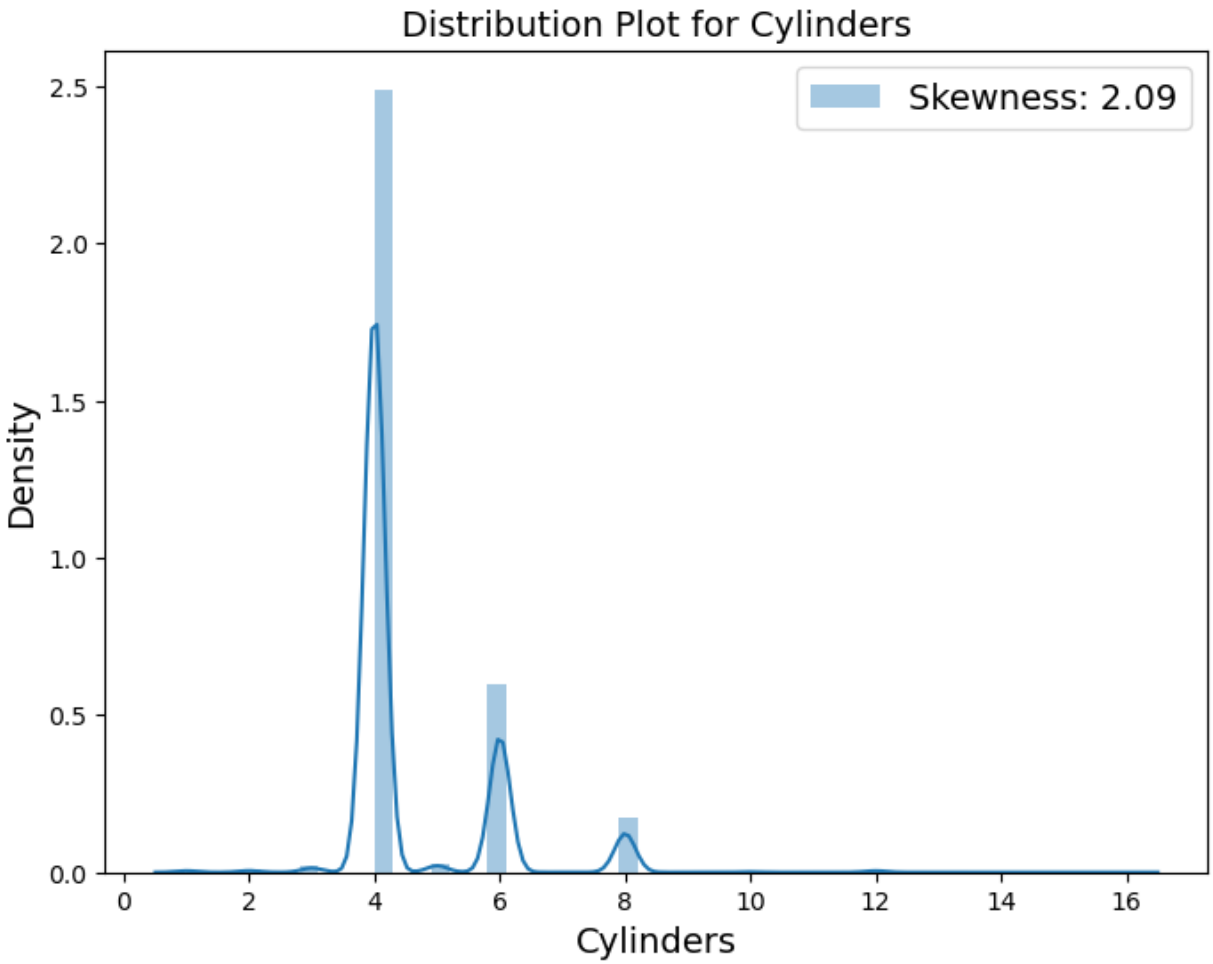

Distribution Plot for Price
1e−6
Skewness: 136.47
Density
0
1
2
3
4
5
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Price

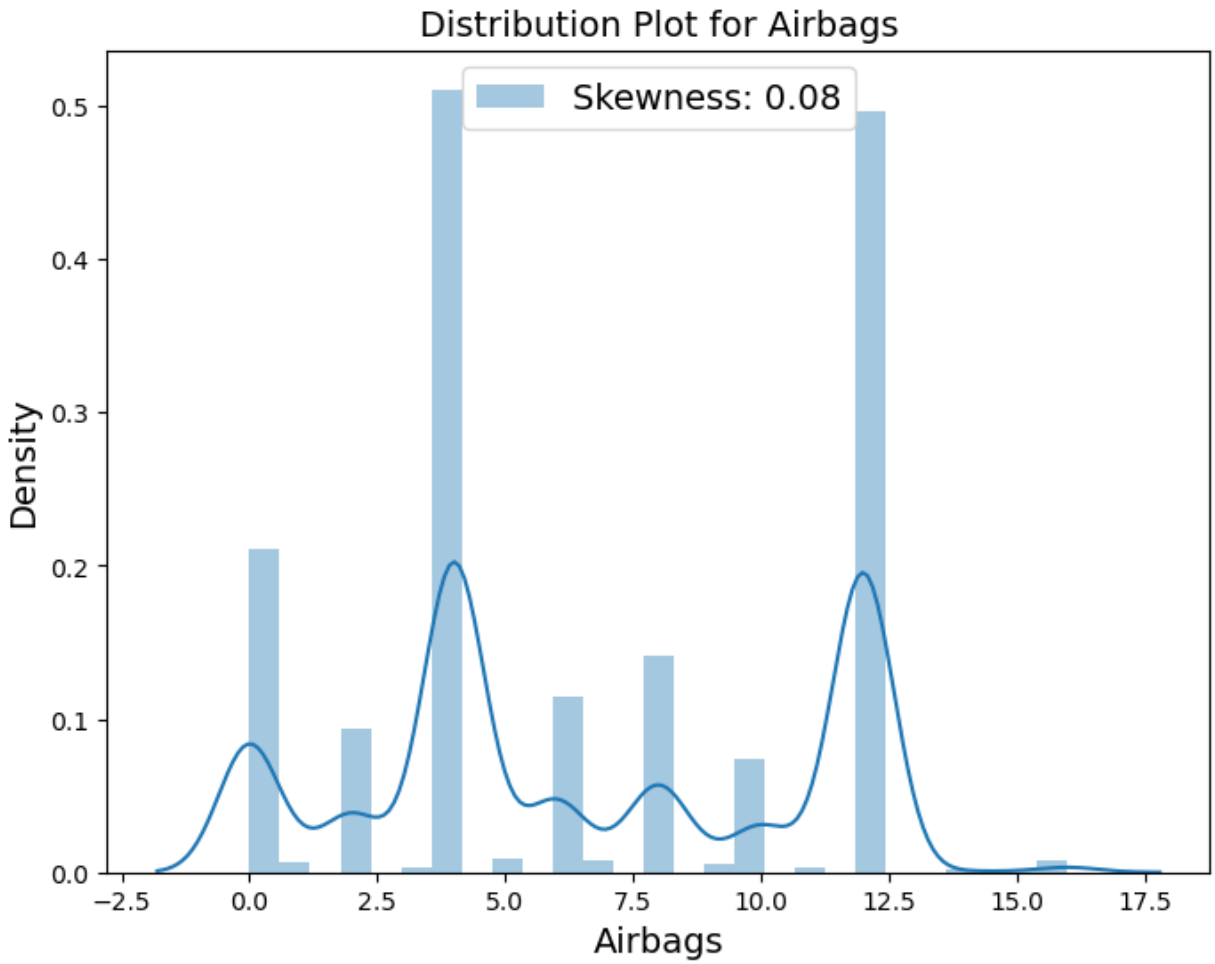

Distribution Plot for Levy
Skewness: 2.38
Density
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0
2000
4000
6000
8000
10000
12000
Levy


Distribution Plot for Prod. year
Skewness: -2.08
Density
0.12
0.10
0.08
0.06
0.04
0.02
0.00
1940
1960
1980
2000
2020
Prod. year


Distribution Plot for Engine volume
Skewness: 2.19
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
Engine volume


Distribution Plot for Mileage
1e−8
Skewness: 38.90
Density
Mileage
1e9
0
1
2
3
4
5
6
0.0
0.5
1.0
1.5
2.0


Distribution Plot for Cylinders
Skewness: 2.09
Density
Cylinders
0.0
0.5
1.0
1.5
2.0
2.5
0
2
4
6
8
10
12
14
16



From the above distributions, we notice the following:

- Price and Mileage are highly skewed, we will adjust for the skeweness during the data pre-processing stage.
- There are primarily only 3 different cylinders, that is, 4,6 and 8. Hence we will clean the data to make this more representative

Question C: Analyze and discuss the relationships between the data attributes, and between the data attributes and label. This involves computing the Pearson Correlation Coefficient (PCC) and generating scatter plots.

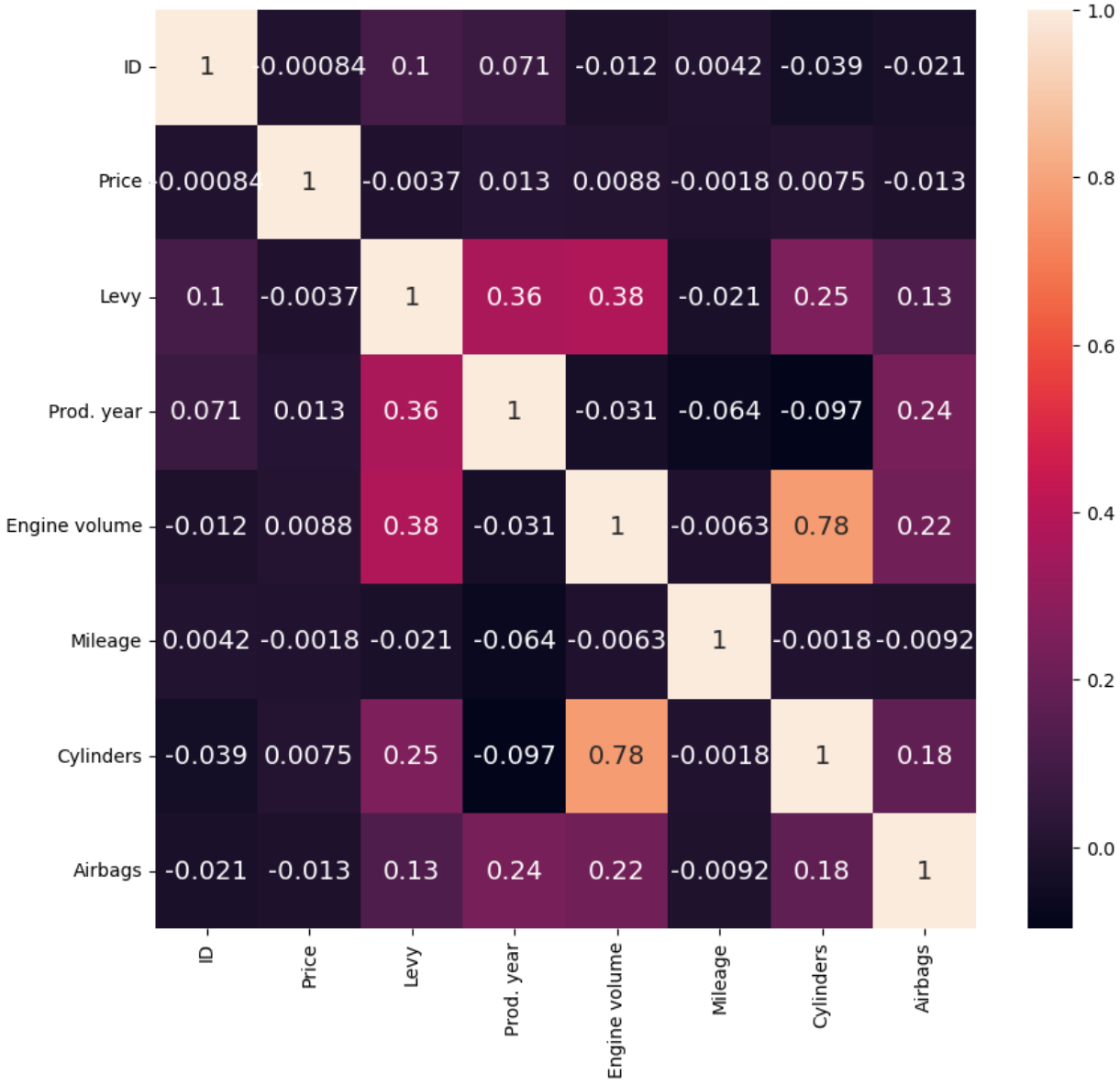
In [16]:
```
car_numerical.corr()
```

Out[16]:

	ID	Price	Levy	Prod. year	Engine volume	Mileage	Cylinders	Airb
ID	1.000000	-0.000844	0.102614	0.071352	-0.012232	0.004157	-0.039319	-0.02(
Price	-0.000844	1.000000	-0.003701	0.012982	0.008753	-0.001758	0.007518	-0.01
Levy	0.102614	-0.003701	1.000000	0.364712	0.377438	-0.021034	0.250950	0.12
Prod. year	0.071352	0.012982	0.364712	1.000000	-0.030906	-0.063501	-0.096797	0.23
Engine volume	-0.012232	0.008753	0.377438	-0.030906	1.000000	-0.006289	0.778524	0.22
Mileage	0.004157	-0.001758	-0.021034	-0.063501	-0.006289	1.000000	-0.001768	-0.00
Cylinders	-0.039319	0.007518	0.250950	-0.096797	0.778524	-0.001768	1.000000	0.17
Airbags	-0.020527	-0.012824	0.128032	0.236969	0.224441	-0.009201	0.176868	1.00(

In [17]:
```
fig,ax=plt.subplots(figsize=(10,9))
sns.heatmap(car_numerical.corr(),annot=True)
```

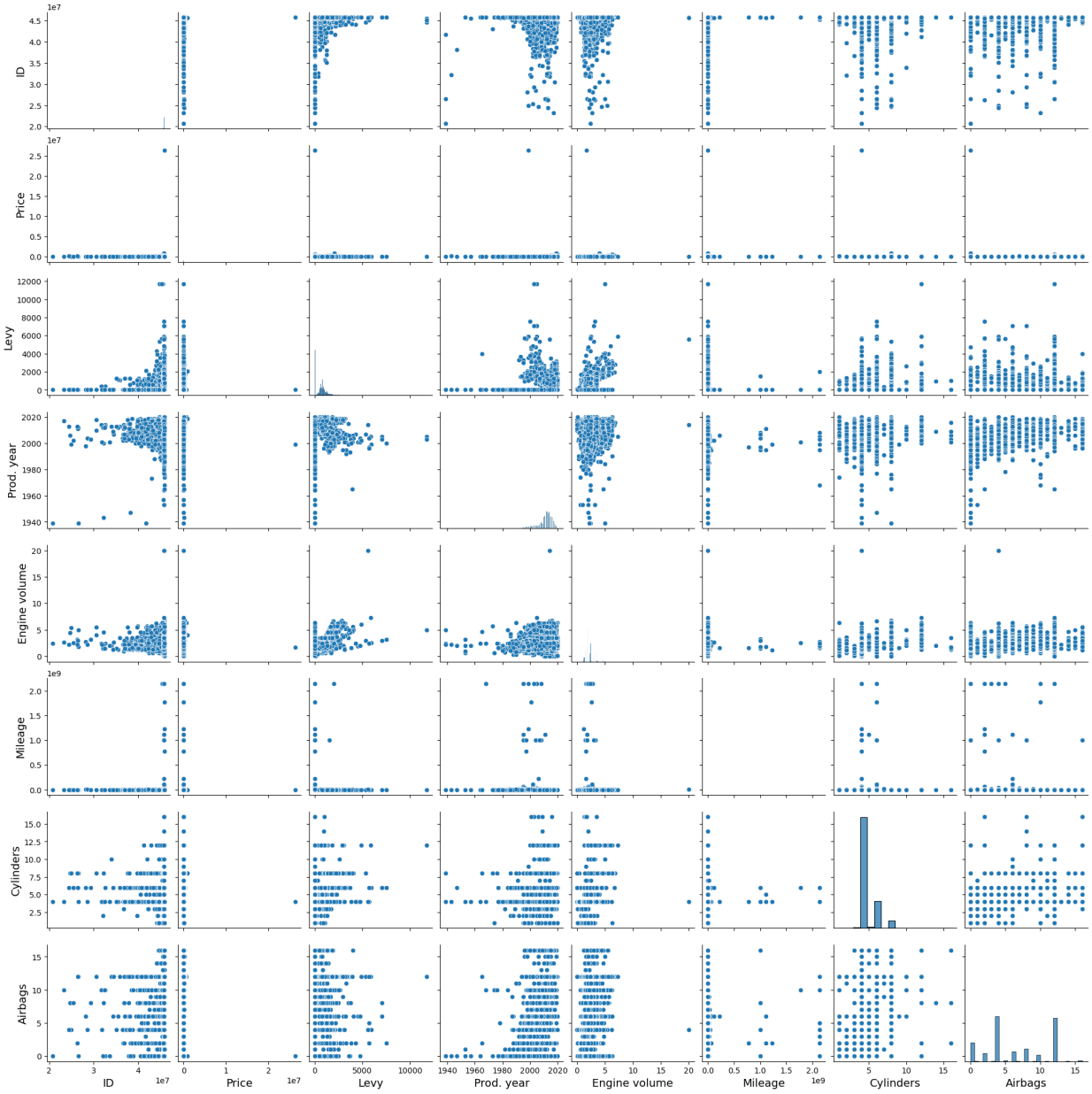
Out[17]:
```
<Axes: >
```



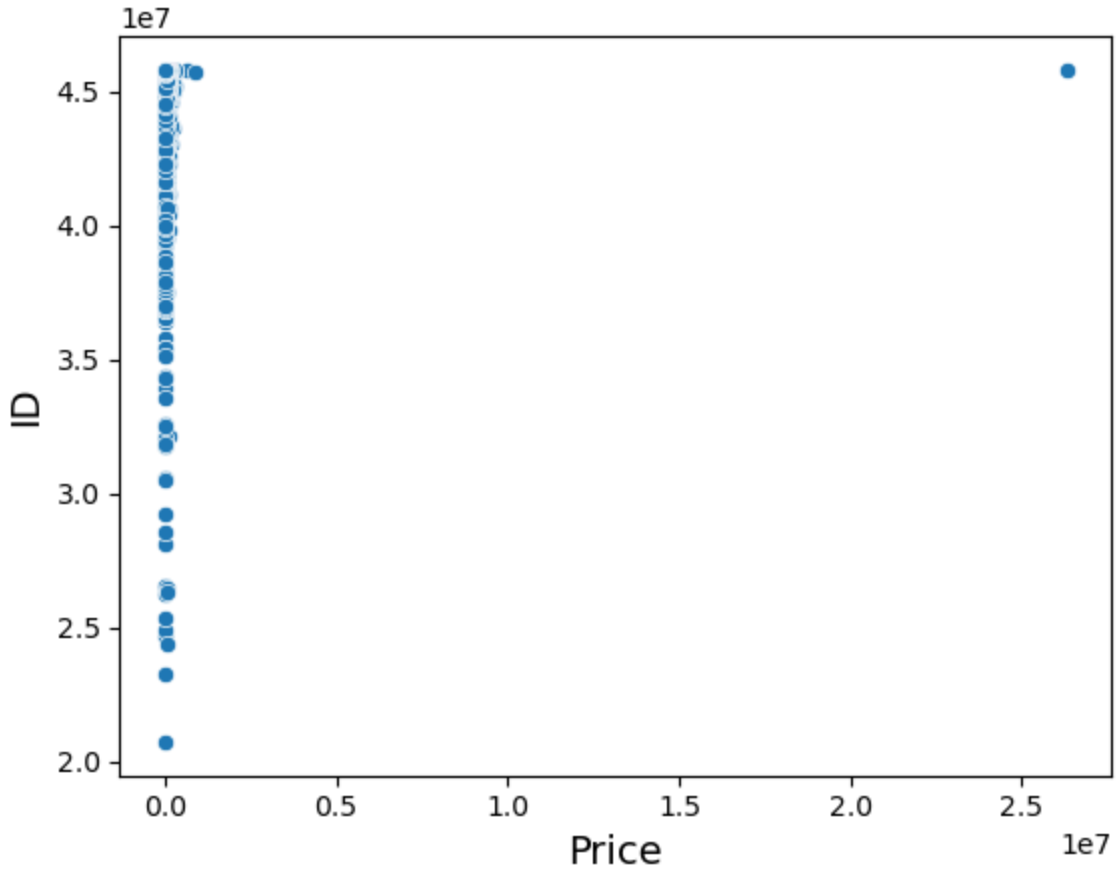
In [18]:
```
car_label = car["Price"]
car_attributes = car.drop("Price", errors = "ignore")
```

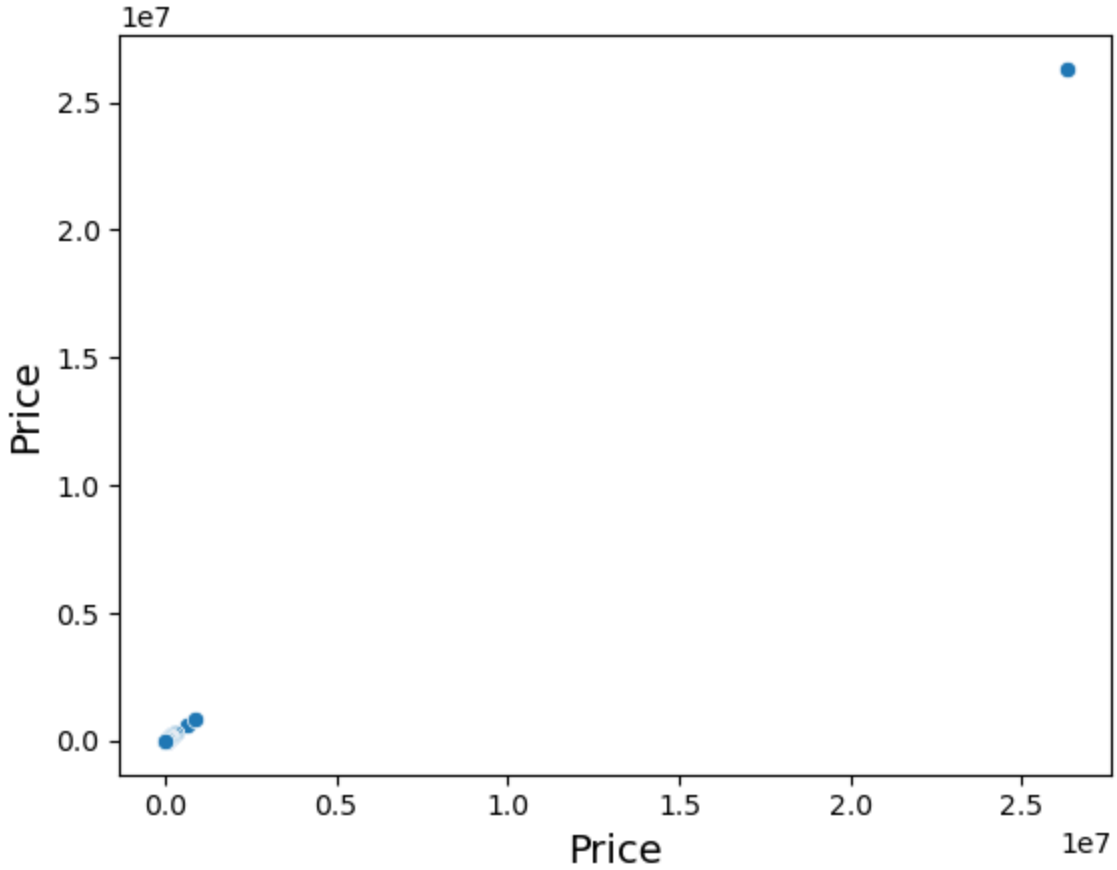
In [19]:
```
sns.pairplot(car_attributes)
```

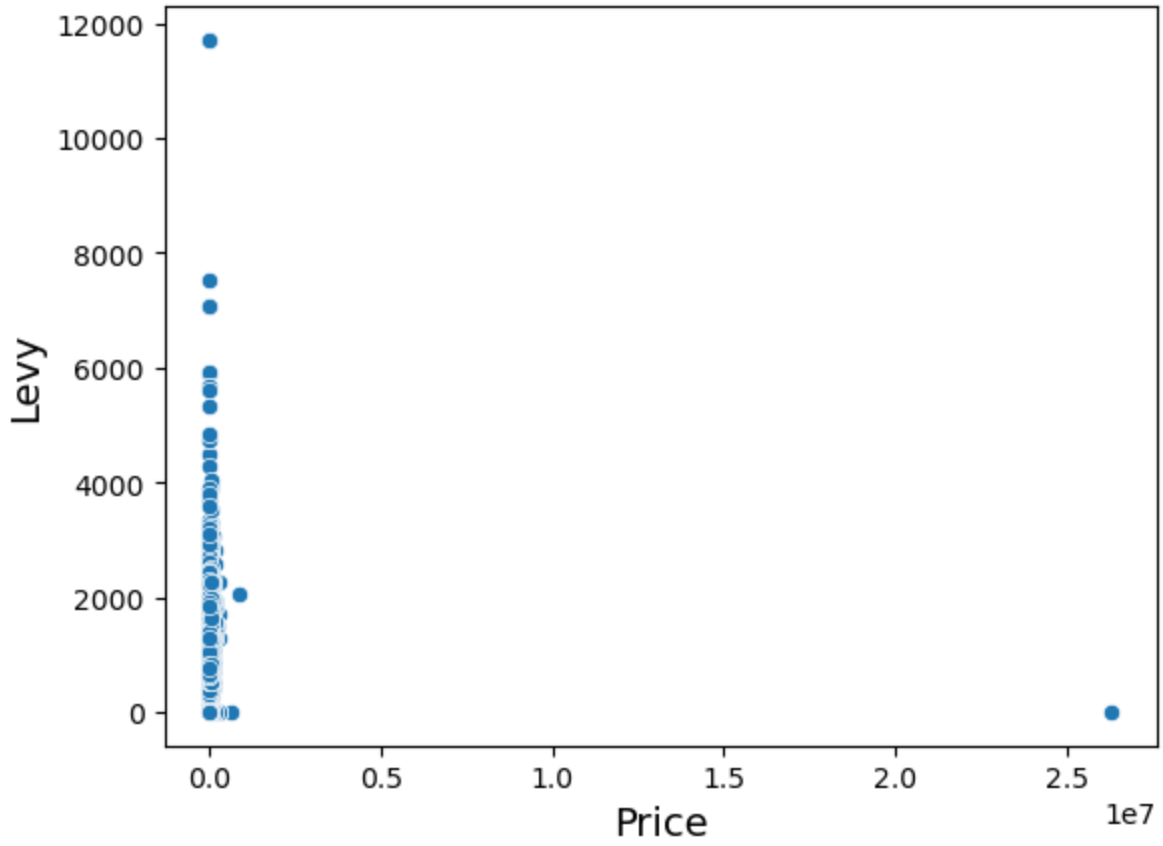
Out[19]:
```
<seaborn.axisgrid.PairGrid at 0x163b66050>
```

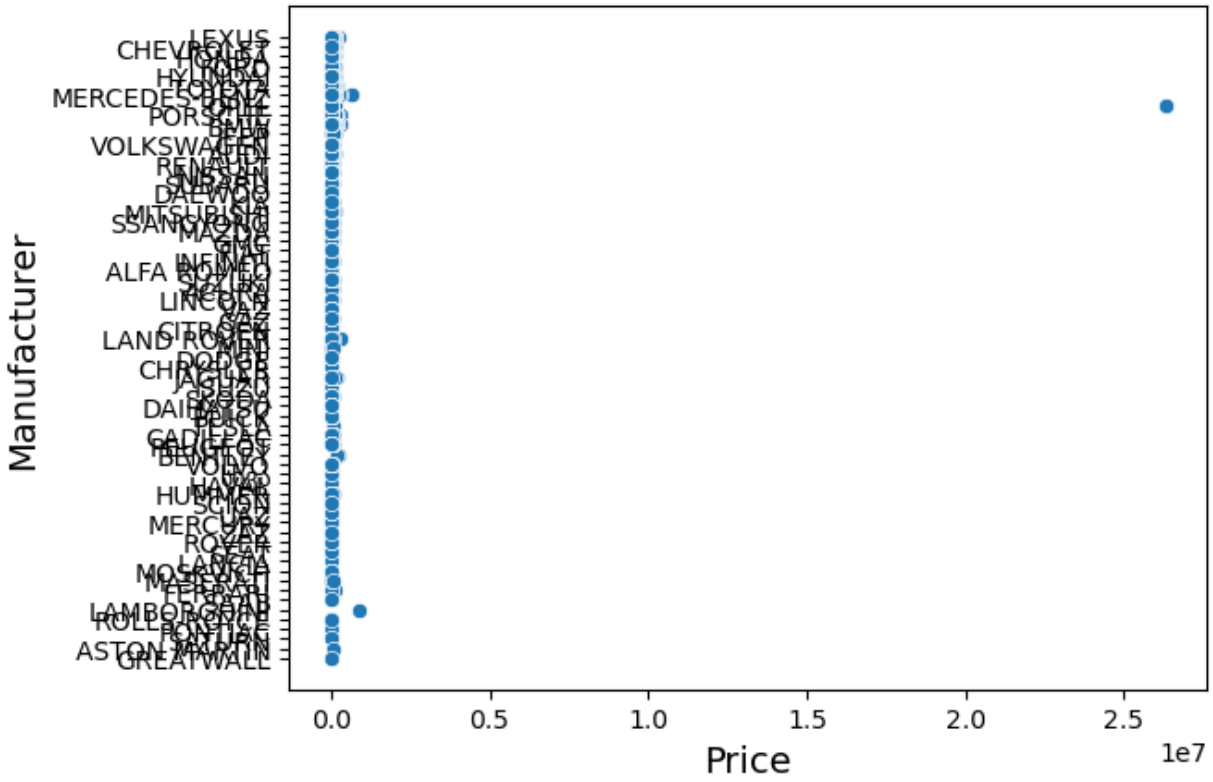


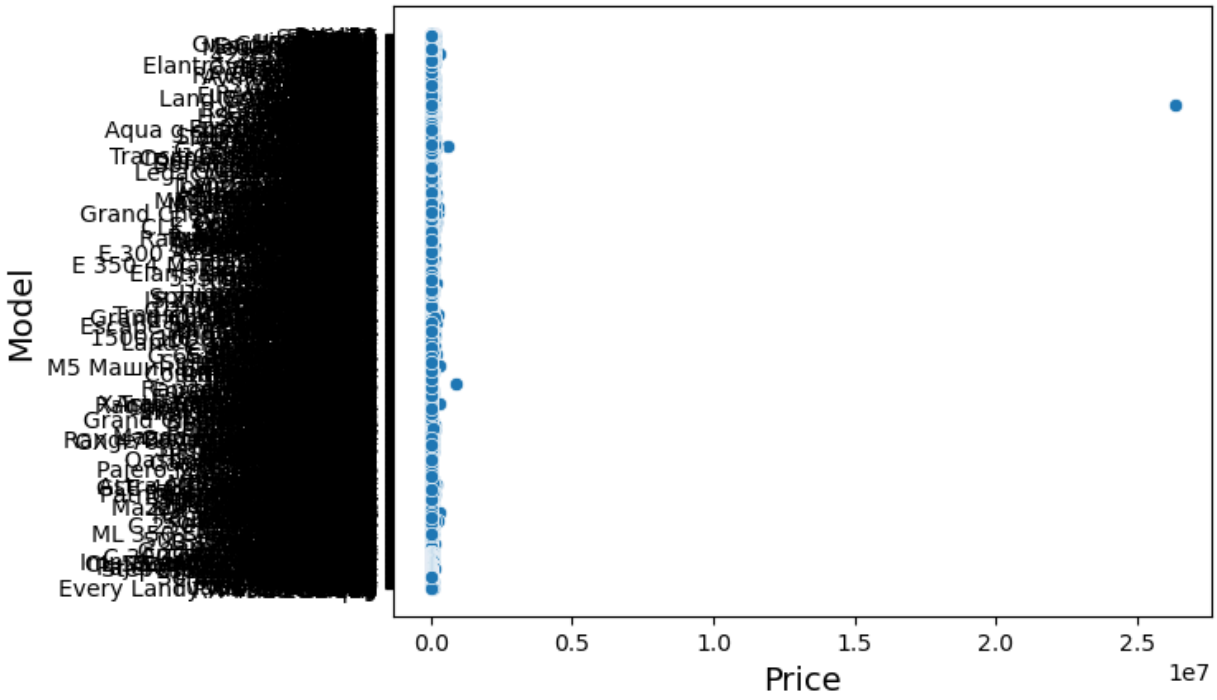
In [20]:
```
for colname in car_attributes:
  sns.scatterplot(data=car, x="Price", y=colname)
  plt.show()
```

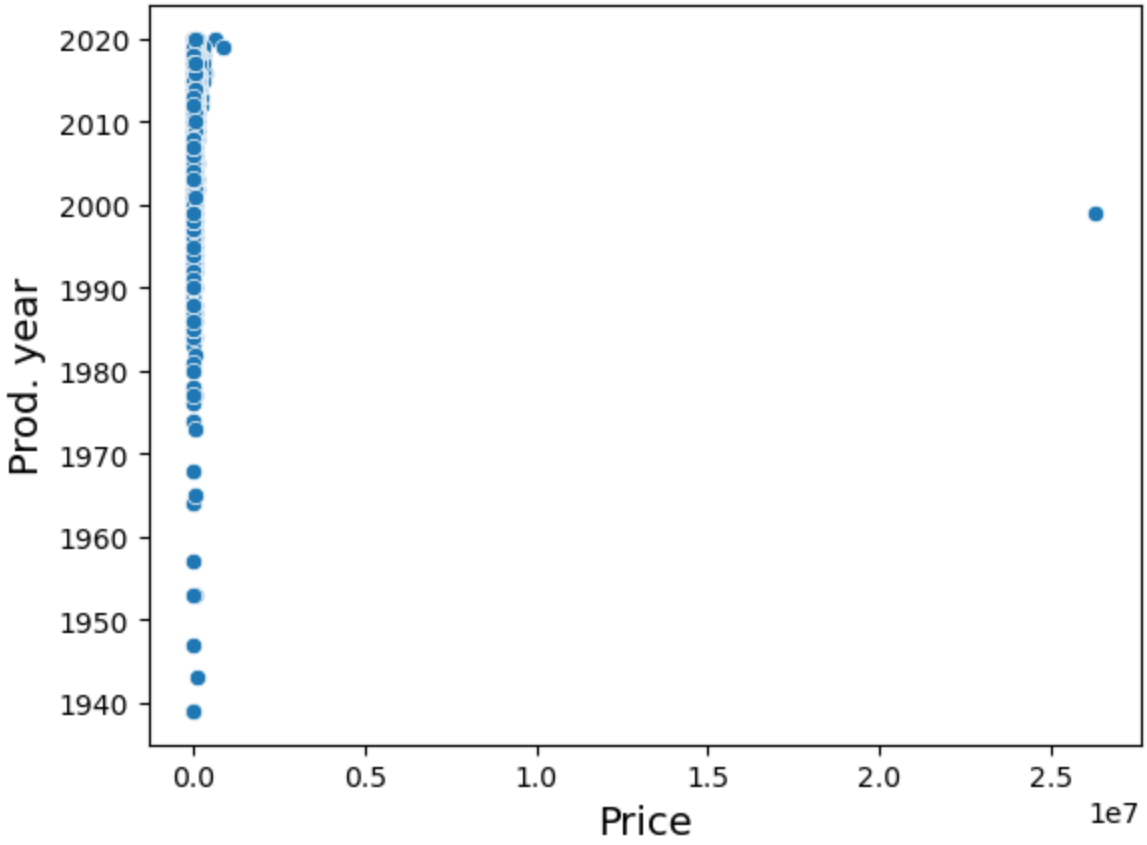

1e7
4.5
4.0
3.5
3.0
2.5
2.0
ID
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Price

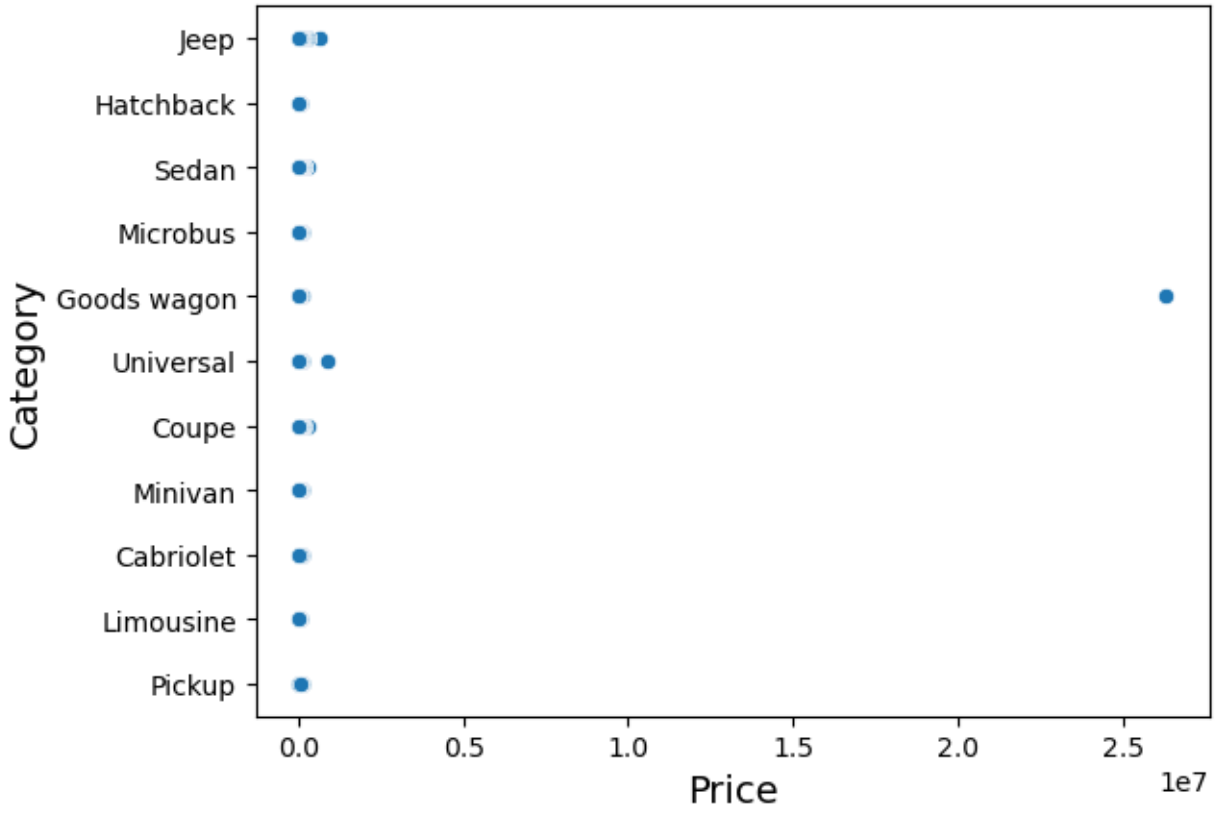

1e7
2.5
2.0
1.5
1.0
0.5
0.0
Price
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Price

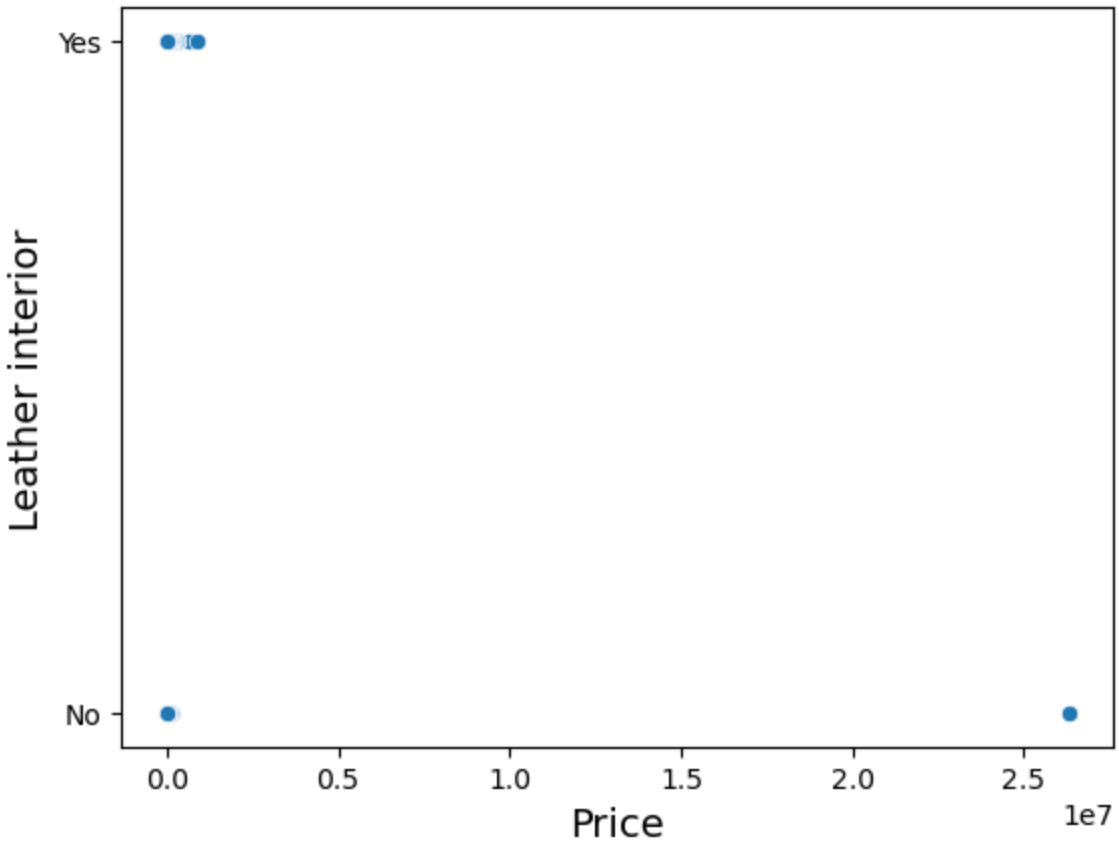

12000
10000
8000
6000
4000
2000
0
Levy
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Price

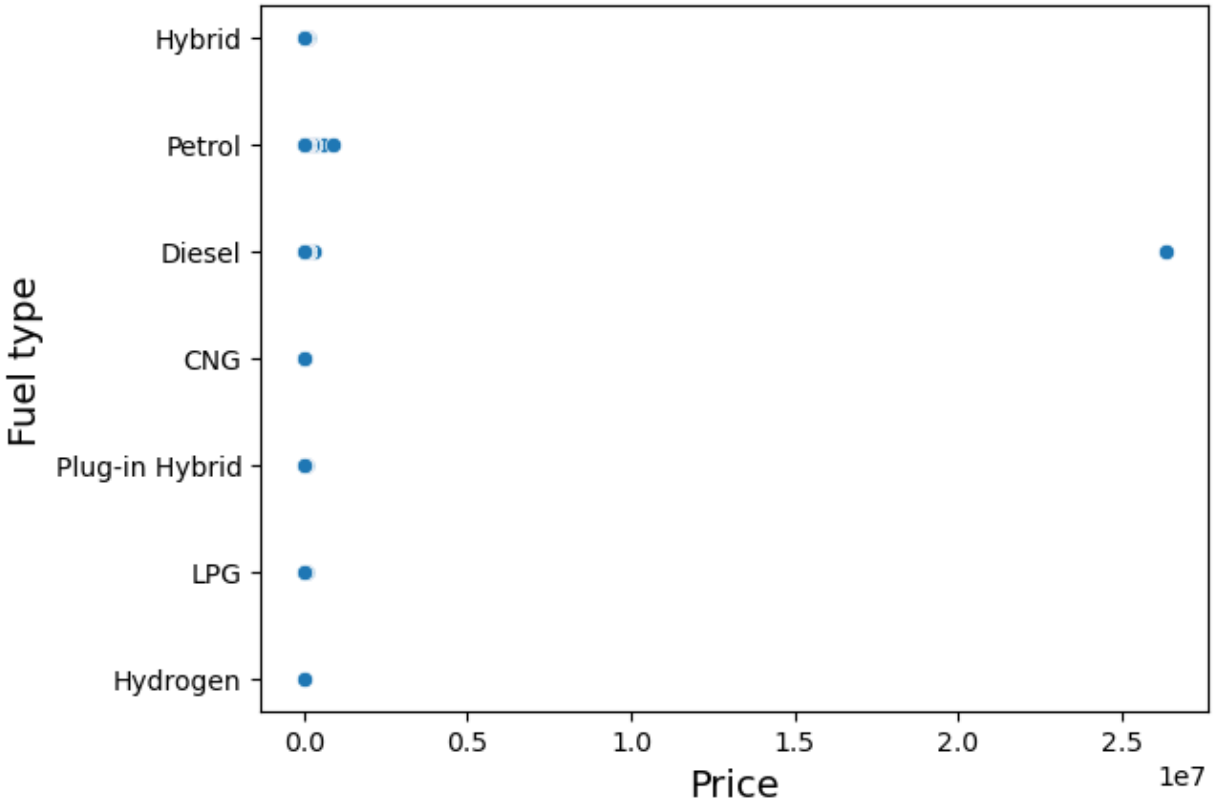

Manufacturer
LEXUS
CHEVROLET
HONDA
HYUNDAI
TOYOTA
MERCEDES-BENZ
PORSCHE
BMW
VOLKSWAGEN
RENAULT
SUBARU
NISSAN
DAEWOO
MITSUBISHI
SSANGYONG
MAZDA
GMC
FIAT
INFINITI
ALFA ROMEO
SUZUKI
HAVAL
LINCOLN
VAZ
CITROEN
LAND ROVER
MINI
DODGE
CHRYSLER
JAGUAR
SEAT
DAIHATSU
PEUGEOT
BUICK
GAZ
CADILLAC
TESLA
SKODA
VOLVO
HTC
HUMMER
SCION
MERCURY
SAAB
ROVER
LANCIA
MOSKVICH
MASERATI
FERRARI
LAMBORGHINI
ROLLS-ROYCE
BENTLEY
ASTON MARTIN
GREATWALL
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Price

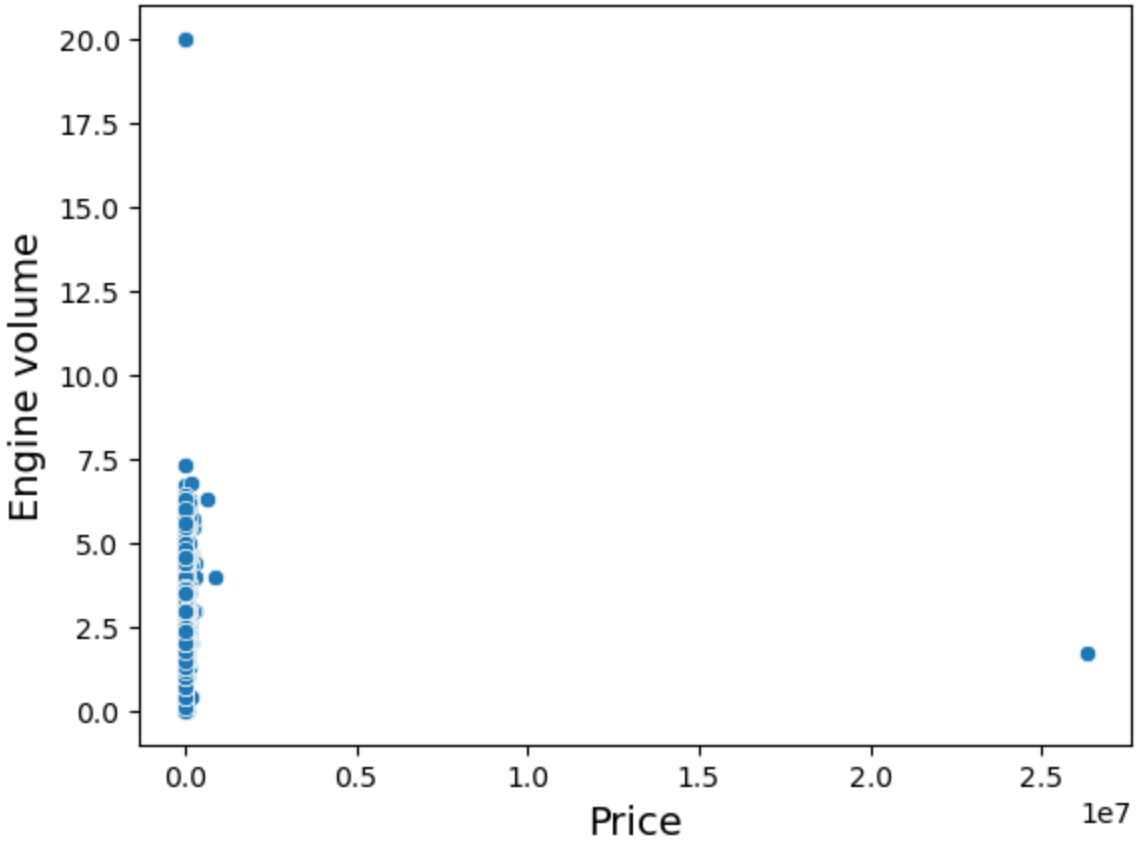

Model
Price
0.0
0.5
1.0
1.5
2.0
2.5
1e7

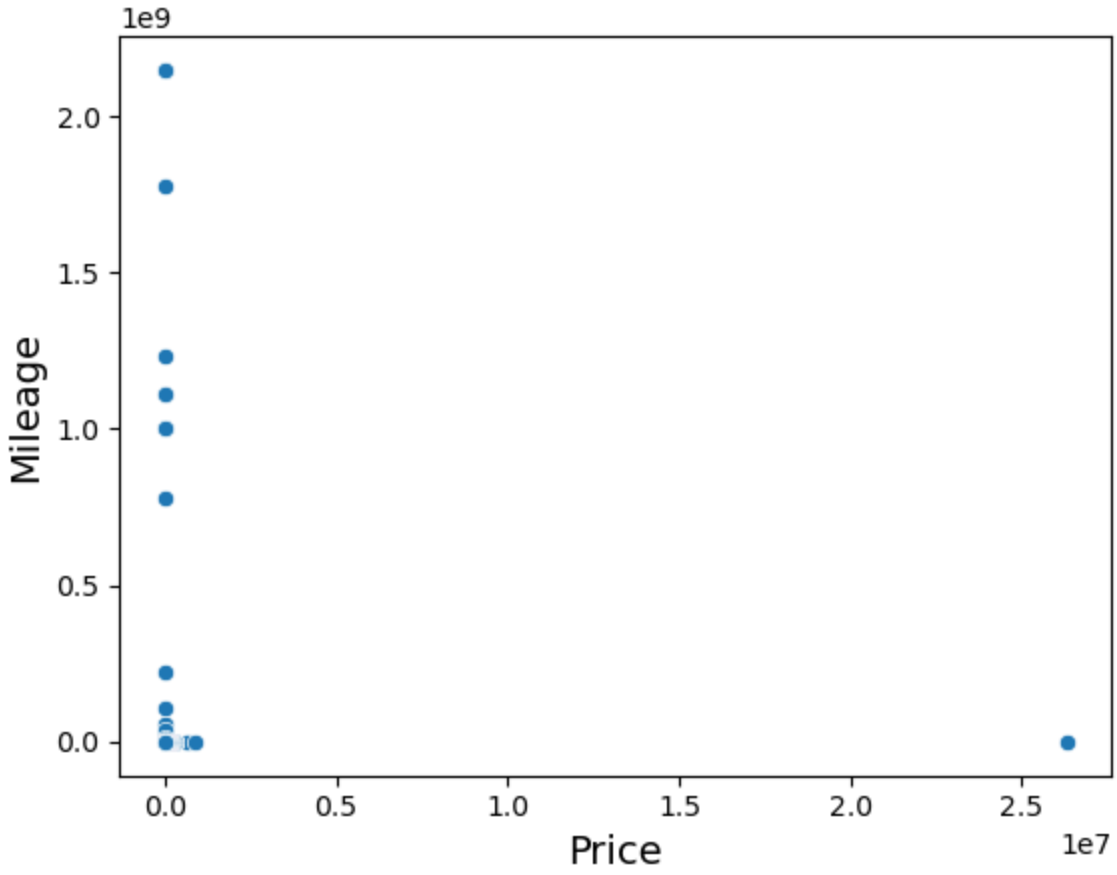

Prod. year
2020
2010
2000
1990
1980
1970
1960
1950
1940
Price
0.0
0.5
1.0
1.5
2.0
2.5
1e7

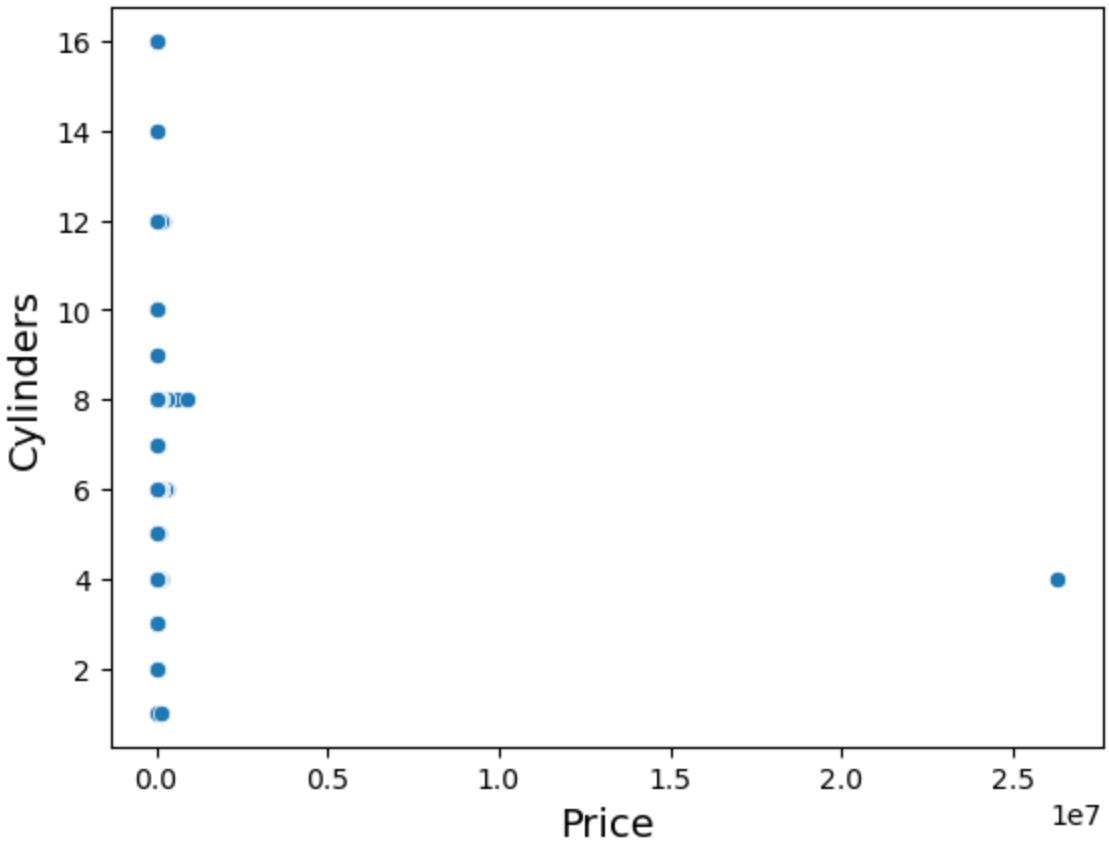

Jeep
Hatchback
Sedan
Microbus
Goods wagon
Universal
Coupe
Minivan
Cabriolet
Limousine
Pickup
Category
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Price

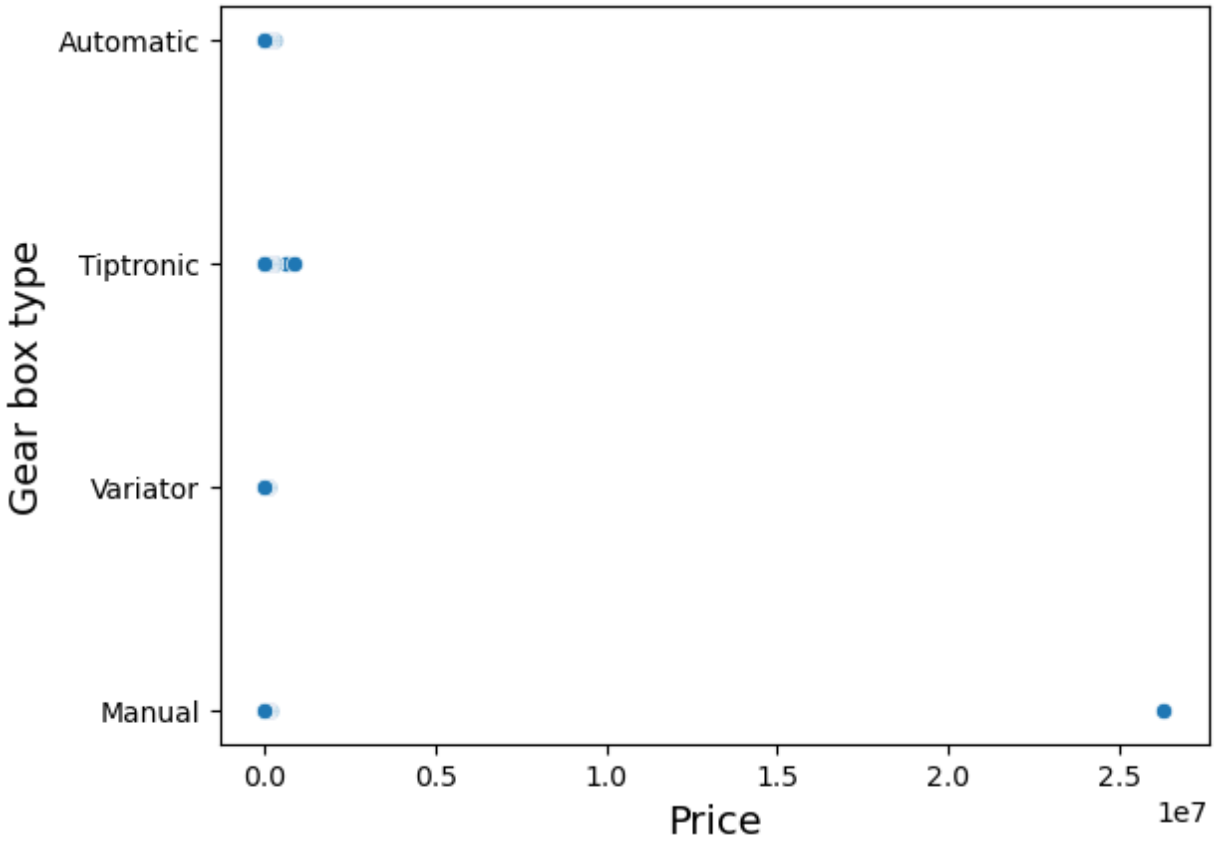

Yes
No
Leather interior
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Price

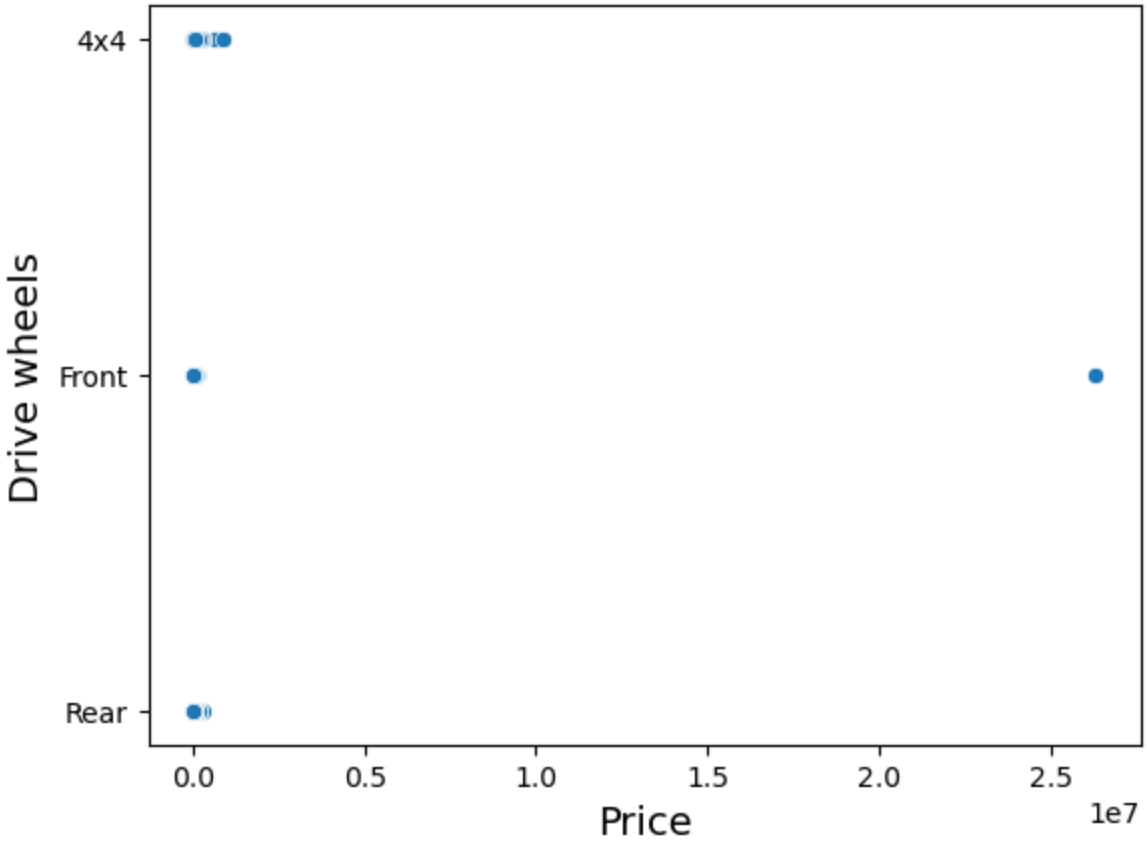

Hybrid
Petrol
Diesel
CNG
Plug-in Hybrid
LPG
Hydrogen
Fuel type
0.0
0.5
1.0
1.5
2.0
2.5
Price
1e7

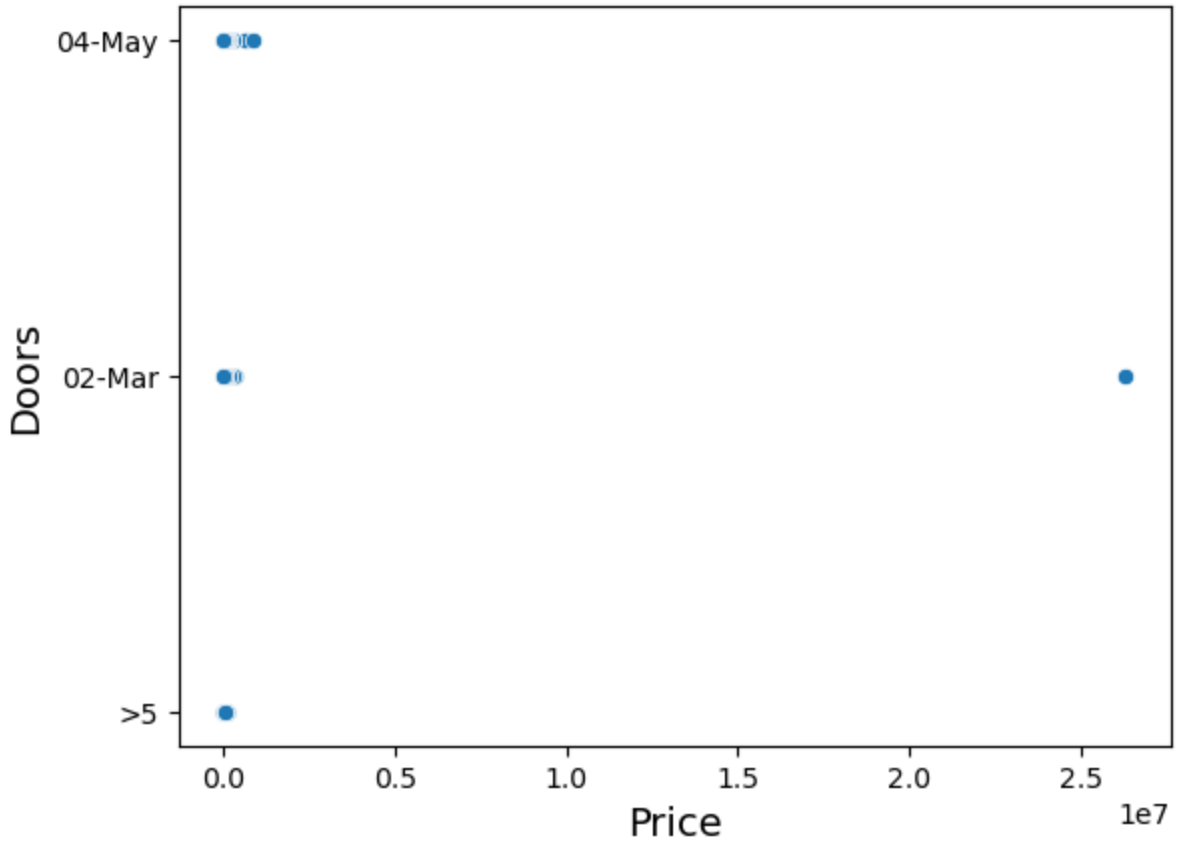

20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Engine volume
0.0
0.5
1.0
1.5
2.0
2.5
Price
1e7

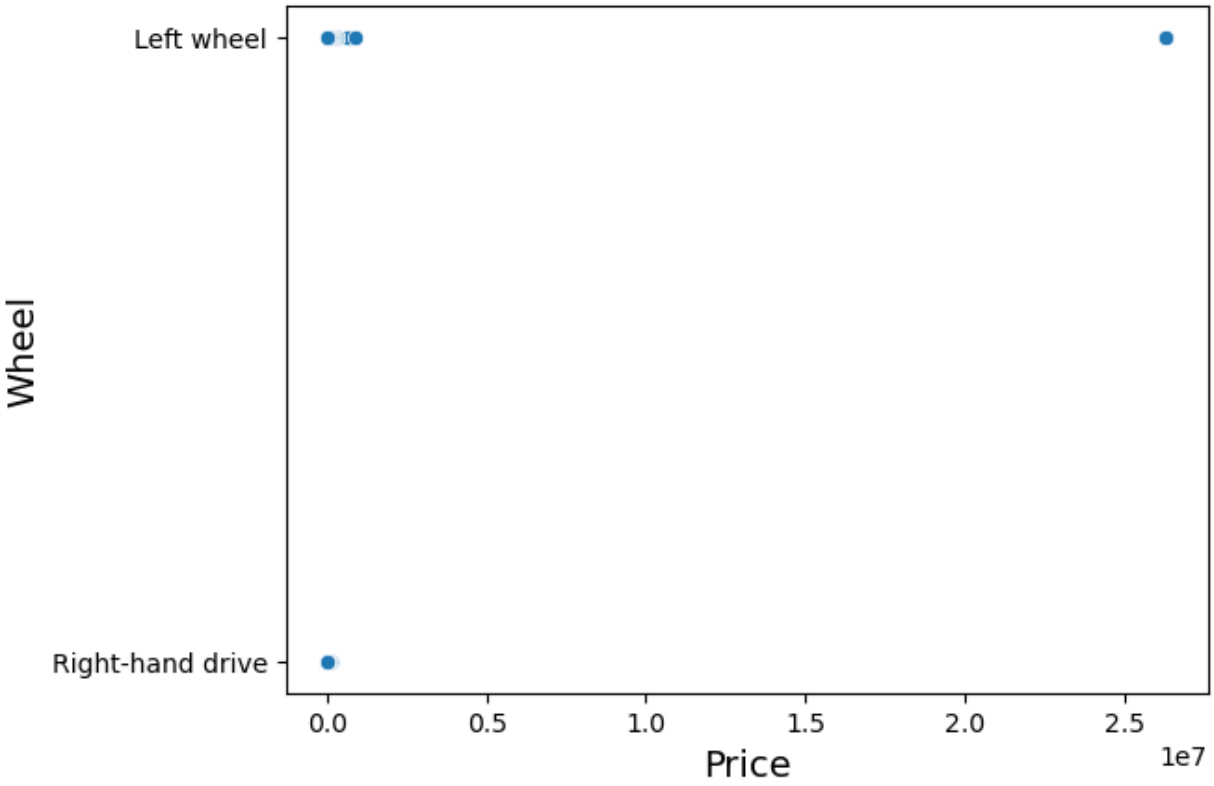

1e9
2.0
1.5
1.0
0.5
0.0
Mileage
0.0
0.5
1.0
1.5
2.0
2.5
Price
1e7

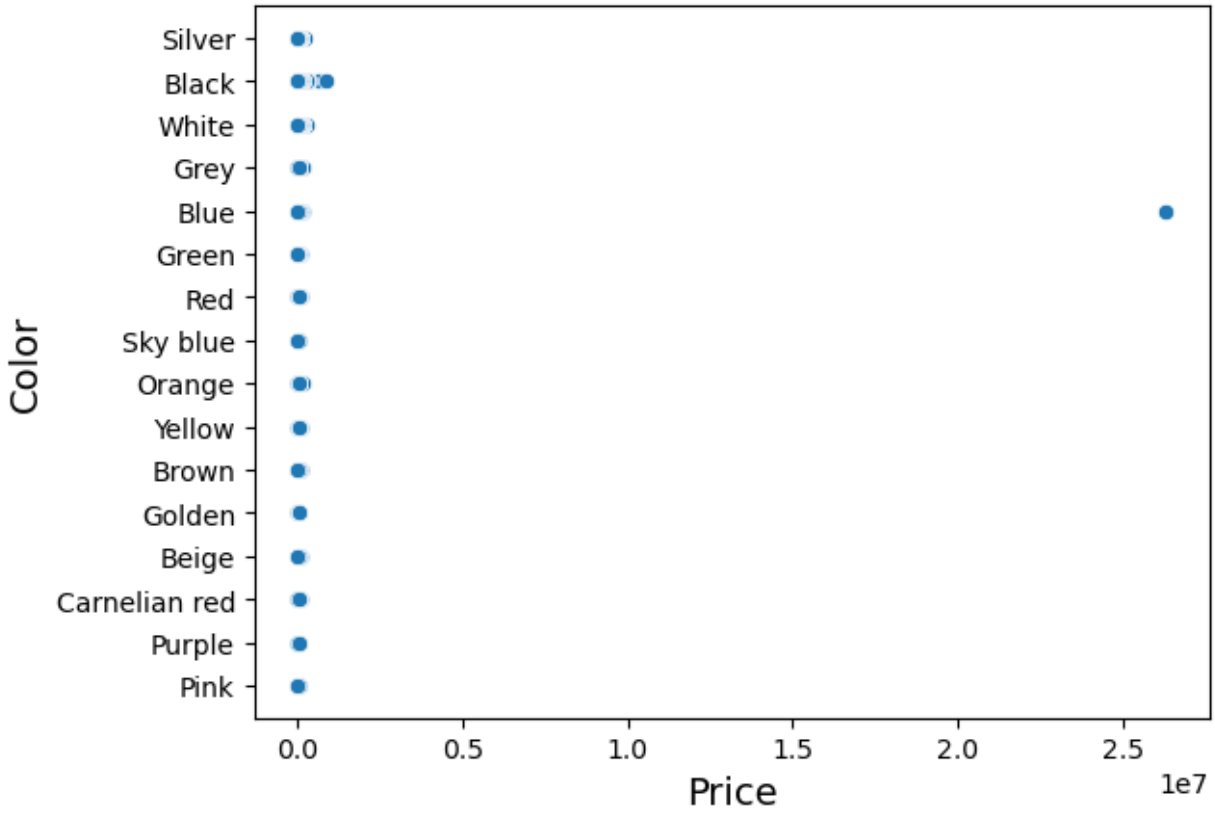

16
14
12
10
8
6
4
2
Cylinders
0.0
0.5
1.0
1.5
2.0
2.5
Price
1e7

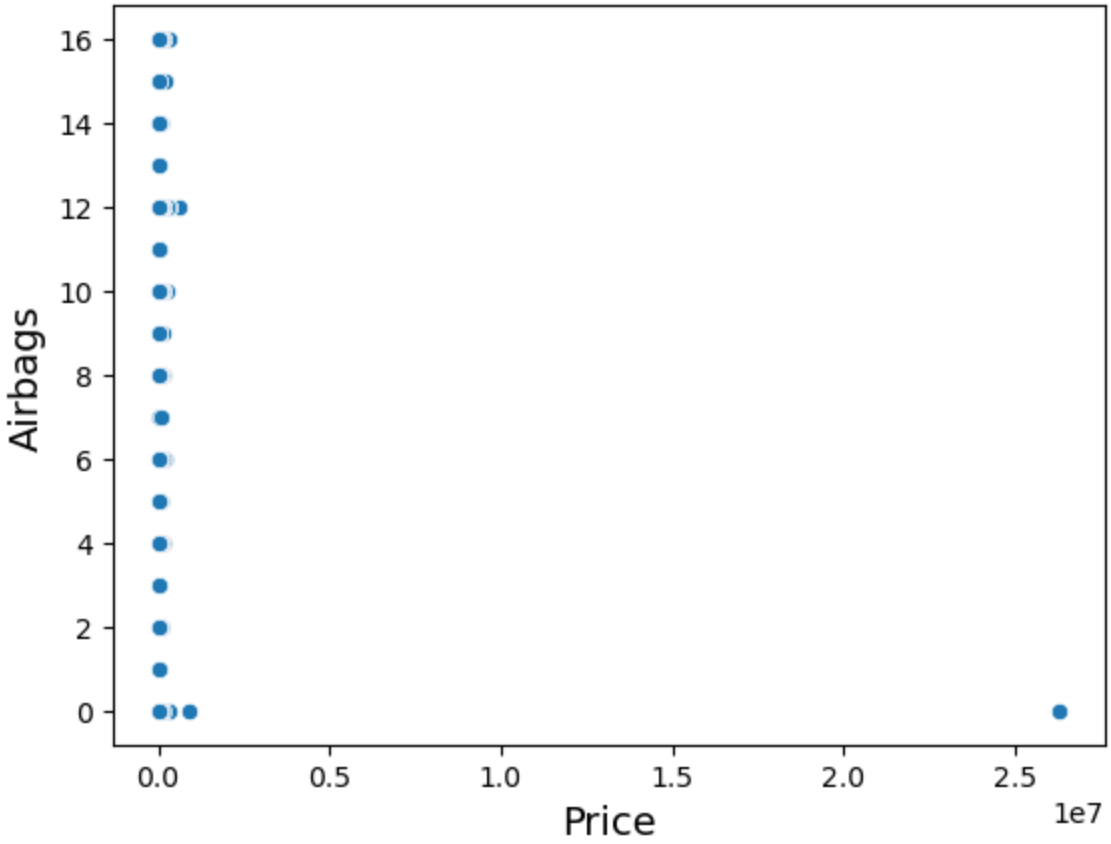

Automatic
Tiptronic
Variator
Manual
Gear box type
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Price


4x4
Front
Rear
Drive wheels
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Price


04-May
02-Mar
>5
Doors
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Price


Left wheel
Right-hand drive
Wheel
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Price





While there the initial correlation matrix reports minimal results, it is likely that the data needs to be treated and the outliers would have to be removed or analyzed before a model can be fitted. Hence, we will re-analyze the correlation matrix once we have conducted feature engineering and analyzed the categorical variables.

Before we can select the data for train the model, we would have to perform certain steps to train the model.

Step 1: Treating the outliers: We would use the IQR methodology to deal with the outliers

Ref: https://www.scaler.com/topics/data-science/handling-outliers-in-data-science/

In [21]:
```
from collections import Counter

def detect_outliers(df, n, car_featureslist):
    outlier_indices = []
    for i in car_featureslist:
        Q1 = np.percentile(df[i], 25)
        Q3 = np.percentile(df[i], 75)
        IQR = Q3 - Q1
        outlier_step = 1.5 * IQR
        outlier_list_col = df[(df[i] < Q1 - outlier_step) | (df[i] > Q3 + outli
        outlier_indices.extend(outlier_list_col)
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list(key for key, value in outlier_indices.items() if v
    return multiple_outliers

drop_outliers = detect_outliers(car, 2, ['Price', 'Levy', 'Prod. year', 'Mileag
print("There are the {} indices that will be dropped: ".format(len(drop_outlier

#Ref: https://stackoverflow.com/questions/50461349/how-to-remove-outlier-from-d
```

```
There are the 383 indices that will be dropped:  [90, 211, 420, 483, 573, 579,
723, 724, 747, 1019, 1083, 1128, 1225, 1364, 1459, 1490, 1509, 1510, 1562, 166
2, 1704, 1823, 2010, 2054, 2283, 2366, 2726, 2768, 2799, 2859, 2912, 2922, 294
1, 2942, 2965, 3101, 3365, 3487, 3684, 3686, 3705, 3707, 3765, 3905, 4068, 418
3, 4237, 4294, 4351, 4465, 4629, 4649, 4661, 4662, 4705, 4709, 4722, 4919, 500
8, 5259, 5412, 5485, 5541, 5718, 5731, 5940, 6335, 6405, 6468, 6768, 6826, 687
3, 6887, 6950, 7010, 7031, 7094, 7147, 7247, 7254, 7283, 7318, 7353, 7565, 762
1, 7667, 7675, 7747, 7749, 7760, 7970, 7997, 8036, 8147, 8246, 8541, 8644, 875
5, 8880, 9101, 9172, 9212, 9233, 9247, 9259, 9327, 9367, 9405, 9441, 9452, 968
8, 9739, 10082, 10085, 10220, 10468, 10520, 10690, 10710, 10714, 10759, 10948,
11035, 11038, 11041, 11131, 11138, 11210, 11529, 11827, 11906, 11941, 11973, 1
2081, 12152, 12224, 12750, 12848, 12905, 12981, 13088, 13320, 13325, 13351, 13
467, 13534, 13631, 13653, 13687, 13745, 13847, 13884, 13889, 13941, 13973, 141
93, 14356, 14435, 14523, 14752, 14763, 14871, 14876, 15267, 15405, 15623, 1565
9, 16279, 16418, 16432, 16525, 16534, 16548, 16614, 16718, 16926, 17005, 1708
5, 17153, 17167, 17287, 17471, 17506, 17508, 17527, 17728, 17760, 17868, 1787
0, 17930, 18144, 18559, 18620, 18640, 18720, 18885, 26, 78, 115, 492, 510, 70
1, 876, 930, 982, 1125, 1241, 1300, 1314, 1349, 1470, 1689, 2051, 2065, 2125,
2151, 2418, 2467, 2495, 2592, 2723, 3231, 3606, 3615, 4147, 4289, 4416, 4558,
4959, 5274, 5339, 5367, 5529, 5848, 5950, 6007, 6196, 6432, 6582, 6807, 6921,
7083, 7119, 7150, 7373, 7533, 7542, 7964, 8235, 8392, 8577, 9076, 9114, 9148,
9250, 9445, 9506, 9620, 9910, 10028, 10251, 10350, 10395, 10495, 10551, 10721,
10767, 10955, 11341, 11351, 11413, 11427, 11656, 11777, 11860, 11911, 12147, 1
2390, 12528, 12639, 13123, 13307, 14011, 14122, 14202, 14336, 14432, 14546, 14
616, 14676, 14726, 14892, 14945, 14979, 15114, 15330, 16119, 16297, 16322, 166
95, 16799, 16885, 16895, 17003, 17135, 17188, 18543, 18769, 18854, 18957, 1896
3, 18984, 19048, 11, 169, 228, 400, 429, 508, 511, 1285, 1588, 1710, 1735, 180
6, 1953, 2732, 2810, 2873, 2952, 2963, 3605, 3698, 3817, 4174, 4371, 4411, 486
6, 4994, 5509, 5696, 6005, 6703, 7045, 7599, 7746, 8590, 9145, 9425, 9644, 972
1, 9994, 10265, 10275, 10281, 10503, 10575, 10869, 10960, 11266, 11275, 11587,
11744, 11764, 12288, 12444, 13094, 15247, 15347, 15616, 15839, 17446, 17634, 1
9199, 428, 1086, 2512, 5860, 6513, 7272, 10973, 13850, 15665]
```

In [22]:
```
car.iloc[drop_outliers, :]
```

Out[22]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Er vo
90	45807330	77775	1604	MERCEDES-BENZ	GL 63 AMG	2014	Jeep	Yes	Petrol	
211	45156280	72130	1885	PORSCHE	Panamera	2010	Hatchback	Yes	Petrol	
420	45763904	81539	1935	LEXUS	GX 460	2016	Jeep	Yes	Petrol	
483	45761340	69935	1646	LEXUS	GX 470	2015	Jeep	Yes	Petrol	
573	45731517	119172	1301	BMW	M6	2014	Coupe	Yes	Petrol	
...	...	...	...	...	...	...	...	...	...	...
6513	45756959	27284	0	TOYOTA	Land Cruiser	2004	Jeep	Yes	Diesel	
7272	45416515	35438	0	BMW	X6	2009	Jeep	Yes	Petrol	
10973	45416515	35438	0	BMW	X6	2009	Jeep	Yes	Petrol	
13850	45796827	1000	0	MERCEDES-BENZ	GLS 63 AMG	2014	Sedan	Yes	Petrol	
15665	45806588	706	1086	DODGE	Avenger	2012	Sedan	Yes	Petrol	

383 rows × 18 columns

In [23]:
```
car = car.drop(drop_outliers, axis = 0).reset_index(drop = True) #dropping the
```

In [24]:
```
car
```

Out[24]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Eng volu
0	45654403	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Hybrid	
1	44731507	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petrol	
2	45774419	8467	0	HONDA	FIT	2006	Hatchback	No	Petrol	
3	45769185	3607	862	FORD	Escape	2011	Jeep	Yes	Hybrid	
4	45809263	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petrol	
...	...	...	...	...	...	...	...	...	...	...
18849	45798355	8467	0	MERCEDES-BENZ	CLK 200	1999	Coupe	Yes	CNG	
18850	45778856	15681	831	HYUNDAI	Sonata	2011	Sedan	Yes	Petrol	
18851	45804997	26108	836	HYUNDAI	Tucson	2010	Jeep	Yes	Diesel	
18852	45793526	5331	1288	CHEVROLET	Captiva	2007	Jeep	Yes	Diesel	
18853	45813273	470	753	HYUNDAI	Sonata	2012	Sedan	Yes	Hybrid	

18854 rows × 18 columns

In [25]:
```
car['Levy'] = car['Levy'].replace(0, np.nan)
levy_index = list(~car['Levy'].isnull())
levy_median = np.median(car['Levy'].loc[levy_index])
levy_median
```

Out[25]:
```
781.0
```

In [26]:
```
car['Levy'].fillna(levy_median, inplace = True)
```

In [27]:
```
meanprice_bycategory = car[['Category', 'Price']].groupby('Category', as_index

# Group by 'Mean_Price' and collect 'Category' values into class_1 and class_2
Category_1 = meanprice_bycategory.loc[meanprice_bycategory['Price'] <= 20000,
Category_2 = meanprice_bycategory.loc[meanprice_bycategory['Price'] > 20000, '

print('Categories with less than or equal to 20000 mean price:', Category_1)
print('Categories with more than 20000 mean price:', Category_2)
```

```
Categories with less than or equal to 20000 mean price: ['Coupe', 'Hatchback',
'Limousine', 'Microbus', 'Sedan']
Categories with more than 20000 mean price: ['Cabriolet', 'Goods wagon', 'Jee
p', 'Minivan', 'Pickup', 'Universal']
```

In [28]:
```
category_data = car['Category']
category_data_new = []

for value in category_data:
    if value in Category_1:
        category_data_new.append(1)
    else:
        category_data_new.append(2)

car['Category'] = category_data_new
```

In [29]:
```
car
```

Out[29]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Eng volu
0	45654403	13328	1399.0	LEXUS	RX 450	2010	2	Yes	Hybrid	
1	44731507	16621	1018.0	CHEVROLET	Equinox	2011	2	No	Petrol	
2	45774419	8467	781.0	HONDA	FIT	2006	1	No	Petrol	
3	45769185	3607	862.0	FORD	Escape	2011	2	Yes	Hybrid	
4	45809263	11726	446.0	HONDA	FIT	2014	1	Yes	Petrol	
...	...	...	...	...	...	...	...	...	...	...
18849	45798355	8467	781.0	MERCEDES-BENZ	CLK 200	1999	1	Yes	CNG	
18850	45778856	15681	831.0	HYUNDAI	Sonata	2011	1	Yes	Petrol	
18851	45804997	26108	836.0	HYUNDAI	Tucson	2010	2	Yes	Diesel	
18852	45793526	5331	1288.0	CHEVROLET	Captiva	2007	2	Yes	Diesel	
18853	45813273	470	753.0	HYUNDAI	Sonata	2012	1	Yes	Hybrid	

18854 rows × 18 columns

In [30]:
```
#Referencing the output we received from the value_counts code, we can combine
fuel_typedata = car['Fuel type']

new_fueltypedata = ['other' if value in {'Hybrid', 'Hydrogen', 'Plug-in Hybrid
unique_values = set(new_fueltypedata)

car['Fuel type'] = new_fueltypedata

car
```

Out[30]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Eng volu
0	45654403	13328	1399.0	LEXUS	RX 450	2010	2	Yes	other	
1	44731507	16621	1018.0	CHEVROLET	Equinox	2011	2	No	Petrol	
2	45774419	8467	781.0	HONDA	FIT	2006	1	No	Petrol	
3	45769185	3607	862.0	FORD	Escape	2011	2	Yes	other	
4	45809263	11726	446.0	HONDA	FIT	2014	1	Yes	Petrol	
...	...	...	...	...	...	...	...	...	...	
18849	45798355	8467	781.0	MERCEDES-BENZ	CLK 200	1999	1	Yes	CNG	
18850	45778856	15681	831.0	HYUNDAI	Sonata	2011	1	Yes	Petrol	
18851	45804997	26108	836.0	HYUNDAI	Tucson	2010	2	Yes	Diesel	
18852	45793526	5331	1288.0	CHEVROLET	Captiva	2007	2	Yes	Diesel	
18853	45813273	470	753.0	HYUNDAI	Sonata	2012	1	Yes	other	

18854 rows × 18 columns

Further we would also combine the Automatic and Variator into one category and the other two gear types into another category

Ref: https://idaoffice.org/posts/variator-what-kind-of-thing-is-this/#:~:text=Gears%2C%20between%20which%20the%20variator,has%20a%20lot%20of%2(

In [31]:
```
gear_data = car['Gear box type']

new_geardata = [1 if value in {'Automatic', 'Variator'} else 2 for value in gea
unique_values = set(new_geardata)

car['Gear box type'] = new_geardata

car
```

Out[31]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Eng volu
0	45654403	13328	1399.0	LEXUS	RX 450	2010	2	Yes	other	
1	44731507	16621	1018.0	CHEVROLET	Equinox	2011	2	No	Petrol	
2	45774419	8467	781.0	HONDA	FIT	2006	1	No	Petrol	
3	45769185	3607	862.0	FORD	Escape	2011	2	Yes	other	
4	45809263	11726	446.0	HONDA	FIT	2014	1	Yes	Petrol	
...	...	...	...	...	...	...	...	...	...	
18849	45798355	8467	781.0	MERCEDES-BENZ	CLK 200	1999	1	Yes	CNG	
18850	45778856	15681	831.0	HYUNDAI	Sonata	2011	1	Yes	Petrol	
18851	45804997	26108	836.0	HYUNDAI	Tucson	2010	2	Yes	Diesel	
18852	45793526	5331	1288.0	CHEVROLET	Captiva	2007	2	Yes	Diesel	
18853	45813273	470	753.0	HYUNDAI	Sonata	2012	1	Yes	other	

18854 rows × 18 columns

Referencing the output we received from the value_counts code, we we would have to clean the data in the doors column to replace values such as 4-May and 2-Mar

In [32]:
```
doors_typedata = car['Doors']

new_doortypedata = ['2-3' if value in {'02-Mar'} else '4-5' if value == '04-May
unique_values = set(new_doortypedata)

car['Doors'] = new_doortypedata

car
```

Out[32]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Eng volu
0	45654403	13328	1399.0	LEXUS	RX 450	2010	2	Yes	other	
1	44731507	16621	1018.0	CHEVROLET	Equinox	2011	2	No	Petrol	
2	45774419	8467	781.0	HONDA	FIT	2006	1	No	Petrol	
3	45769185	3607	862.0	FORD	Escape	2011	2	Yes	other	
4	45809263	11726	446.0	HONDA	FIT	2014	1	Yes	Petrol	
...	...	...	...	...	...	...	...	...	...	
18849	45798355	8467	781.0	MERCEDES-BENZ	CLK 200	1999	1	Yes	CNG	
18850	45778856	15681	831.0	HYUNDAI	Sonata	2011	1	Yes	Petrol	
18851	45804997	26108	836.0	HYUNDAI	Tucson	2010	2	Yes	Diesel	
18852	45793526	5331	1288.0	CHEVROLET	Captiva	2007	2	Yes	Diesel	
18853	45813273	470	753.0	HYUNDAI	Sonata	2012	1	Yes	other	

18854 rows × 18 columns

Further converting the Production year to Age data so that the attribute can be better analyzed

In [33]:
```
prodyear_data = car['Prod. year']
new_age_data = []

for value in prodyear_data:
    new_age_data.append(2023 - value)

car['Age'] = new_age_data
```

In [34]:
```
car.drop(['Prod. year'], axis = 1, inplace = True)

car
```

Out[34]:

	ID	Price	Levy	Manufacturer	Model	Category	Leather interior	Fuel type	Engine volume	M
0	45654403	13328	1399.0	LEXUS	RX 450	2	Yes	other	3.5	18
1	44731507	16621	1018.0	CHEVROLET	Equinox	2	No	Petrol	3.0	19
2	45774419	8467	781.0	HONDA	FIT	1	No	Petrol	1.3	20
3	45769185	3607	862.0	FORD	Escape	2	Yes	other	2.5	16
4	45809263	11726	446.0	HONDA	FIT	1	Yes	Petrol	1.3	
...	...	...	...	...	...	...	...	...	...	
18849	45798355	8467	781.0	MERCEDES-BENZ	CLK 200	1	Yes	CNG	2.0	30
18850	45778856	15681	831.0	HYUNDAI	Sonata	1	Yes	Petrol	2.4	16
18851	45804997	26108	836.0	HYUNDAI	Tucson	2	Yes	Diesel	2.0	17
18852	45793526	5331	1288.0	CHEVROLET	Captiva	2	Yes	Diesel	2.0	5
18853	45813273	470	753.0	HYUNDAI	Sonata	1	Yes	other	2.4	18

18854 rows × 18 columns

In [35]:
```
from scipy import stats

columns_to_adjust = ['Levy', 'Age', 'Price', 'Mileage', 'Engine volume']

for column in columns_to_adjust:
    # For other columns, you can apply the transformation directly
    car[column] = [1 if value == 0 else value for value in car[column]]
    # Additional steps for the other columns
    if column in ['Levy', 'Age', 'Price', 'Mileage', 'Engine volume']:
        modified_column, _ = stats.boxcox(car[column])
        car[column] = modified_column
```

In order to apply feature scaling to the datset, the methodology applied here is box-cox transformation to correct for the skewness. Ref:(Working with skewed data) https://anshikaaxena.medium.com/how-skewed-data-can-skrew-your-linear-regression-model-accuracy-and-transfromation-can-help-62c6d3fe4c53

In [36]:
```
car
```

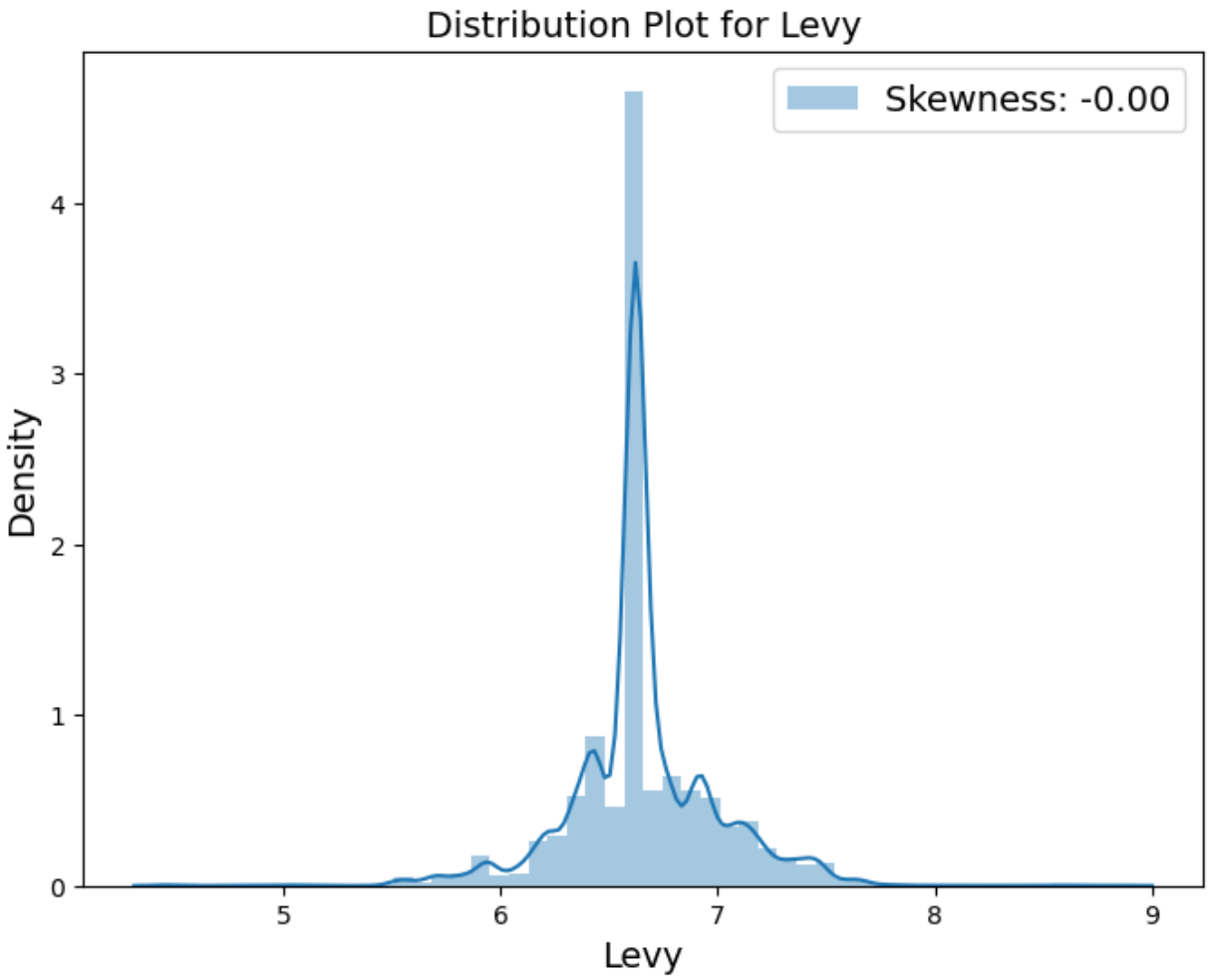
Out[36]:

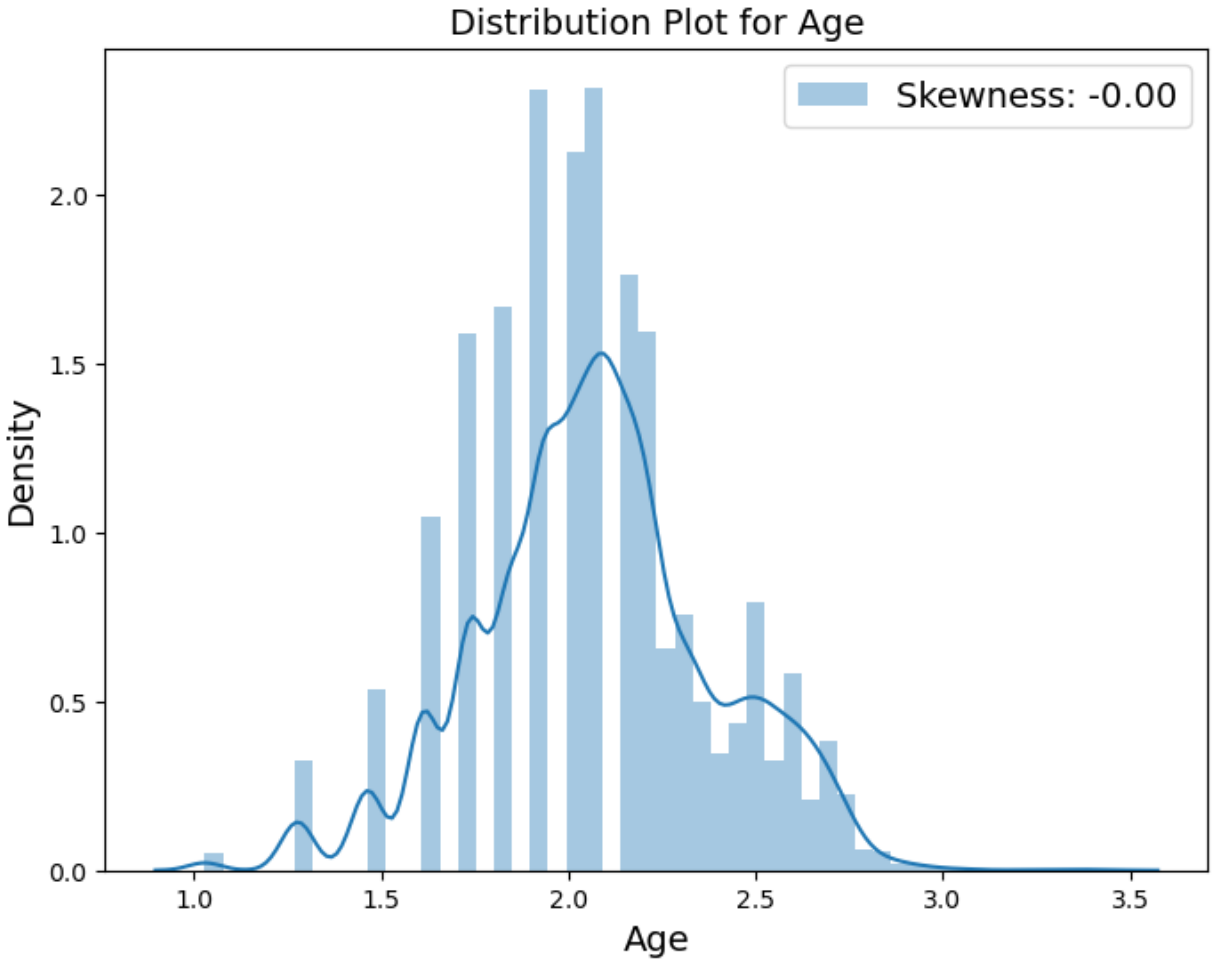
	ID	Price	Levy	Manufacturer	Model	Category	Leather interior	Fuel type	Engine vo
0	45654403	49.647560	7.201868	LEXUS	RX 450	2	Yes	other	1.29
1	44731507	53.121011	6.887520	CHEVROLET	Equinox	2	No	Petrol	1.13
2	45774419	43.164922	6.625353	HONDA	FIT	1	No	Petrol	0.26
3	45769185	33.038016	6.722984	FORD	Escape	2	Yes	other	0.93
4	45809263	47.731553	6.070764	HONDA	FIT	1	Yes	Petrol	0.26
...	...	...	...	...	...	...	...	...	
18849	45798355	43.164922	6.625353	MERCEDES-BENZ	CLK 200	1	Yes	CNG	0.70
18850	45778856	52.183728	6.686749	HYUNDAI	Sonata	1	Yes	Petrol	0.89
18851	45804997	60.948014	6.692684	HYUNDAI	Tucson	2	Yes	Diesel	0.70
18852	45793526	37.368233	7.120145	CHEVROLET	Captiva	2	Yes	Diesel	0.70
18853	45813273	16.872588	6.589227	HYUNDAI	Sonata	1	Yes	other	0.89

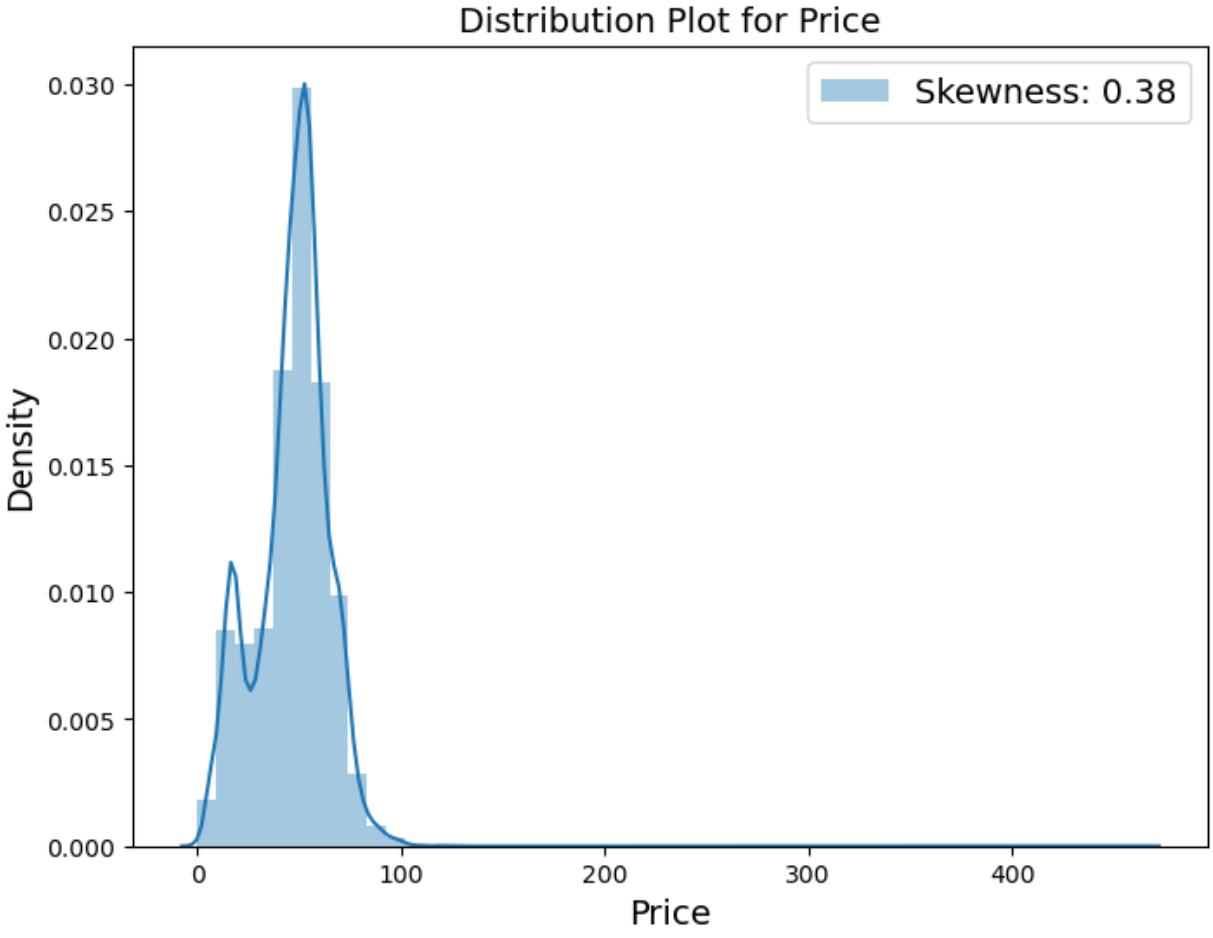
18854 rows × 18 columns

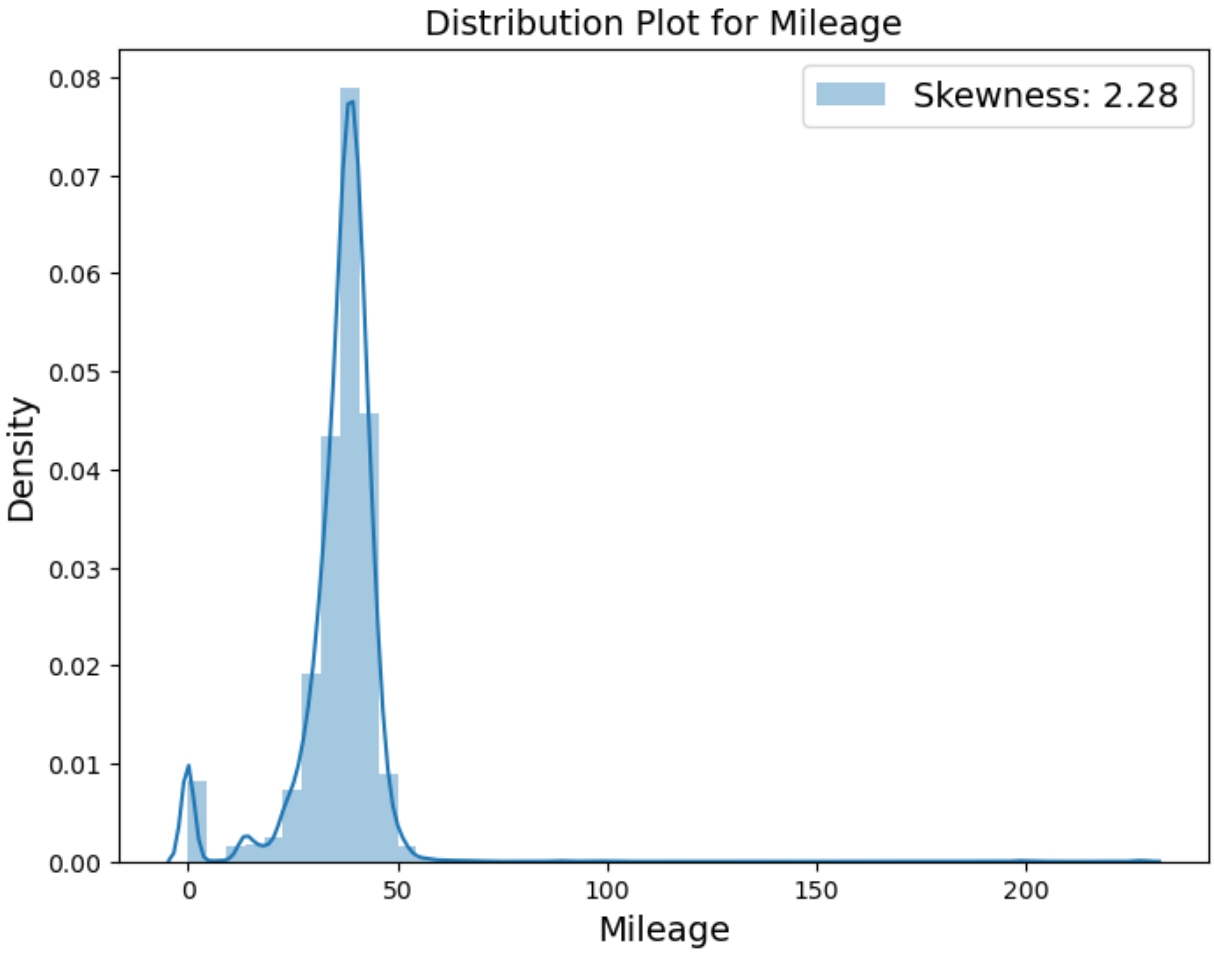
In [37]:

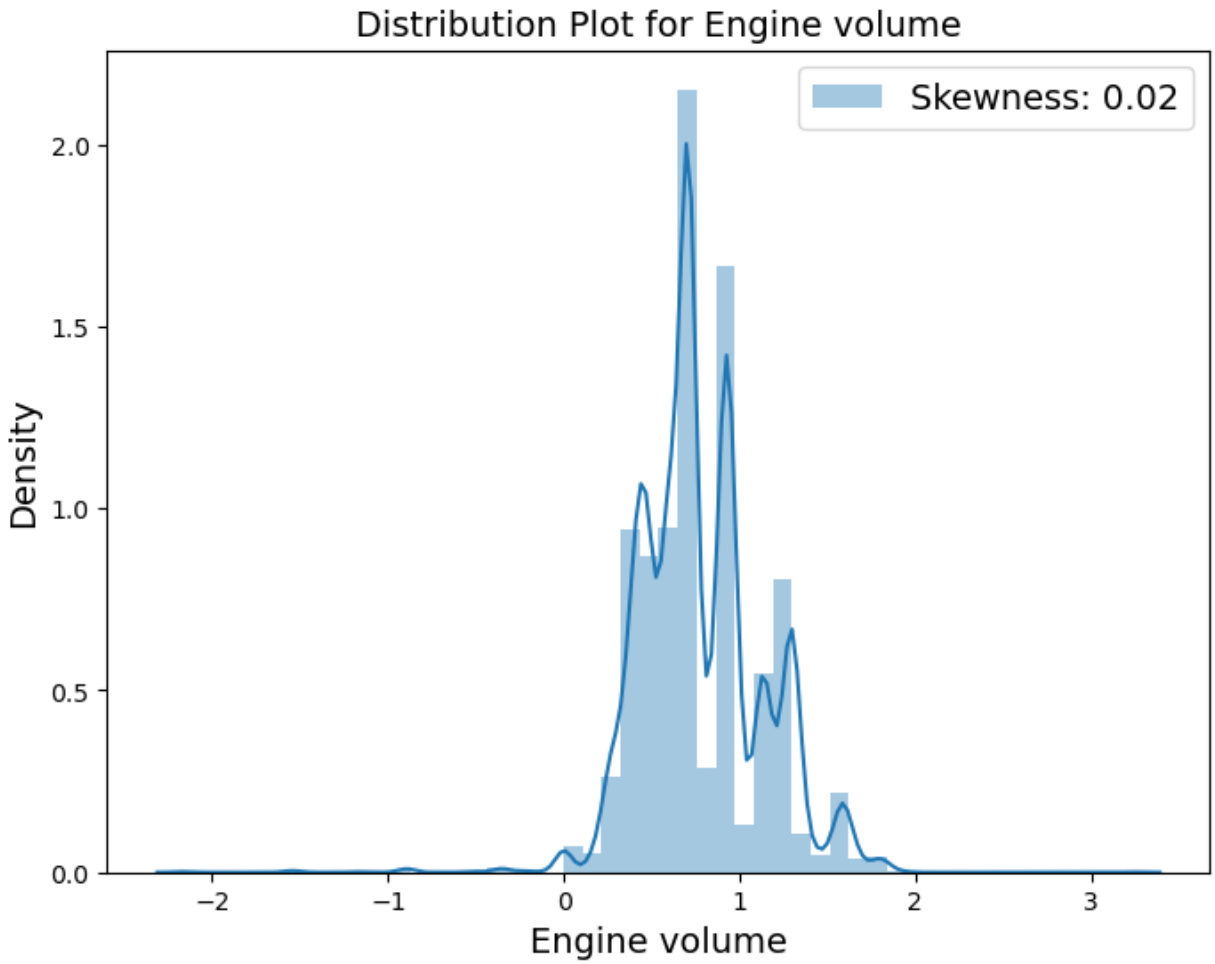
```
warnings.filterwarnings("ignore")

columns_to_review = ['Levy', 'Age', 'Price', 'Mileage', 'Engine volume']

for i in columns_to_review:
    plt.figure(figsize=(8, 6))  # Adjust the figure size as needed
    sns.distplot(car[i], label='Skewness: %.2f' % (car[i].skew()))
    plt.legend()
    plt.title(f'Distribution Plot for {i}')
    plt.show()

warnings.resetwarnings()
```


Distribution Plot for Levy
Skewness: -0.00
Density
Levy
0
1
2
3
4
5
6
7
8
9


Distribution Plot for Age
Skewness: -0.00
Density
Age
0.0
0.5
1.0
1.5
2.0
1.0
1.5
2.0
2.5
3.0
3.5


Distribution Plot for Price
Skewness: 0.38
Density
Price
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0
100
200
300
400


Distribution Plot for Mileage
Skewness: 2.28
Density
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
50
100
150
200
Mileage



After conducting the transformation, we can now notice that the skewness has been taken care of and the data can now be used towards training and testing the model.

In [38]:
```
car
```

Out[38]:

	ID	Price	Levy	Manufacturer	Model	Category	Leather interior	Fuel type	Ei vo
0	45654403	49.647560	7.201868	LEXUS	RX 450	2	Yes	other	1.29
1	44731507	53.121011	6.887520	CHEVROLET	Equinox	2	No	Petrol	1.13
2	45774419	43.164922	6.625353	HONDA	FIT	1	No	Petrol	0.26
3	45769185	33.038016	6.722984	FORD	Escape	2	Yes	other	0.93
4	45809263	47.731553	6.070764	HONDA	FIT	1	Yes	Petrol	0.26
...	...	...	...	...	...	...	...	...	
18849	45798355	43.164922	6.625353	MERCEDES-BENZ	CLK 200	1	Yes	CNG	0.70
18850	45778856	52.183728	6.686749	HYUNDAI	Sonata	1	Yes	Petrol	0.89
18851	45804997	60.948014	6.692684	HYUNDAI	Tucson	2	Yes	Diesel	0.70
18852	45793526	37.368233	7.120145	CHEVROLET	Captiva	2	Yes	Diesel	0.70
18853	45813273	16.872588	6.589227	HYUNDAI	Sonata	1	Yes	other	0.89

18854 rows × 18 columns

In [39]:
```
car.corr()
```

```
/var/folders/1r/4rz_ktkd1r30hx6t62xt8zzw0000gn/T/ipykernel_75835/2754339606.p
y:1: FutureWarning: The default value of numeric_only in DataFrame.corr is dep
recated. In a future version, it will default to False. Select only valid colu
mns or specify the value of numeric_only to silence this warning.
  car.corr()
```

Out[39]:

	ID	Price	Levy	Category	Engine volume	Mileage	Cylinders	Gear t
ID	1.000000	0.023950	0.001361	0.013203	0.002594	0.017979	-0.034945	-0.075
Price	0.023950	1.000000	-0.002287	0.171508	0.003240	-0.045081	-0.055562	0.158
Levy	0.001361	-0.002287	1.000000	0.234161	0.521253	0.001351	0.389134	-0.004
Category	0.013203	0.171508	0.234161	1.000000	0.276533	0.009925	0.174085	-0.049
Engine volume	0.002594	0.003240	0.521253	0.276533	1.000000	0.076301	0.695431	0.045
Mileage	0.017979	-0.045081	0.001351	0.009925	0.076301	1.000000	0.044053	-0.017
Cylinders	-0.034945	-0.055562	0.389134	0.174085	0.695431	0.044053	1.000000	0.090
Gear box type	-0.075694	0.158970	-0.004487	-0.049303	0.045820	-0.017132	0.090804	1.000
Airbags	-0.020249	-0.099048	0.099413	-0.102504	0.262260	0.005463	0.194144	0.036
Age	-0.041954	-0.228173	0.005809	-0.017266	0.033336	0.149203	0.123297	0.346

In order to convert the categorical variables into numerical values, we will be using the one hot encoder technique

Ref - Code from chapter 2 of Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd edition).

In [40]:
```
car_dataset_encoded = pd.get_dummies(data = car, columns = ['Category', 'Fuel t
car_dataset_encoded
```

Out[40]:

	ID	Price	Levy	Manufacturer	Model	Leather interior	Engine volume	Mileage
0	45654403	49.647560	7.201868	LEXUS	RX 450	Yes	1.295043	40.946529
1	44731507	53.121011	6.887520	CHEVROLET	Equinox	No	1.131039	41.201952
2	45774419	43.164922	6.625353	HONDA	FIT	No	0.264186	41.532703
3	45769185	33.038016	6.722984	FORD	Escape	Yes	0.938775	40.181360
4	45809263	47.731553	6.070764	HONDA	FIT	Yes	0.264186	35.614096
...	...	...	...	...	...	...	...	...
18849	45798355	43.164922	6.625353	MERCEDES-BENZ	CLK 200	Yes	0.705963	44.946879
18850	45778856	52.183728	6.686749	HYUNDAI	Sonata	Yes	0.895979	39.830632
18851	45804997	60.948014	6.692684	HYUNDAI	Tucson	Yes	0.705963	37.327756
18852	45793526	37.368233	7.120145	CHEVROLET	Captiva	Yes	0.705963	31.661736
18853	45813273	16.872588	6.589227	HYUNDAI	Sonata	Yes	0.895979	40.986079

18854 rows × 27 columns

We would dropping the below columns, as they would not be impacting the price of a vehicle

In [41]:
```
car_dataset_encoded.drop(['ID', 'Manufacturer', 'Model', 'Leather interior', 'I
```

In [42]:
```
car_dataset_encoded
```

Out[42]:

	Price	Levy	Engine volume	Mileage	Cylinders	Airbags	Age	Category_1
0	49.647560	7.201868	1.295043	40.946529	6.0	12	2.209502	0
1	53.121011	6.887520	1.131039	41.201952	6.0	8	2.150286	0
2	43.164922	6.625353	0.264186	41.532703	4.0	2	2.403890	1
3	33.038016	6.722984	0.938775	40.181360	4.0	0	2.150286	0
4	47.731553	6.070764	0.264186	35.614096	4.0	4	1.932725	1
...	...	...	...	...	...	...	...	...
18849	43.164922	6.625353	0.705963	44.946879	4.0	5	2.644795	1
18850	52.183728	6.686749	0.895979	39.830632	4.0	8	2.150286	1
18851	60.948014	6.692684	0.705963	37.327756	4.0	4	2.209502	0
18852	37.368233	7.120145	0.705963	31.661736	4.0	4	2.360503	0
18853	16.872588	6.589227	0.895979	40.986079	4.0	12	2.085269	1

18854 rows × 21 columns

In [43]:
```
car_label = car["Price"]
car_model_attributes = car_dataset_encoded.drop(columns = ["Price"], errors = '
```

In [44]:
```
car_label
```

Out[44]:
```
0         49.647560
1         53.121011
2         43.164922
3         33.038016
4         47.731553
            ...
18849     43.164922
18850     52.183728
18851     60.948014
18852     37.368233
18853     16.872588
Name: Price, Length: 18854, dtype: float64
```

In [45]:
```
car_model_attributes
```

Out[45]:

	Levy	Engine volume	Mileage	Cylinders	Airbags	Age	Category_1	Category_2
0	7.201868	1.295043	40.946529	6.0	12	2.209502	0	1
1	6.887520	1.131039	41.201952	6.0	8	2.150286	0	1
2	6.625353	0.264186	41.532703	4.0	2	2.403890	1	0
3	6.722984	0.938775	40.181360	4.0	0	2.150286	0	1
4	6.070764	0.264186	35.614096	4.0	4	1.932725	1	0
...	...	...	...	...	...	...	...	...
18849	6.625353	0.705963	44.946879	4.0	5	2.644795	1	0
18850	6.686749	0.895979	39.830632	4.0	8	2.150286	1	0
18851	6.692684	0.705963	37.327756	4.0	4	2.209502	0	1
18852	7.120145	0.705963	31.661736	4.0	4	2.360503	0	1
18853	6.589227	0.895979	40.986079	4.0	12	2.085269	1	0

18854 rows × 20 columns

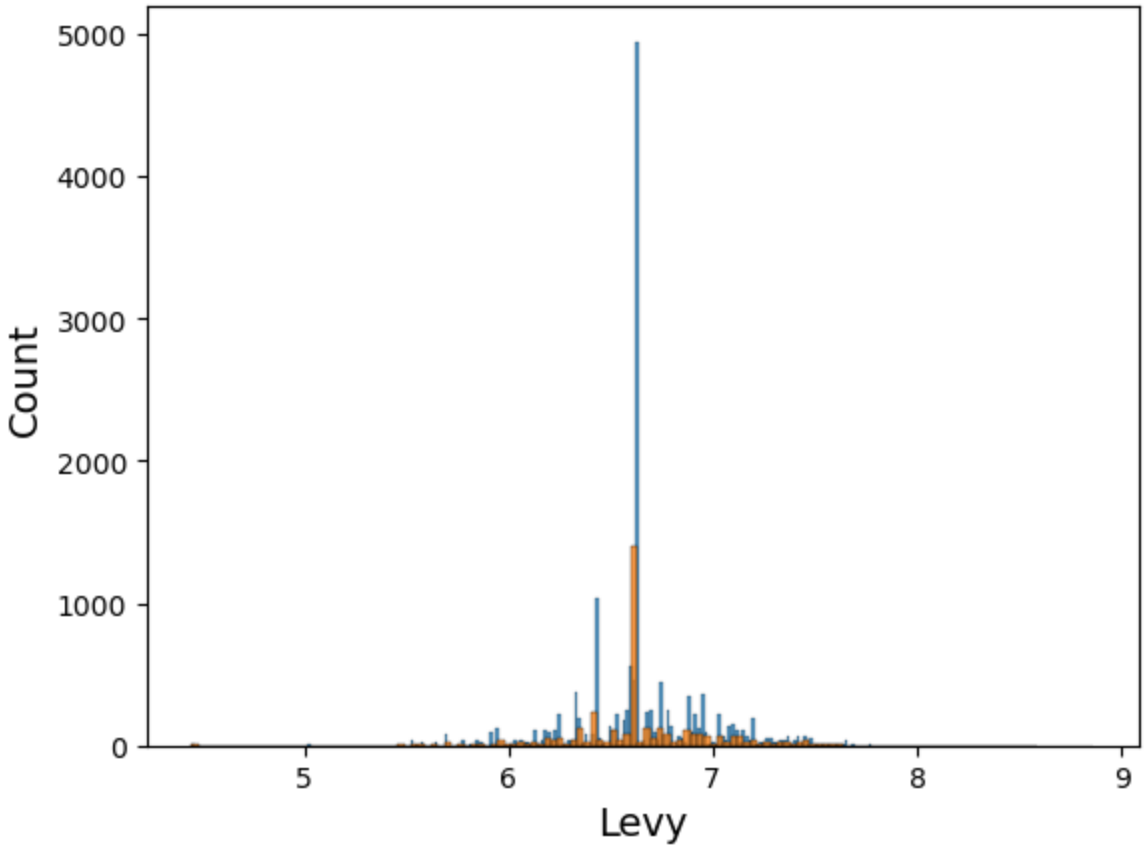
Question D: Select 20% of the data for testing. Describe how you did that and verify that your test portion of the data is representative of the entire dataset.

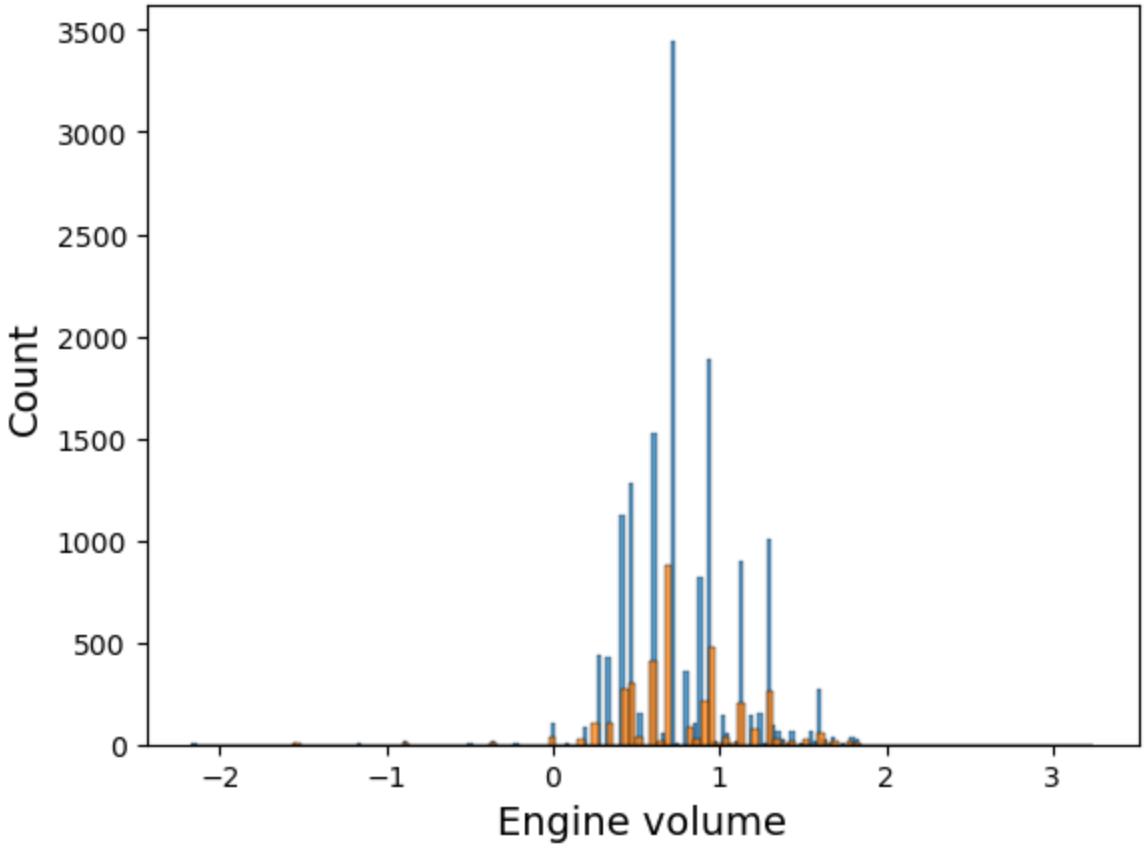
In [46]:
```
from sklearn.model_selection import train_test_split
```

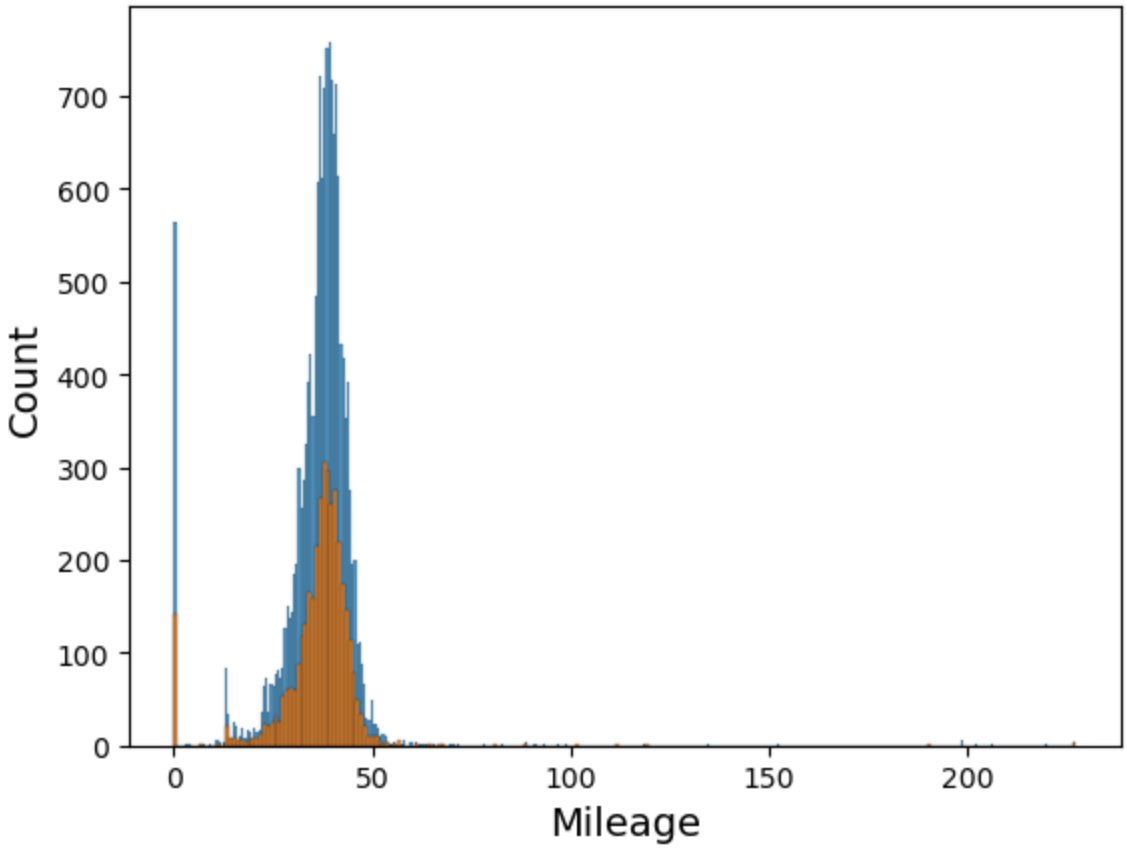
In [47]:
```
## We use train_test_split with parameter shuffle set to True and test_size as
X_train, X_test, y_train, y_test = train_test_split(car_model_attributes, car_
```

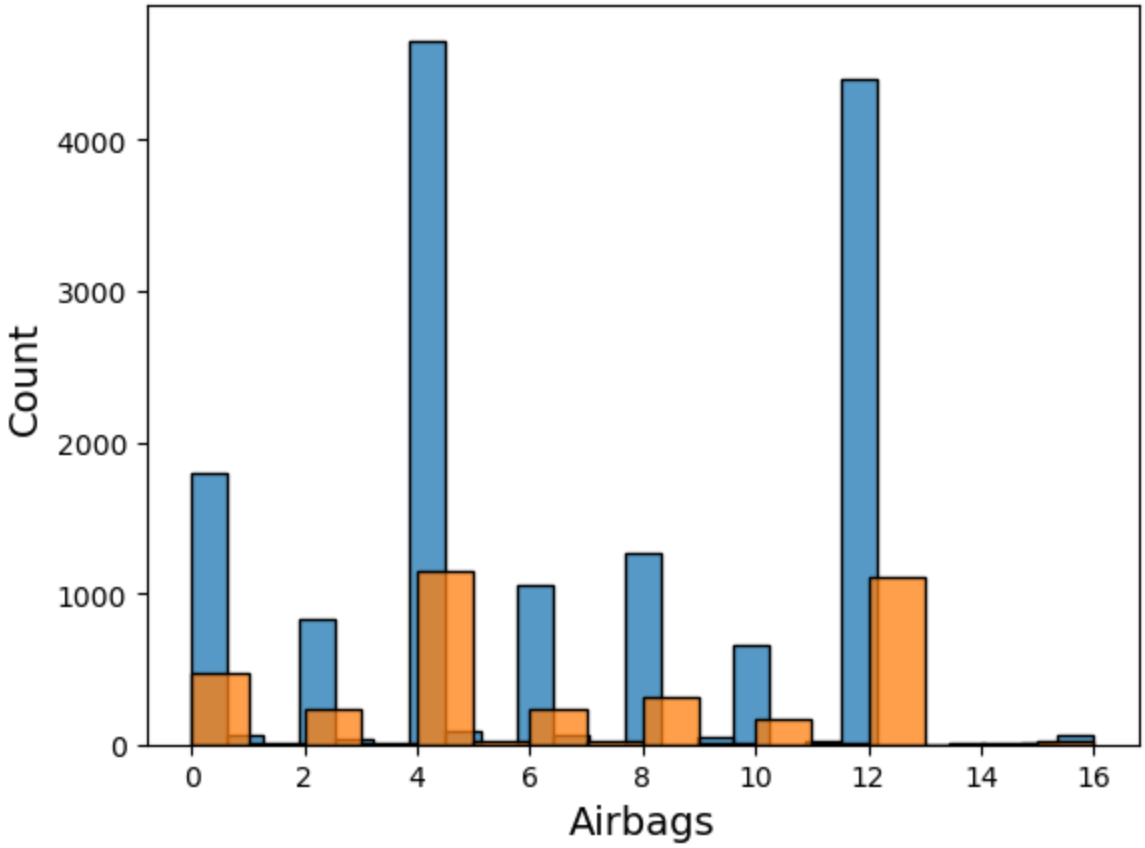
In [48]:
```
X_train_numerical = X_train.select_dtypes(include=np.number)
X_test_numerical = X_test.select_dtypes(include=np.number)
```

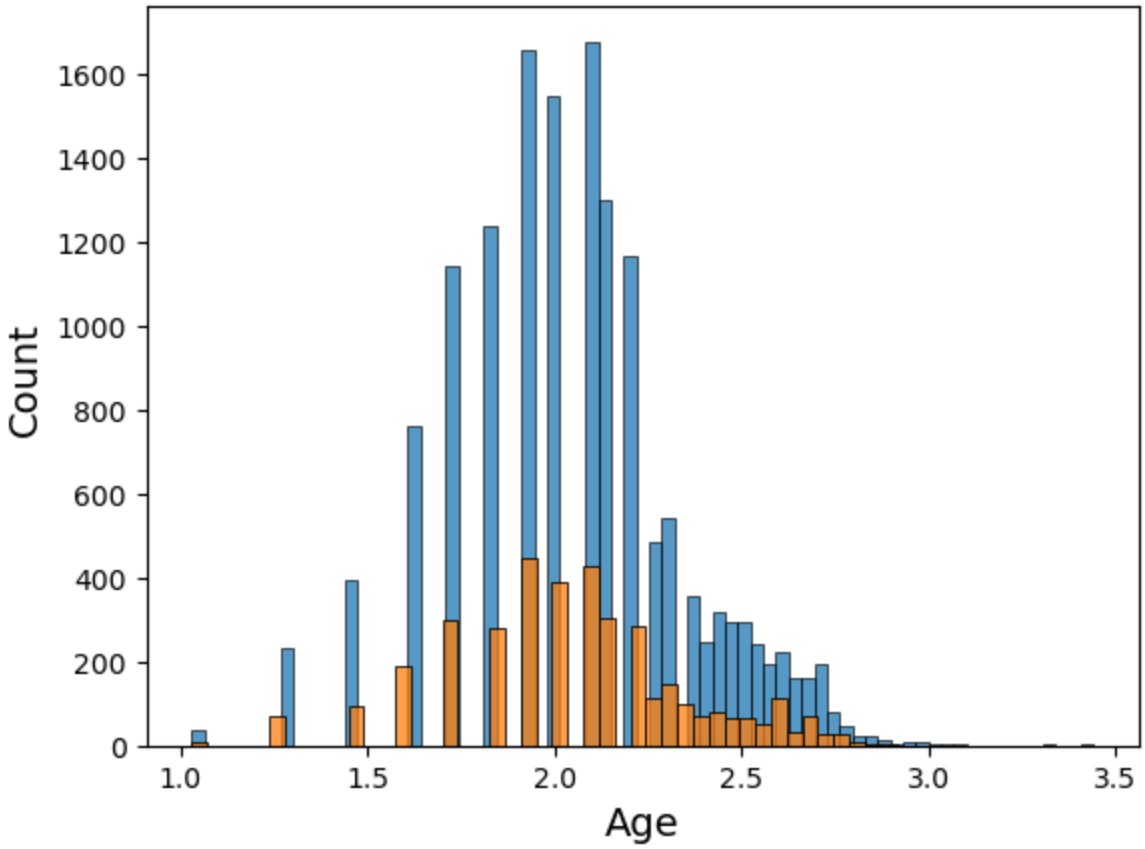
In [49]:
```
car_model_numerical = ['Levy','Engine volume','Mileage','Airbags','Age']

for colname in car_model_numerical:
      plt.subplot(111)
      sns.histplot(data=X_train_numerical,x=colname)

      plt.subplot(111)
      sns.histplot(data=X_test_numerical,x=colname)

      plt.show()
```


5000
4000
3000
2000
1000
0
Count
5
6
7
8
9
Levy


3500
3000
2500
2000
1500
1000
500
0
Count
−2
−1
0
1
2
3
Engine volume


Count
Mileage
0
100
200
300
400
500
600
700
0
50
100
150
200


Count
Airbags
0
1000
2000
3000
4000
0
2
4
6
8
10
12
14
16



From the above distribution set, we can analyze that our test set and train set follow similar patterns and hence we can verify that we have randomly picked a representative train and test set

In [50]:
```
X_train
```

Out[50]:

	Levy	Engine volume	Mileage	Cylinders	Airbags	Age	Category_1	Category_2
15843	6.625353	0.705963	40.549739	4.0	8	2.552685	0	1
14682	6.625353	0.596985	37.861362	6.0	12	2.013272	1	0
8953	6.625353	0.705963	36.219356	4.0	12	2.672663	0	1
9623	6.686749	0.895979	37.083329	4.0	6	2.150286	0	1
638	6.692684	0.705963	38.010142	4.0	4	2.209502	0	1
...	...	...	...	...	...	...	...	...
11284	6.622816	0.938775	40.075783	4.0	12	2.013272	1	0
11964	6.604872	0.705963	44.533389	4.0	4	1.841473	1	0
5390	6.526852	0.705963	38.103222	4.0	4	2.150286	1	0
860	6.765659	0.895979	0.000000	4.0	0	1.841473	1	0
15795	6.589227	0.895979	35.761083	4.0	0	2.085269	1	0

15083 rows × 20 columns

In [51]:
```
y_train = pd.DataFrame(y_train, columns=['Price'])

y_train
```

Out[51]:

	Price
15843	44.132076
14682	63.231601
8953	39.921886
9623	55.725908
638	49.348201
...	...
11284	58.320330
11964	47.338228
5390	53.639592
860	51.534978
15795	14.098127

15083 rows × 1 columns

Question E: Train a Linear Regression model using the training data with four-fold cross-validation using appropriate evaluation metric. Do this with a closed-form solution (using the Normal Equation or SVD) and with SGD. Perform Ridge, Lasso and Elastic Net regularization – try a few values of penalty term and describe its impact. Explore the impact of other hyperparameters, like batch size and learning rate (no need for grid search). Describe your findings. For SGD, display the training and validation loss as a function of training iteration.

In [52]:
```
#Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error

kf = KFold(n_splits=4)
lin_reg = LinearRegression()
# Create a list to store the performance metrics (e.g., Mean Squared Error) fo
performance_metrics = []

for train_index, test_index in kf.split(X_train):
    X_train_kFold, X_test_kFold = X_train.iloc[train_index,:], X_train.iloc[tes
    y_train_kFold, y_test_kFold = y_train.iloc[train_index], y_train.iloc[test_


  # Fit the linear regression model on the training data
    lin_reg.fit(X_train_kFold, y_train_kFold)
```

```
  # Make predictions on the test data
    y_pred_kFold = lin_reg.predict(X_test_kFold)

  # Calculate the Mean Squared Error for this iteration
    mse = mean_squared_error(y_test_kFold, y_pred_kFold)

    performance_metrics.append(mse)

# Calculate the average performance metric across all iterations
average_performance = np.mean(performance_metrics)
```

In [53]:
```
average_performance
```

Out[53]:
```
250.7248551398091
```

The evaluation metric used throughout the project, is mean squared error. Based on the above training results, the average mse for the multi-linear model is 250.7248551398091

In [54]:
```
lin_reg.predict(X_test)
```

Out[54]:
```
array([[47.57603856],
       [38.80014893],
       [34.0037786 ],
       ...,
       [46.02848233],
       [41.82593644],
       [33.42105592]])
```

In [55]:
```
X_train = X_train.values
y_train = y_train.values
```

In [56]:
```
# As asked in the question, we would be running a four-fold valiadation, hence
k = 4

# Storing the theta values for each fold
theta_values = []

# Initialize KFold cross-validator
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Initialize an empty list to store the MSE values for each fold
mse_values = []

# Perform k-fold cross-validation
for train_index, test_index in kf.split(X_train):
    X_train_fold, X_val_fold = X_train[train_index], X_train[test_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[test_index]

    # Calculating the coefficients using the normal equation
    X_transpose = np.transpose(X_train_fold)
    X_transpose_X = np.dot(X_transpose, X_train_fold)
    X_transpose_X_inv = np.linalg.inv(X_transpose_X)
    X_transpose_y = np.dot(X_transpose, y_train_fold)
    theta = np.dot(X_transpose_X_inv, X_transpose_y)

    # Append the theta values for this fold to the list
    theta_values.append(theta)
```

```
    # Make predictions using the calculated theta values
    y_val_pred = np.dot(X_val_fold, theta)

    # Calculate and append the MSE for this fold
    mse_fold = mean_squared_error(y_val_fold, y_val_pred)
    mse_values.append(mse_fold)

# Calculate the average coefficients (theta) across all folds
average_theta = np.mean(theta_values, axis=0)

# Calculate the average MSE across all folds
average_mse = np.mean(mse_values)

print("Average Coefficients (theta) across all folds:", average_theta)
print("Average MSE across all folds:", average_mse)
```

```
Average Coefficients (theta) across all folds: [[-3.76984682e+02]
 [ 1.28918102e+02]
 [ 5.91208288e+00]
 [-2.16076864e+01]
 [ 6.88947221e+00]
 [ 5.20481079e+02]
 [-6.39323726e+16]
 [-6.39323726e+16]
 [ 4.92806878e+16]
 [ 4.92806878e+16]
 [ 4.92806878e+16]
 [ 4.92806878e+16]
 [ 4.92806878e+16]
 [ 1.18550030e+16]
 [ 1.18550030e+16]
 [ 2.79668170e+15]
 [ 2.79668170e+15]
 [ 2.79668170e+15]
 [-2.62400000e+03]
 [-2.56800000e+03]]
Average MSE across all folds: 361054.73779043125
```

While the MSE from OLS is representative of the data, the MSE arrived at from using the normal equation is unrealistically high. In order to take this account, we would be conducting performing regularization techniques, such that the mse is more representative of the data.

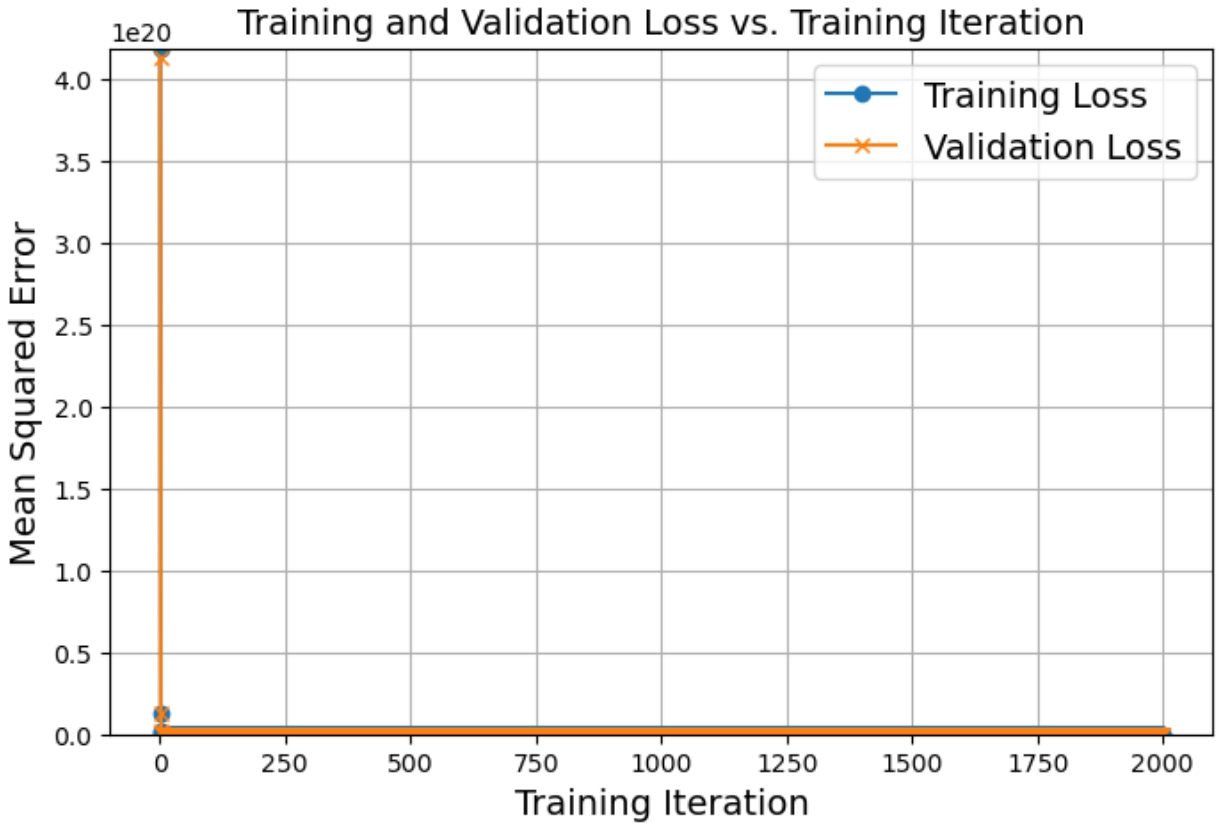
In [57]:

```
import warnings
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold

warnings.filterwarnings("ignore")

# Define the number of folds (k)
k = 4

# Create the SGDRegressor model with appropriate hyperparameters
alphaList = [0.001, 0.01, 0.1, 1, 10]
for alpha in alphaList:
    sgd_model = SGDRegressor(loss='squared_error', alpha=alpha, max_iter=500, t
```

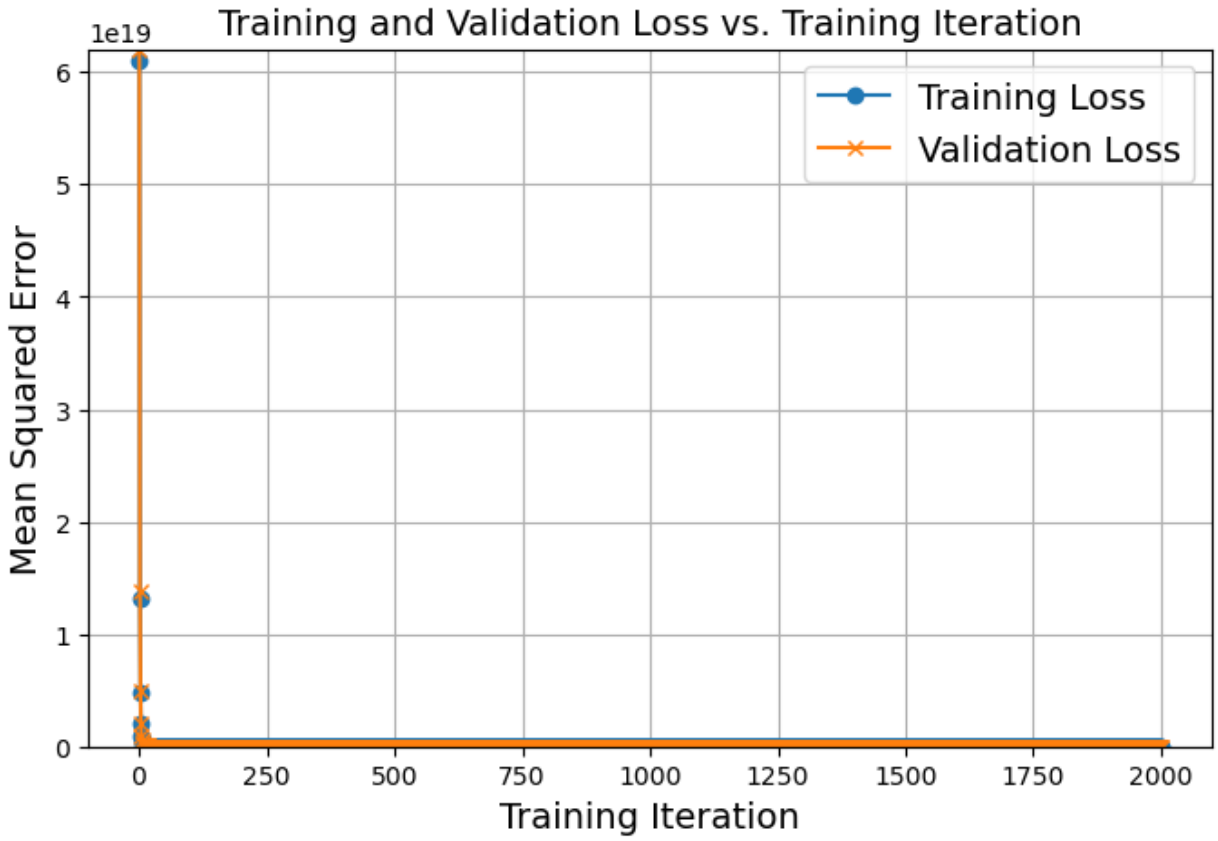
```
# Initialize lists to store training and validation loss
training_loss = []
validation_loss = []

# Initialize KFold cross-validator
kf = KFold(n_splits=k, shuffle=True, random_state=42)

for train_index, test_index in kf.split(X_train):
    X_train_fold, X_val_fold = X_train[train_index], X_train[test_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[test_index]

    # Number of training iterations
    n_iterations = 500

    # Early stopping parameters
    early_stopping_rounds = 5  # Number of iterations with no improvement t
    best_val_loss = float('inf')  # Initialize the best validation loss to
    no_improvement_count = 0  # Initialize the count of iterations with no

    for iteration in range(n_iterations):
        # Fit the model for one iteration (one pass through the training da
        sgd_model.partial_fit(X_train_fold, y_train_fold)

        # Predict on the training data
        y_train_pred = sgd_model.predict(X_train_fold)

        # Calculate training loss (Mean Squared Error) and append to the li
        train_loss = mean_squared_error(y_train_fold, y_train_pred)
        training_loss.append(train_loss)

        # Predict on the validation data
        y_val_pred = sgd_model.predict(X_val_fold)

        # Calculate validation loss (Mean Squared Error) and append to the
        val_loss = mean_squared_error(y_val_fold, y_val_pred)
        validation_loss.append(val_loss)

        # Check for early stopping
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            no_improvement_count = 0
        else:
            no_improvement_count += 1

        if no_improvement_count >= early_stopping_rounds:
            print(f"Early stopping at iteration {iteration + 1} due to no i
            break

# Plot training and validation loss as a function of training iteration
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(training_loss) + 1), training_loss, label='Training L
plt.plot(range(1, len(validation_loss) + 1), validation_loss, label='Valida
plt.xlabel('Training Iteration')
plt.ylabel('Mean Squared Error')
plt.title('Training and Validation Loss vs. Training Iteration')
plt.legend()
plt.grid()
```

```
    # Set the y-axis limit (change the values as needed)
    y_limit = (0.5, max(max(training_loss), max(validation_loss)))
    plt.ylim(y_limit)

    # Final model evaluation
    y_pred_sgd = sgd_model.predict(X_test)
    final_mse = mean_squared_error(y_test, y_pred_sgd)
    print("Final Mean Squared Error (SGD):", final_mse)

    plt.show()

warnings.resetwarnings()
```

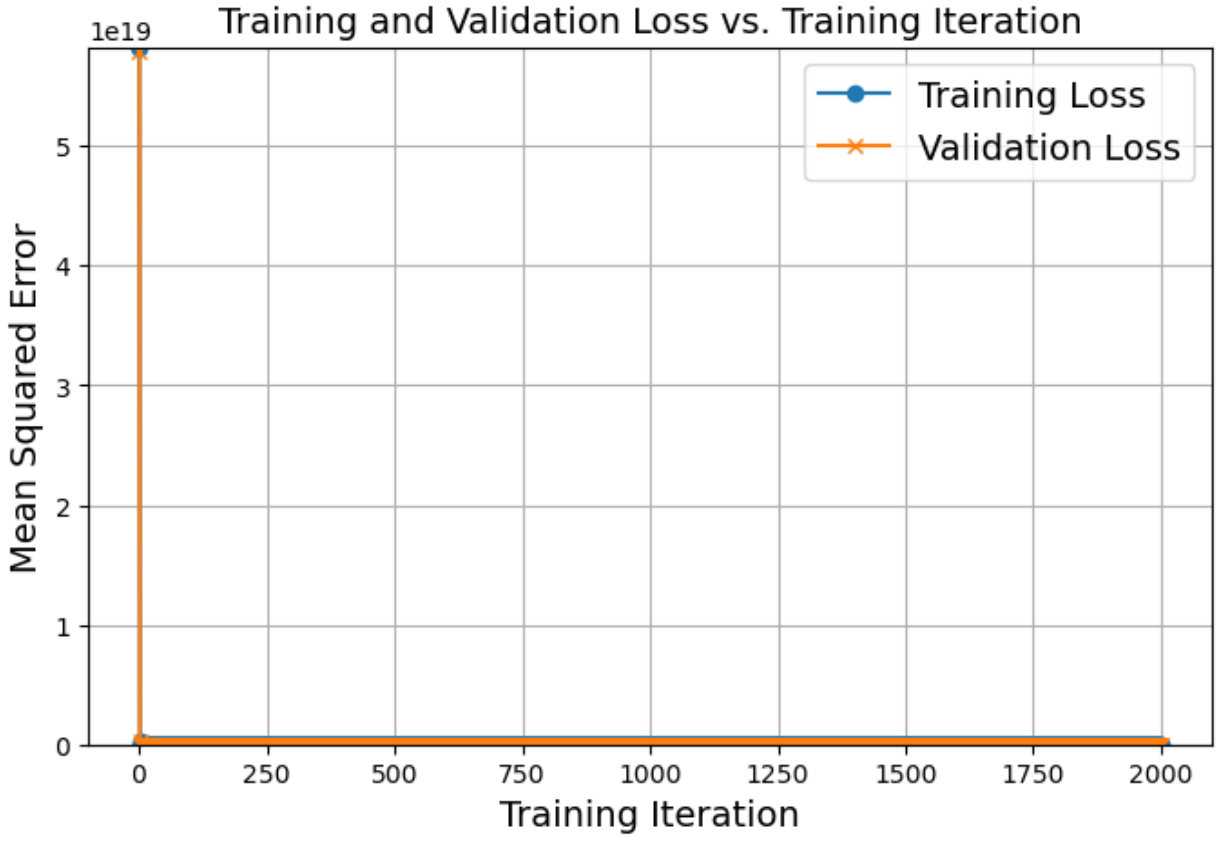
```
Final Mean Squared Error (SGD): 59156393640.71866
```



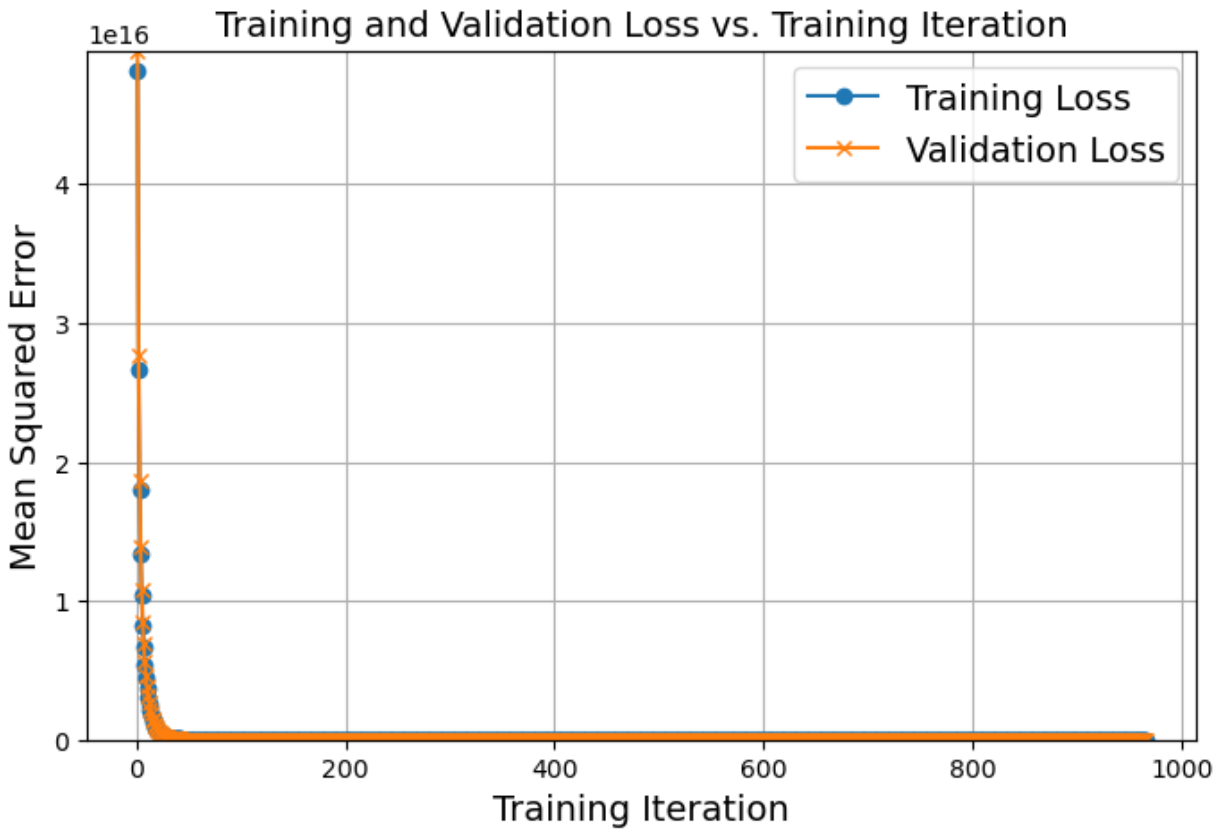

```
Final Mean Squared Error (SGD): 269511236045.3755
```



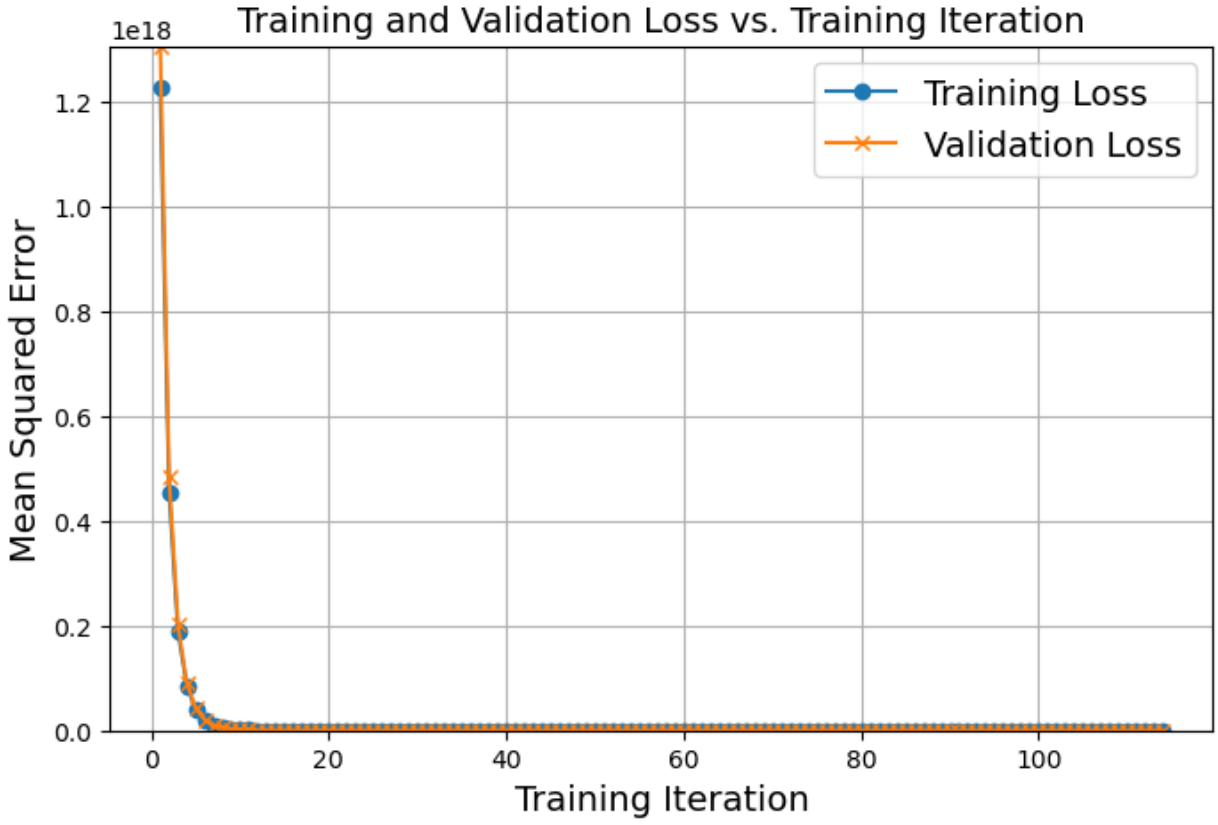

```
Final Mean Squared Error (SGD): 1351.2974080321453
```




```
Early stopping at iteration 454 due to no improvement in validation loss.
Early stopping at iteration 6 due to no improvement in validation loss.
Early stopping at iteration 6 due to no improvement in validation loss.
Final Mean Squared Error (SGD): 321.25075035925784
```




```
Early stopping at iteration 96 due to no improvement in validation loss.
Early stopping at iteration 6 due to no improvement in validation loss.
Early stopping at iteration 6 due to no improvement in validation loss.
Early stopping at iteration 6 due to no improvement in validation loss.
Final Mean Squared Error (SGD): 340.5961580549421
```



One of the inferences we can make from the training and validation loss, is that it is conducting a good fit. However, one of the disadvantage of SGD Regression model is that it sensitive to the features having different scales of data. This could be one of the reasons that could explain the high MSE. Ref: https://scikit-learn.org/stable/modules/sgd.html

Note: Standard Scaler or MinMaxScaler has anot been applied as the feature engineering was already performed on the dataset using box-ocox while we were controlling for the skewness.

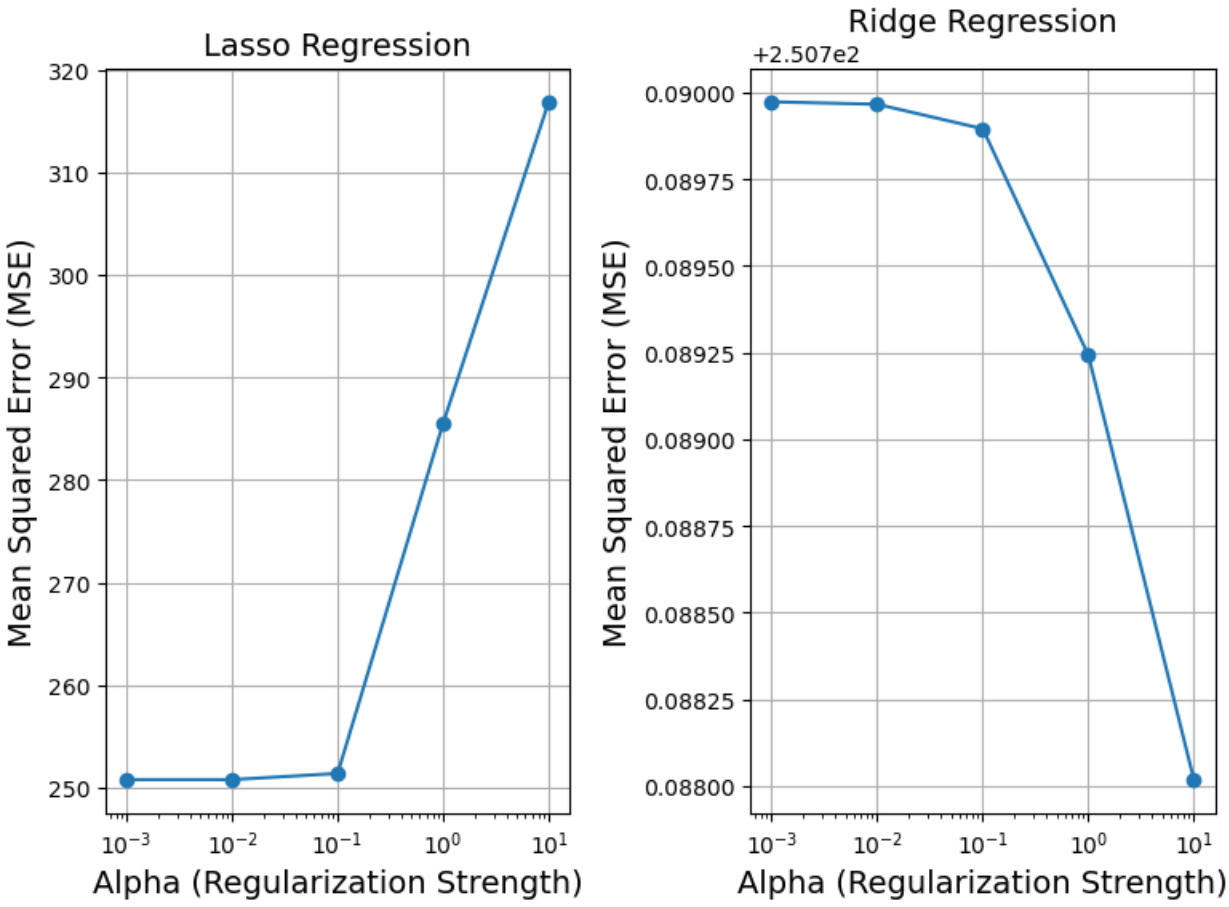
In [58]:
```
# Analyzing the mean scores
print(f"Mean MSE: {np.mean(final_mse)}")
```

```
Mean MSE: 340.5961580549421
```

In [59]:
```
from sklearn.linear_model import Lasso, Ridge
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt

warnings.filterwarnings("ignore")

# Define a list of alpha values to try
alphas = [0.001, 0.01, 0.1, 1.0, 10.0]

# Initialize empty lists to store the MSE values
lasso_mse_values = []
ridge_mse_values = []

for alpha in alphas:
    lasso_mse_fold = []
    ridge_mse_fold = []

    for train_index, test_index in kf.split(X_train):
        X_train_fold, X_test_fold = X_train[train_index], X_train[test_index]
        y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]

        # Create Lasso and Ridge models with the current alpha
        lasso_reg = Lasso(alpha=alpha)
        ridge_reg = Ridge(alpha=alpha)

        # Fit the Lasso model
        lasso_reg.fit(X_train_fold, y_train_fold)

        # Make predictions with the trained Lasso model
        lasso_predictions = lasso_reg.predict(X_test_fold)

        # Calculate MSE for Lasso
        lasso_mse = mean_squared_error(y_test_fold, lasso_predictions)
        lasso_mse_fold.append(lasso_mse)

        # Fit the Ridge model
        ridge_reg.fit(X_train_fold, y_train_fold)

        # Make predictions with the trained Ridge model
        ridge_predictions = ridge_reg.predict(X_test_fold)

        # Calculate MSE for Ridge
        ridge_mse = mean_squared_error(y_test_fold, ridge_predictions)
```

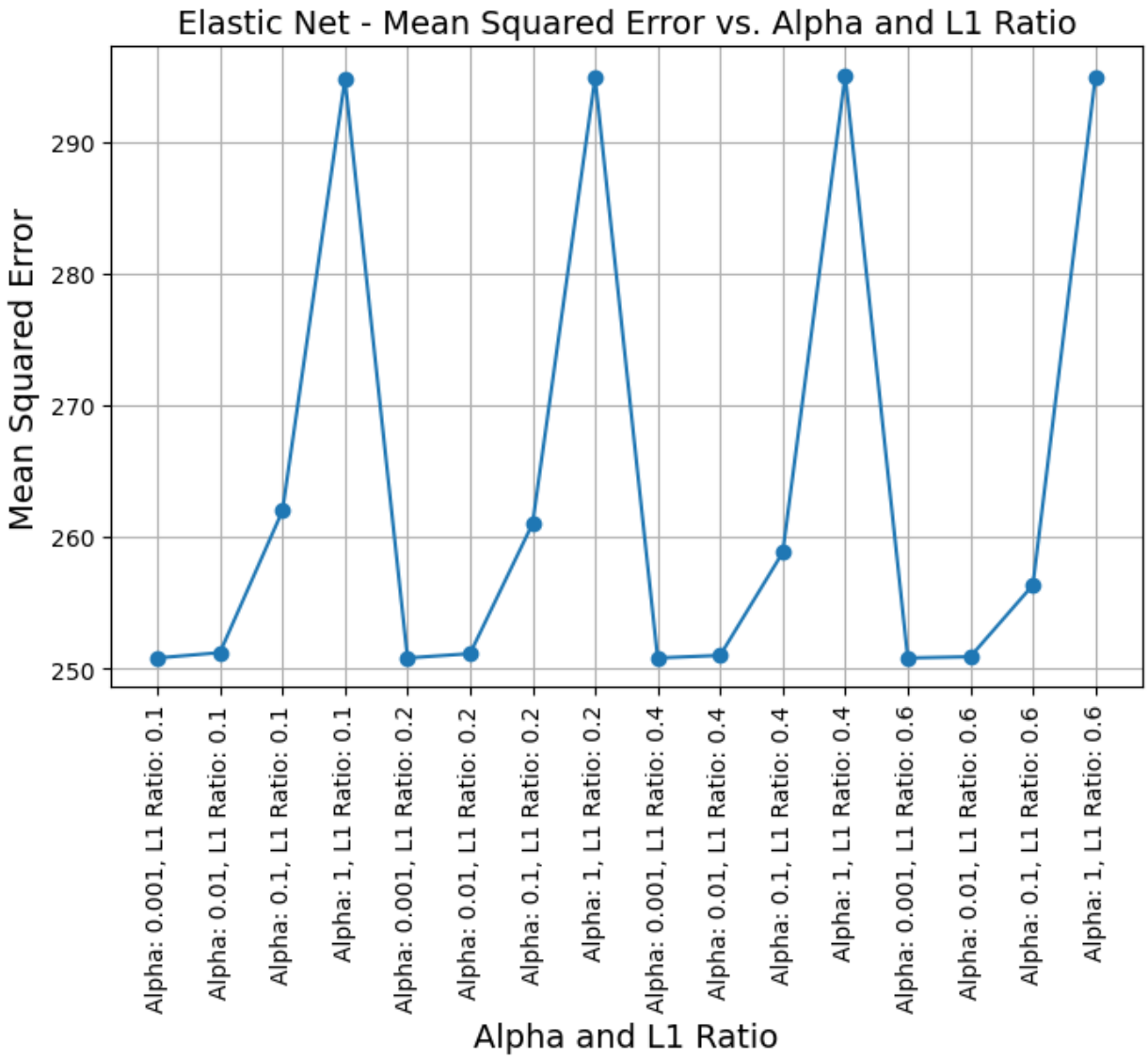
```
        ridge_mse_fold.append(ridge_mse)

    # Calculate the mean MSE across the four folds for each alpha
    lasso_mse_values.append(np.mean(lasso_mse_fold))
    ridge_mse_values.append(np.mean(ridge_mse_fold))

    # Print the MSE values for each alpha
    print(f"Alpha = {alpha}")
    print("Lasso Regression Mean CV MSE:", np.mean(lasso_mse_fold))
    print("Ridge Regression Mean CV MSE:", np.mean(ridge_mse_fold))
    print("===")

# Create individual plots for each regularization technique
plt.figure(figsize=(12, 6))

# Lasso Regression Plot
plt.subplot(131)
plt.semilogx(alphas, lasso_mse_values, marker='o')
plt.xlabel('Alpha (Regularization Strength)')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Lasso Regression')
plt.grid(True)

# Ridge Regression Plot
plt.subplot(132)
plt.semilogx(alphas, ridge_mse_values, marker='o')
plt.xlabel('Alpha (Regularization Strength)')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Ridge Regression')
plt.grid(True)

plt.tight_layout()
plt.show()

warnings.resetwarnings()
```

```
Alpha = 0.001
Lasso Regression Mean CV MSE: 250.78712824661574
Ridge Regression Mean CV MSE: 250.78997356091526
===
Alpha = 0.01
Lasso Regression Mean CV MSE: 250.78582231486007
Ridge Regression Mean CV MSE: 250.7899664275362
===
Alpha = 0.1
Lasso Regression Mean CV MSE: 251.38818553682682
Ridge Regression Mean CV MSE: 250.789895644287
===
Alpha = 1.0
Lasso Regression Mean CV MSE: 285.54851057400396
Ridge Regression Mean CV MSE: 250.7892426774809
===
Alpha = 10.0
Lasso Regression Mean CV MSE: 316.8640941772081
Ridge Regression Mean CV MSE: 250.78801862355635
===
```



In [60]:
```
import warnings
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold

warnings.filterwarnings("ignore")

# Define the number of folds (k)
k = 4

# Create lists of alpha (regularization strength) and l1_ratio (L1 mixing para
alpha_list = [0.001, 0.01, 0.1, 1]
l1_ratio_list = [0.1, 0.2, 0.4, 0.6]  # Different L1 ratios for Elastic Net

alpha_l1_list = []  # Store alpha and l1_ratio combinations
elastic_mse_vals = []  # Store Elastic Net MSE values

for l1_ratio in l1_ratio_list:
    for alpha in alpha_list:
        elastic_net_model = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, max_ite

        # Initialize KFold cross-validator
        kf = KFold(n_splits=k, shuffle=True, random_state=42)

        fold_training_losses = []  # Store training losses for each fold
        fold_validation_losses = []  # Store validation losses for each fold

        for train_index, test_index in kf.split(X_train):  # You should replac
```

```
            X_train_fold, X_test_fold = X_train[train_index], X_train[test_inde
            y_train_fold, y_test_fold = y_train[train_index], y_train[test_inde

            # Fit the Elastic Net model
            elastic_net_model.fit(X_train_fold, y_train_fold)

            # Predict on the training data
            y_train_pred = elastic_net_model.predict(X_train_fold)

            # Calculate training loss (Mean Squared Error) and append to the li
            train_loss = mean_squared_error(y_train_fold, y_train_pred)
            fold_training_losses.append(train_loss)

            # Predict on the validation data
            y_val_pred = elastic_net_model.predict(X_test_fold)

            # Calculate validation loss (Mean Squared Error) and append to the
            val_loss = mean_squared_error(y_test_fold, y_val_pred)
            fold_validation_losses.append(val_loss)

        # Calculate the mean training and validation loss across folds for this
        mean_training_loss = np.mean(fold_training_losses)
        mean_validation_loss = np.mean(fold_validation_losses)  # Fix this line

        alpha_l1_list.append((alpha, l1_ratio))
        elastic_mse_vals.append(mean_validation_loss)


# Plot training and validation loss as a function of alpha
plt.figure(figsize=(8, 5))
plt.plot(range(len(alpha_l1_list)), elastic_mse_vals, marker='o')
plt.xticks(range(len(alpha_l1_list)), [f"Alpha: {alpha}, L1 Ratio: {l1_ratio}"
plt.xlabel('Alpha and L1 Ratio')
plt.ylabel('Mean Squared Error')
plt.title('Elastic Net - Mean Squared Error vs. Alpha and L1 Ratio')
plt.grid()
plt.show()

warnings.resetwarnings()
```



After running the Ridge, Lasso and Elastic Net Regularization techniques, we can make the following inferences regarding it's impact:

- We can notice that as the alpha value increases, the mean CV MSE is also increasing. Thus implying that higher values of alpha lead to more regularization, thereby increasing bias and decreasing variance. This would lead to under-fitting the model
- Further, Lasso can be best used when the dataset has features with poor predictive power. While ridge regression is useful for the grouping effect, in which colinear features can be selected together. Further evaluation would be conducted during the file model evaluation, as to decide which model would best predict the price of the car given the features.

Ref: https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036

In [61]:
```
X_train.shape
```

Out[61]:
```
(15083, 20)
```

In [62]:
```
# Inside the minibatch_gradient_descent function
def minibatch_gradient_descent(X, y, theta, lr, n_iter, batch_size, gradient_cl
    cost_history = []
    for _ in range(n_iter):
        indices = np.random.choice(len(X), batch_size, replace=False)
        X_batch = X[indices]
        y_batch = y[indices]
        gradient = (1 / batch_size) * X_batch.T.dot(X_batch.dot(theta) - y_batc

        # Clip gradients to prevent extreme updates
        gradient = np.clip(gradient, -gradient_clip_threshold, gradient_clip_th

        theta -= lr * gradient
        cost = np.mean((X.dot(theta) - y) ** 2)
        cost_history.append(cost)
    return theta, cost_history
```

In [63]:
```
X_train.shape
```

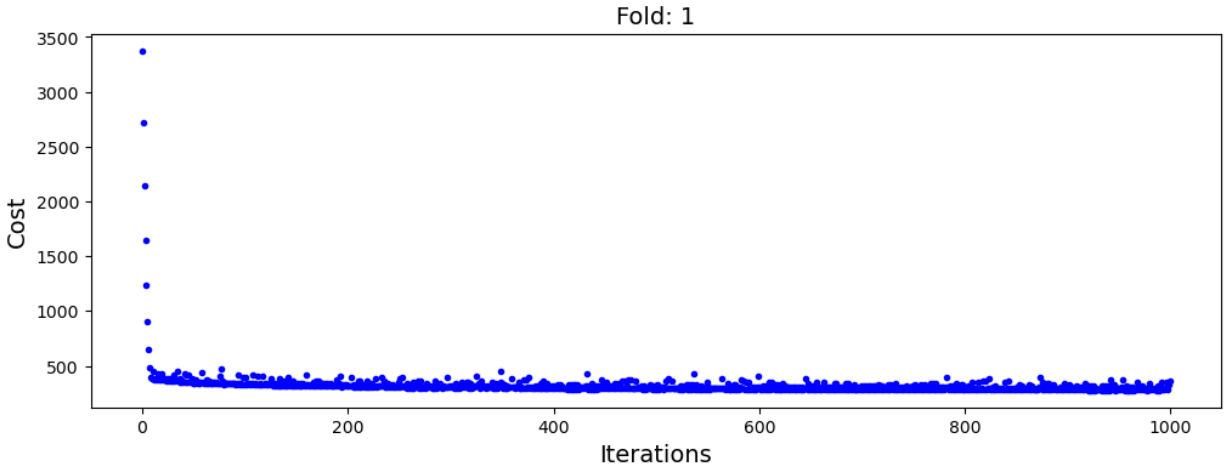
Out[63]:
```
(15083, 20)
```

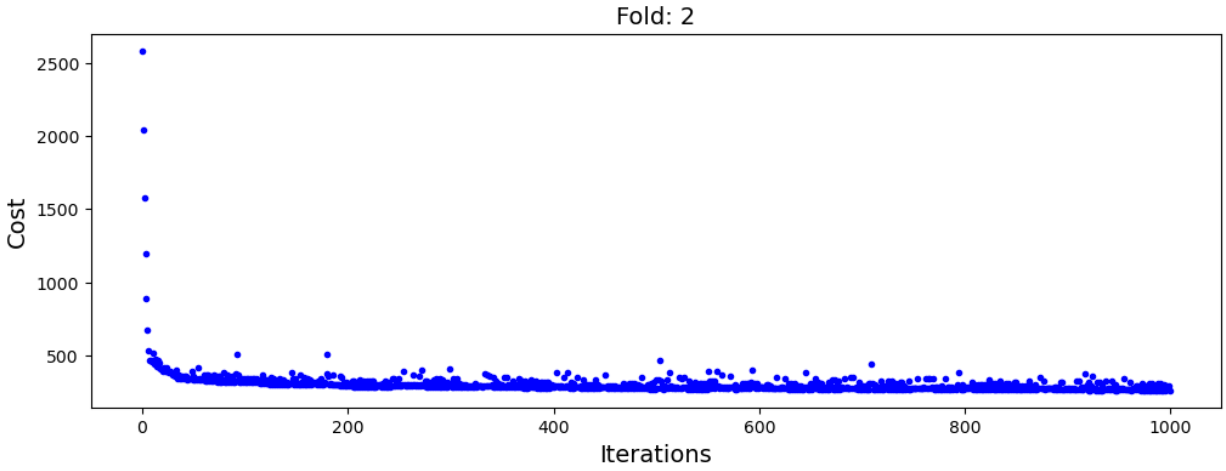
Trial one: Batch Size - 20 and Learning Rate = 0.1

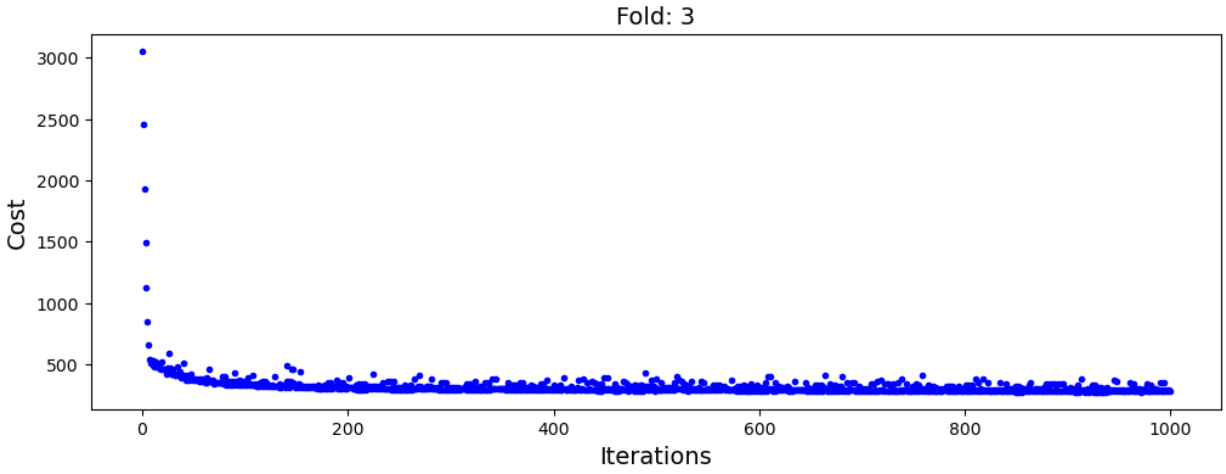
In [64]:
```
i = 1
gradient_clip_threshold = 1.0  # Adjust this threshold as needed

for train_index, val_index in kf.split(X_train):
    X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]

    # Add bias column to fold data
    X_train_fold = np.c_[np.ones((len(X_train_fold), 1)), X_train_fold]
    X_val_fold = np.c_[np.ones((len(X_val_fold), 1)), X_val_fold]

    lr = 0.1
    n_iter = 1000
    batch_size = 20
    theta = np.random.randn(21, 1)

    # Train the model using minibatch gradient descent with gradient clipping
    theta, cost_history = minibatch_gradient_descent(X_train_fold, y_train_fold

    print('Predictions for fold:', i)
    print('Intercept value:', theta[0, 0], '\nWeight values:', theta[1:])

    # Use the validation data and mean square error to evaluate performance
    N = len(y_val_fold)
    y_hat = np.dot(X_val_fold, theta)
    mse_val = (1 / N) * (np.sum(np.square(y_hat - y_val_fold)))
    print('Mean Square Error:', round(mse_val, 3), '\n')

    fig, ax = plt.subplots(figsize=(12, 4))
    plt.title('Fold: ' + str(i))
    ax.set_ylabel('Cost')
    ax.set_xlabel('Iterations')
    _ = ax.plot(range(n_iter), cost_history, 'b.')

    i += 1
```

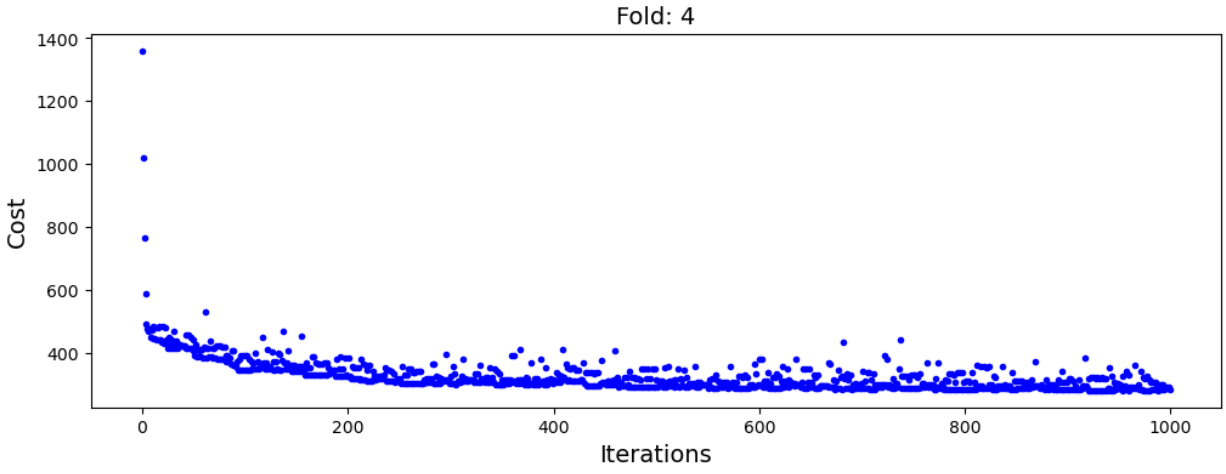
```
Predictions for fold: 1
Intercept value: 5.625893832459101
Weight values: [[ 4.92944497]
 [-2.46666201]
 [ 0.16910987]
 [-0.28277652]
 [-0.15361621]
 [-4.55657919]
 [ 4.77485435]
 [ 7.98626133]
 [-2.6176143 ]
 [10.34134463]
 [ 7.21733247]
 [ 4.01966064]
 [ 0.57780019]
 [ 2.62688943]
 [10.71322657]
 [ 0.53880457]
 [ 4.51300852]
 [ 3.27129833]
 [ 5.66799615]
 [ 2.13194657]]
Mean Square Error: 357.176

Predictions for fold: 2
Intercept value: 5.090359866033951
Weight values: [[ 4.3491479 ]
 [-3.52279082]
 [ 0.0193452 ]
 [ 0.05827932]
 [-0.45810625]
 [-3.23960355]
 [ 2.33518459]
 [ 6.2217895 ]
 [-4.10303755]
 [10.79103191]
 [ 7.14699443]
 [ 4.02425126]
 [ 0.73990764]
 [ 2.00899113]
 [11.42349786]
 [ 1.8716997 ]
 [ 5.55489429]
 [ 2.07881377]
 [ 6.82134208]
 [ 1.21903569]]
Mean Square Error: 320.867

Predictions for fold: 3
Intercept value: 7.660526636261999
Weight values: [[ 4.42536772]
 [-2.81139661]
 [-0.21237567]
 [-0.55057953]
 [-0.25753381]
 [-3.80416554]
 [ 4.5172334 ]
 [ 8.44441394]
 [-1.52031786]
 [10.88020179]
```

```
 [ 6.90180347]
 [ 3.25178386]
 [ 0.41745271]
 [ 3.47418572]
 [11.0897594 ]
 [ 2.76559052]
 [ 5.99371277]
 [ 3.42182634]
 [ 8.40559319]
 [ 3.67275441]]
Mean Square Error: 263.397

Predictions for fold: 4
Intercept value: 5.534530715345567
Weight values: [[ 4.54312275]
 [-1.89073228]
 [-0.02460458]
 [-0.69507151]
 [ 0.03643599]
 [-2.90094439]
 [ 1.61876876]
 [ 4.5987207 ]
 [-2.98403132]
 [10.37563131]
 [ 5.62612267]
 [ 2.97450545]
 [-0.40344995]
 [ 1.18469815]
 [ 9.18360854]
 [-0.5899539 ]
 [ 3.7536803 ]
 [ 2.13502315]
 [ 7.47160762]
 [ 2.51147547]]
Mean Square Error: 265.186
```









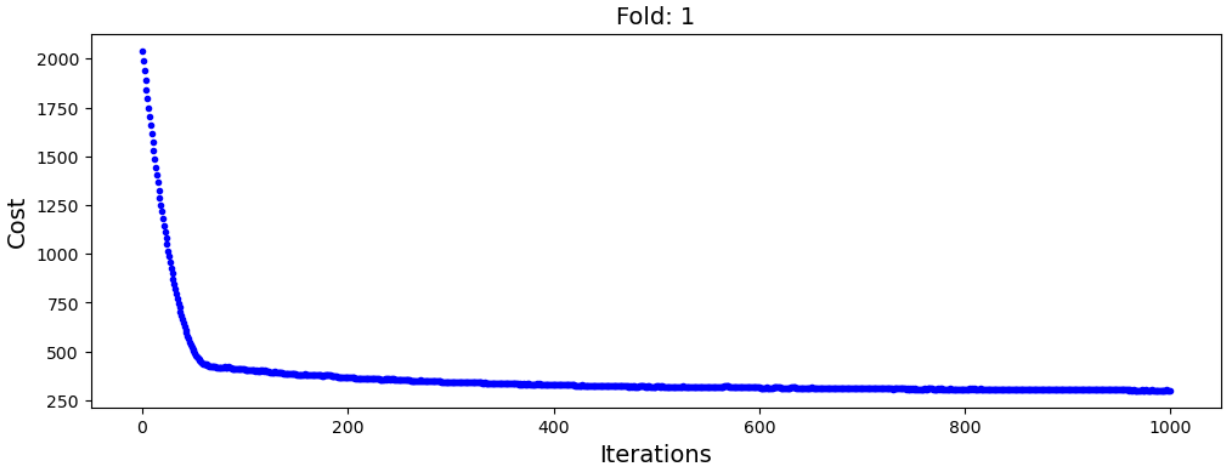
Trial two: Batch size - 52 , learning rate= 0.01

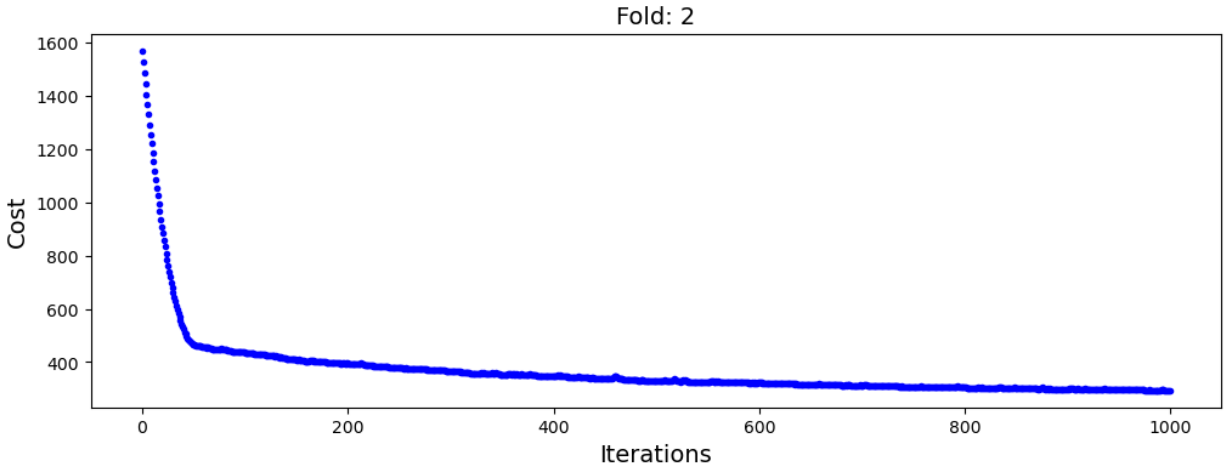
In [65]:
```
i = 1
gradient_clip_threshold = 1.0  # Adjust this threshold as needed

for train_index, val_index in kf.split(X_train):
    X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]

    # Add bias column to fold data
    X_train_fold = np.c_[np.ones((len(X_train_fold), 1)), X_train_fold]
    X_val_fold = np.c_[np.ones((len(X_val_fold), 1)), X_val_fold]
    lr = 0.01
```

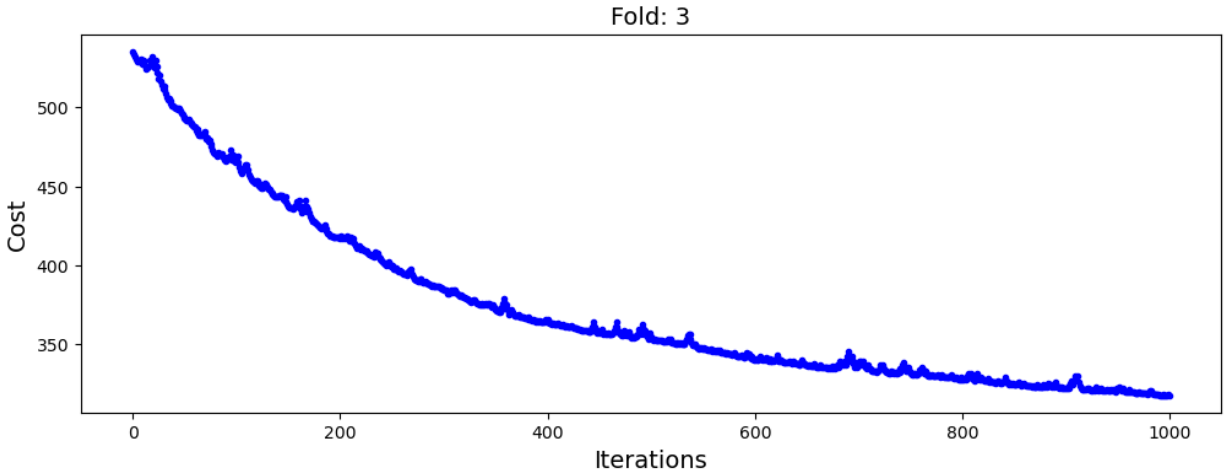
```
    n_iter = 1000
    batch_size = 52
    theta = np.random.randn(21, 1)  # Initialize coefficients (matching the nu

    # Train the model using minibatch gradient descent with gradient clipping
    theta, cost_history = minibatch_gradient_descent(X_train_fold, y_train_fol

    print('Predictions for fold:', i)
    print('Intercept value:', theta[0, 0], '\nWeight values:', theta[1:])

    # Use the validation data and mean square error to evaluate performance
    N = len(y_val_fold)
    y_hat = np.dot(X_val_fold, theta)
    mse_val = (1 / N) * (np.sum(np.square(y_hat - y_val_fold)))
    print('Mean Square Error:', round(mse_val, 3), '\n')

    fig, ax = plt.subplots(figsize=(12, 4))
    plt.title('Fold: ' + str(i))
    ax.set_ylabel('Cost')
    ax.set_xlabel('Iterations')
    _ = ax.plot(range(n_iter), cost_history, 'b.')

    i += 1
```

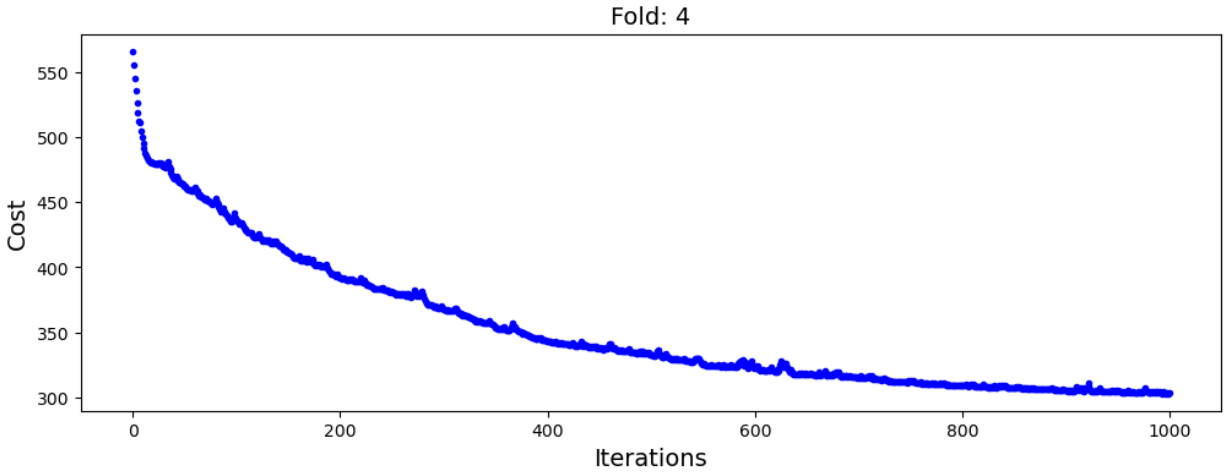
```
Predictions for fold: 1
Intercept value: 3.4823756213925043
Weight values: [[ 3.78999689]
 [-0.34474204]
 [ 0.03953982]
 [ 0.39779175]
 [-0.09431354]
 [ 1.98932566]
 [ 1.65966839]
 [ 4.8112342 ]
 [-0.57821786]
 [ 6.0267099 ]
 [ 2.00109905]
 [ 1.57671159]
 [-1.81165885]
 [ 0.14417832]
 [ 4.75197115]
 [ 0.4759259 ]
 [ 3.30082506]
 [ 1.82718247]
 [ 3.28419102]
 [-1.91181558]]
Mean Square Error: 286.675

Predictions for fold: 2
Intercept value: 2.8613417343115897
Weight values: [[ 2.43034603]
 [ 0.93410583]
 [ 0.20073323]
 [ 1.85310293]
 [-0.28036003]
 [-0.08395264]
 [ 1.90223986]
 [ 4.30853368]
 [-0.93119937]
 [ 5.53059959]
 [ 1.48199036]
 [ 1.465059  ]
 [-2.41581859]
 [ 0.89735998]
 [ 5.59734549]
 [ 0.49459913]
 [ 2.64852527]
 [-0.56952782]
 [ 4.28664903]
 [ 0.20002385]]
Mean Square Error: 353.234

Predictions for fold: 3
Intercept value: 2.5804760946672562
Weight values: [[ 2.50702025]
 [ 0.62463466]
 [ 0.14372769]
 [ 1.50134999]
 [-0.10825506]
 [ 2.07596491]
 [ 0.37851841]
 [ 2.95488217]
 [-1.77014006]
 [ 5.1419652 ]
```

```
 [ 0.83904862]
 [ 1.41960905]
 [-1.57267952]
 [ 1.90852722]
 [ 6.10221483]
 [ 0.27904599]
 [ 3.04646562]
 [ 0.81241065]
 [ 2.86802356]
 [-0.68818648]]
Mean Square Error: 291.855

Predictions for fold: 4
Intercept value: 2.286167745586501
Weight values: [[ 3.70689392]
 [-0.03052489]
 [ 0.13554294]
 [ 0.79352385]
 [-0.13797441]
 [ 0.1716873 ]
 [ 0.37493373]
 [ 3.14480487]
 [-1.690737  ]
 [ 5.26813774]
 [ 0.79596542]
 [ 1.50503954]
 [-1.88874597]
 [ 1.493646  ]
 [ 6.6337436 ]
 [-0.12010443]
 [ 3.7447225 ]
 [ 0.35760382]
 [ 2.88291882]
 [-0.99571384]]
Mean Square Error: 285.678
```









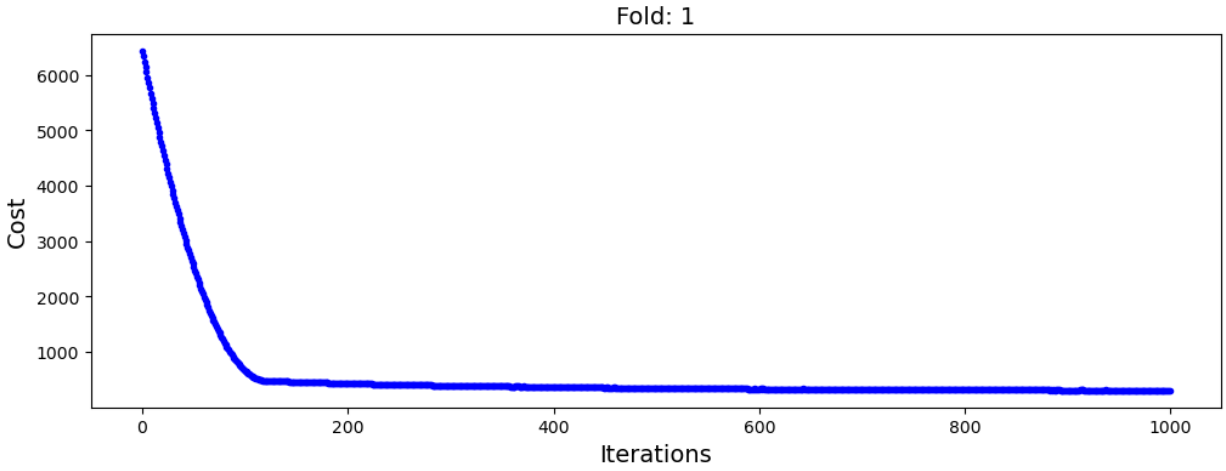
Trail 3: Batch Size - 68, Learning rate - 0.01

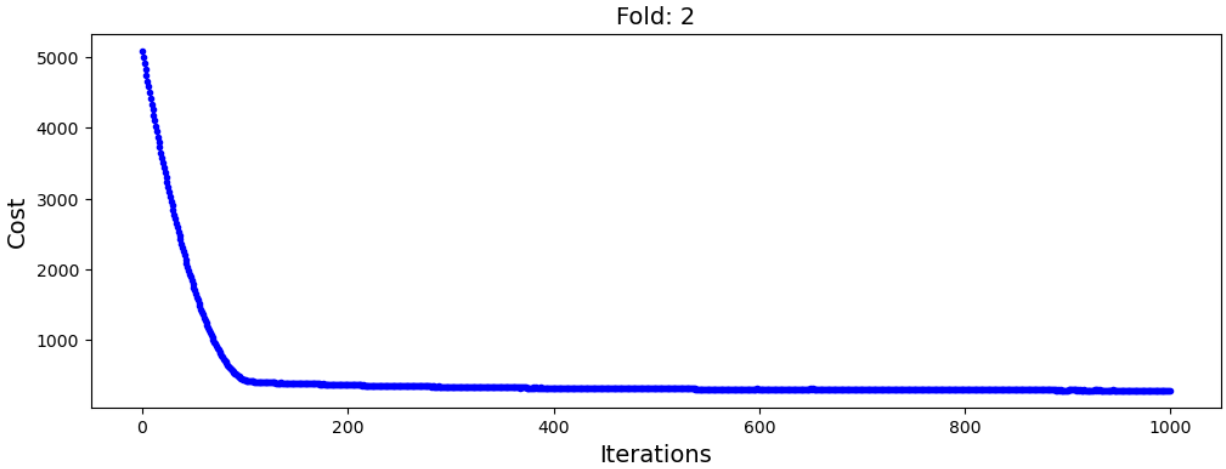
In [66]:
```
i = 1
gradient_clip_threshold = 1.0  # Adjust this threshold as needed

for train_index, val_index in kf.split(X_train):
    X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]

    # Add bias column to fold data
    X_train_fold = np.c_[np.ones((len(X_train_fold), 1)), X_train_fold]
    X_val_fold = np.c_[np.ones((len(X_val_fold), 1)), X_val_fold]
```

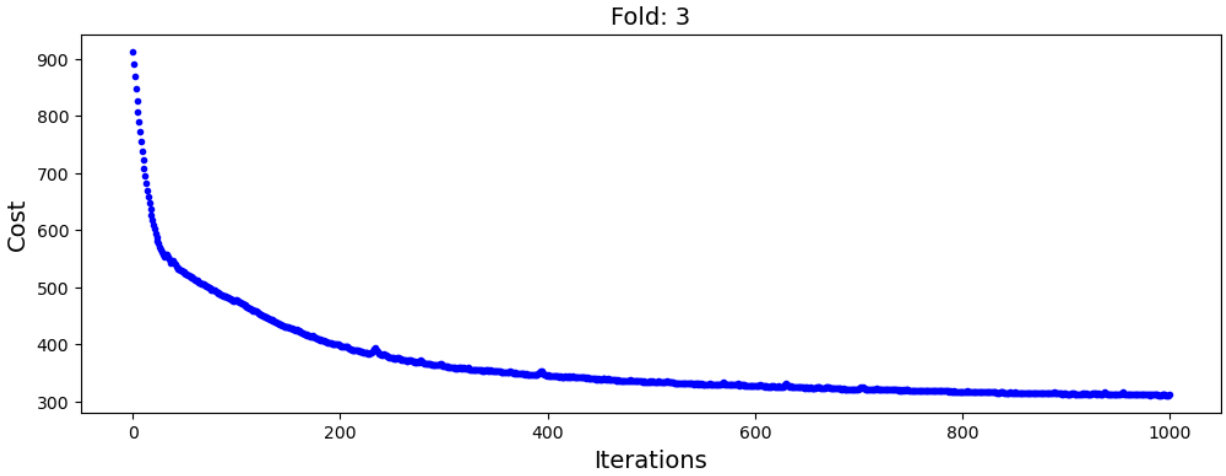
```
lr = 0.01
n_iter = 1000
batch_size = 68
theta = np.random.randn(21, 1)  # Initialize coefficients (matching the nu

# Train the model using minibatch gradient descent with gradient clipping
theta, cost_history = minibatch_gradient_descent(X_train_fold, y_train_fol

print('Predictions for fold:', i)
print('Intercept value:', theta[0, 0], '\nWeight values:', theta[1:])

# Use the validation data and mean square error to evaluate performance
N = len(y_val_fold)
y_hat = np.dot(X_val_fold, theta)
mse_val = (1 / N) * (np.sum(np.square(y_hat - y_val_fold)))
print('Mean Square Error:', round(mse_val, 3), '\n')

fig, ax = plt.subplots(figsize=(12, 4))
plt.title('Fold: ' + str(i))
ax.set_ylabel('Cost')
ax.set_xlabel('Iterations')
_ = ax.plot(range(n_iter), cost_history, 'b.')

i += 1
```

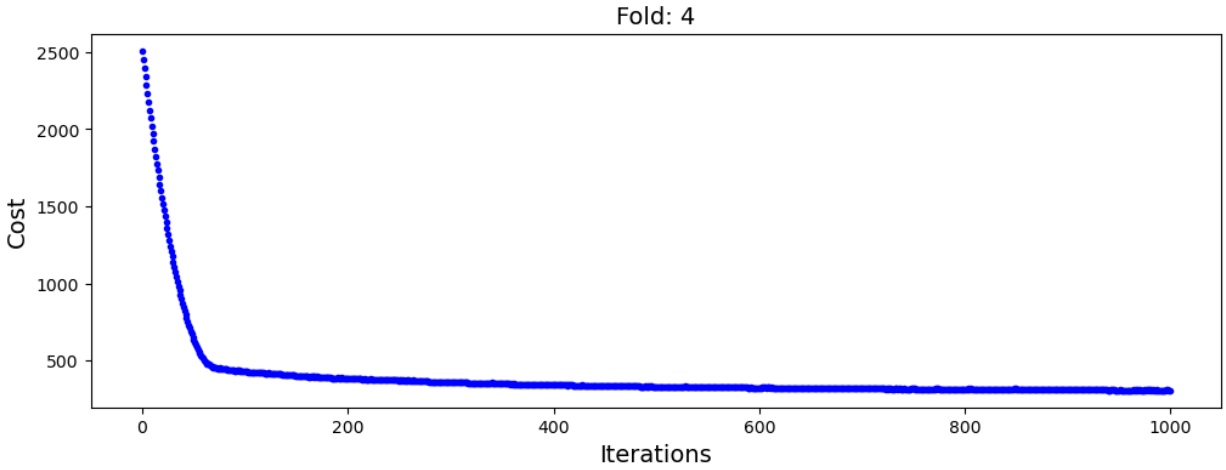
```
Predictions for fold: 1
Intercept value: 2.4845864500559163
Weight values: [[ 2.76896161]
 [ 1.79893079]
 [ 0.18616859]
 [ 0.41256697]
 [-0.09772892]
 [ 1.38816851]
 [ 0.38260693]
 [ 3.4697551 ]
 [ 0.27390833]
 [ 5.79058214]
 [ 1.14369616]
 [ 1.95908014]
 [-2.10753723]
 [ 1.43925294]
 [ 5.98434825]
 [ 1.36260253]
 [ 3.74001184]
 [ 2.18186081]
 [ 4.09681512]
 [ 0.47748894]]
Mean Square Error: 294.045

Predictions for fold: 2
Intercept value: 4.325823179403423
Weight values: [[ 2.89843261]
 [ 0.71814976]
 [ 0.0356586 ]
 [ 1.23941503]
 [-0.21656822]
 [ 1.00402597]
 [ 1.31142613]
 [ 3.88200914]
 [-0.67924255]
 [ 6.42343846]
 [ 2.577098  ]
 [ 2.51435793]
 [-1.19169275]
 [ 1.51469716]
 [ 5.85745798]
 [ 2.23195009]
 [ 4.85406999]
 [ 1.25139069]
 [ 2.77759661]
 [-0.96546652]]
Mean Square Error: 344.051

Predictions for fold: 3
Intercept value: 2.028108478993636
Weight values: [[ 3.16828489]
 [-0.04196573]
 [ 0.09589617]
 [ 1.18465383]
 [-0.23843413]
 [ 1.82136131]
 [-0.49459665]
 [ 1.93203125]
 [-1.45349884]
 [ 5.8057623 ]
```

```
 [ 1.83684856]
 [ 1.16441507]
 [-2.58949655]
 [ 1.54101899]
 [ 5.80766086]
 [ 0.04187309]
 [ 4.3957649 ]
 [ 1.87685349]
 [ 3.09796462]
 [-0.60247152]]
Mean Square Error: 285.609

Predictions for fold: 4
Intercept value: 1.4794048050752941
Weight values: [[ 3.5581701 ]
 [-0.18438851]
 [ 0.04240365]
 [ 1.45923382]
 [-0.24471314]
 [ 1.665232  ]
 [ 1.24990659]
 [ 3.88797075]
 [-0.59139468]
 [ 6.0706068 ]
 [ 1.78854837]
 [ 1.47416459]
 [-1.22497296]
 [-0.01571208]
 [ 3.83539071]
 [-0.92819044]
 [ 1.01582712]
 [-0.48504501]
 [ 4.96550712]
 [ 0.27014088]]
Mean Square Error: 293.444
```









From the above graphs, we can observe that as the batch size increases, the convergence of the cost function does not change to a large extent. However, if we have a lower batch size the average MSE tends to lower, thereby indicating that a better performance of the model.

While a lower batch would give us a better MSE, small batch sizes can be more susceptible to random fluctuations in the training data, while larger batch sizes are more resistant to these fluctuations but may converge more slowly. Hence, this trade-off needs to be kept in mind while deciding the batch size and learning rates. We can choose the ideal batch size through urther experimentation of the model.

For this model, the batch size of 20 and learning rate of 0.1 gives the best MSE value.

Ref: https://www.sabrepc.com/blog/Deep-Learning-and-AI/Epochs-Batch-Size-Iterations

Question F: Repeat the previous step with polynomial regression. Using validation loss, explore if your model overfits/underfits the data.

In [67]:
```
X_test = X_test.values
y_test = y_test.values
```

In [68]:
```
from sklearn.preprocessing import PolynomialFeatures

poly_degree = 2
poly_features = PolynomialFeatures(degree = poly_degree, include_bias=False)
```

In [69]:
```
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly  = poly_features.transform(X_test)
```

In [70]:
```
X_train_poly.shape
```

Out[70]:
```
(15083, 230)
```

Ref for polynomial: Code from chapter 2 of Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd edition).

In [71]:
```
#four-fold cross validation with polynomial regression

poly_performance_metrics = []

kf = KFold(n_splits=4)

for train_index, test_index in kf.split(X_train_poly):

    X_train_poly_kFold, X_test_poly_kFold = X_train_poly[train_index], X_train_
    y_train_kFold     , y_test_kFold      = y_train[train_index]     , y_train[

    # Fit the polynomial linear regression model on the training data
    poly_reg = LinearRegression()
    poly_reg.fit(X_train_poly_kFold, y_train_kFold)

    # Make predictions on the test data
    y_pred_kFold = poly_reg.predict(X_test_poly_kFold)

    # Calculate the Mean Squared Error for this iteration
    mse_poly = mean_squared_error(y_test_kFold, y_pred_kFold)

    # Store the performance metric
    poly_performance_metrics.append(mse_poly)

# Calculate the average performance metric across all iterations
poly_average_performance = np.mean(poly_performance_metrics)
```

In [72]:
```
poly_average_performance
```

Out[72]:
```
194.2745215882449
```

In [73]:
```
# As asked in the question, we would be running a four-fold valiadation, hence
k = 4

# Storing the theta values for each fold
theta_values_poly = []

# Initialize KFold cross-validator
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Initialize an empty list to store the MSE values for each fold
mse_values_poly = []

# Perform k-fold cross-validation
for train_index, val_index in kf.split(X_train_poly_kFold):
    X_train_fold_poly, X_val_fold_poly = X_train_poly_kFold[train_index], X_tra
    y_train_fold_poly, y_val_fold_poly = y_train_kFold[train_index], y_train_kF

    # Calculate the coefficients using the normal equation for each fold
    X_transpose_poly = np.transpose(X_train_fold_poly)
    X_transpose_X_poly = np.dot(X_transpose_poly, X_train_fold_poly)
    X_transpose_X_inv_poly = np.linalg.pinv(X_transpose_X_poly)
    X_transpose_y_poly = np.dot(X_transpose_poly, y_train_fold_poly)
    theta_poly = np.dot(X_transpose_X_inv_poly, X_transpose_y_poly)

    theta_values_poly.append(theta_poly)

    # Make predictions using the calculated theta values
    y_val_pred_poly = np.dot(X_val_fold_poly, theta_poly)

    # Calculate and append the MSE for this fold
    mse_fold = mean_squared_error(y_val_fold_poly, y_val_pred_poly)
    mse_values_poly.append(mse_fold)

# Calculate the average coefficients (theta) across all folds
average_theta_poly = np.mean(theta_values_poly, axis=0)

# Calculate the average MSE across all folds
average_mse_poly = np.mean(mse_values_poly)

print("Average Coefficients (theta) across all folds:", average_theta_poly)
print("Average MSE across all folds:", average_mse_poly)
```

```
Average Coefficients (theta) across all folds: [[-4.49764335e+00]
 [ 1.31874771e+01]
 [-3.99880034e-01]
 [-2.04955570e+00]
 [ 2.47745887e+00]
 [-1.31592979e+01]
 [ 1.32404103e+01]
 [ 1.95282414e+01]
 [ 1.90550226e+01]
 [ 2.75595764e+01]
 [-8.21034996e+00]
 [ 1.32176339e+00]
 [-6.95729138e+00]
 [ 1.99598697e+01]
 [ 1.28089363e+01]
 [ 1.04031176e+01]
 [ 5.26185089e+00]
 [ 1.71038931e+01]
 [ 8.62255979e+00]
 [ 2.41462192e+01]
 [ 1.19738539e+00]
 [-3.92026336e+00]
 [ 1.22641794e-01]
 [-2.19433438e-01]
 [-7.56662537e-01]
 [-1.74136502e+00]
 [-1.40951341e+00]
 [-3.08810345e+00]
 [-7.07449249e+00]
 [-7.45703820e+00]
 [ 4.89243686e+00]
 [ 1.61055136e+00]
 [ 3.53096076e+00]
 [-4.94877336e+00]
 [ 4.51145921e-01]
 [-7.76982014e-01]
 [-2.19943804e-01]
 [-3.50070329e+00]
 [ 1.54128990e+00]
 [-6.03891260e+00]
 [ 5.63902361e-01]
 [-3.17538647e-02]
 [-9.75654429e-01]
 [-4.96874072e-01]
 [ 2.43273319e+00]
 [ 6.82797710e+00]
 [ 6.35988890e+00]
 [ 1.35649939e-01]
 [ 1.36185786e+01]
 [-1.55093618e+00]
 [ 5.13175928e+00]
 [-4.14718526e+00]
 [ 3.00265587e+00]
 [ 1.01852104e+01]
 [-3.07414195e+00]
 [-2.59708496e-01]
 [ 1.65217170e+01]
 [ 5.25509926e+00]
 [ 7.93276681e+00]
 [-8.24510426e-04]
```

```
[ 2.65915515e-02]
[-1.14070695e-03]
[ 1.49524005e-01]
[-1.53637627e-01]
[-2.46218935e-01]
[-7.36145718e-02]
[ 1.63971980e-02]
[-1.93371114e-01]
[-2.55662987e-02]
[-1.23710030e-01]
[-1.69412996e-01]
[-2.30456107e-01]
[-1.64891640e-01]
[-1.02400946e-01]
[-1.32562572e-01]
[-2.88472885e-01]
[-1.11401339e-01]
[ 9.63712224e-02]
[ 4.23641771e-01]
[ 1.32101314e+00]
[-1.05072650e+00]
[-9.98641078e-01]
[-2.82715089e-01]
[-2.12652965e+00]
[ 8.89900275e-01]
[-6.49722874e-01]
[ 1.19699310e-01]
[-1.04982564e+00]
[-9.99544615e-01]
[ 1.87023221e+00]
[-7.95702843e-02]
[-3.84003197e+00]
[-8.98119333e-01]
[-1.15124722e+00]
[-2.95071437e-01]
[-6.54967888e-02]
[ 1.03935079e+00]
[ 1.43810237e+00]
[-4.09881566e-02]
[ 6.96334616e-03]
[ 9.60811489e-01]
[ 4.72932374e-02]
[ 1.50336836e+00]
[ 3.75993451e-01]
[ 2.10145765e+00]
[ 1.61945898e-01]
[ 8.59576985e-01]
[ 1.45593028e+00]
[ 1.23080413e+00]
[ 1.24665106e+00]
[ 3.08027783e+00]
[-5.29226884e+00]
[-7.86708567e+00]
[-3.25780624e+00]
[-1.68558163e+01]
[ 4.33297717e+00]
[-2.28992710e+00]
[ 4.91121717e+00]
[-1.73782666e+00]
[-1.14215284e+01]
```

```
[-5.15290881e+00]
[-6.79801053e-02]
[-7.93846612e+00]
[-4.01467861e+00]
[-9.14467725e+00]
[ 1.32404860e+01]
[ 0.00000000e+00]
[ 8.33786313e+00]
[ 1.24988117e+01]
[-6.93231632e+00]
[ 1.11636002e+00]
[-1.78023237e+00]
[ 8.92360723e+00]
[ 4.31687876e+00]
[ 3.83699029e+00]
[ 7.56003133e-01]
[ 8.64749260e+00]
[ 4.26824384e+00]
[ 8.97224226e+00]
[ 1.95282871e+01]
[ 1.07171995e+01]
[ 1.50608123e+01]
[-1.27801168e+00]
[ 2.05365108e-01]
[-5.17707810e+00]
[ 1.10362548e+01]
[ 8.49203239e+00]
[ 6.56609207e+00]
[ 4.50585500e+00]
[ 8.45633999e+00]
[ 4.35430189e+00]
[ 1.51739852e+01]
[ 1.90550626e+01]
[ 0.00000000e+00]
[ 0.00000000e+00]
[ 0.00000000e+00]
[ 0.00000000e+00]
[ 1.07691088e+01]
[ 8.28595372e+00]
[ 4.74152757e+00]
[ 2.17576086e+00]
[ 1.21377741e+01]
[ 9.29595314e+00]
[ 9.75910945e+00]
[ 2.75596240e+01]
[ 0.00000000e+00]
[ 0.00000000e+00]
[ 0.00000000e+00]
[ 1.20005740e+01]
[ 1.55590499e+01]
[ 9.82402916e+00]
[ 6.01745855e+00]
[ 1.17181363e+01]
[ 1.17113279e+01]
[ 1.58482961e+01]
[-8.21032798e+00]
[ 0.00000000e+00]
[ 0.00000000e+00]
[-2.30268742e-01]
[-7.98005926e+00]
```

```
 [-1.61224240e+00]
 [-2.75950613e+00]
 [-3.83857946e+00]
 [-5.69531184e+00]
 [-2.51501616e+00]
 [ 1.32172507e+00]
 [ 0.00000000e+00]
 [ 7.15118514e-01]
 [ 6.06606508e-01]
 [ 1.11414191e+00]
 [ 1.05099098e+00]
 [-8.43407864e-01]
 [ 3.95605216e-01]
 [ 9.26119644e-01]
 [-6.95731059e+00]
 [-3.29467077e+00]
 [-3.66263987e+00]
 [-3.66437389e+00]
 [-1.22284613e+00]
 [-2.07009051e+00]
 [-7.08502897e+00]
 [ 1.27718462e-01]
 [ 1.99598624e+01]
 [ 0.00000000e+00]
 [ 6.51819493e+00]
 [ 1.94077918e+00]
 [ 1.15008880e+01]
 [ 6.21709037e+00]
 [ 1.37427718e+01]
 [ 1.28089111e+01]
 [ 3.88488742e+00]
 [ 3.32107900e+00]
 [ 5.60294464e+00]
 [ 2.40545550e+00]
 [ 1.04034556e+01]
 [ 1.04030824e+01]
 [ 0.00000000e+00]
 [ 0.00000000e+00]
 [ 1.78916599e+00]
 [ 8.61391638e+00]
 [ 5.26185824e+00]
 [ 0.00000000e+00]
 [ 4.63552741e-01]
 [ 4.79830538e+00]
 [ 1.71038326e+01]
 [ 6.36982690e+00]
 [ 1.07340057e+01]
 [ 8.62254580e+00]
 [ 0.00000000e+00]
 [ 2.41462274e+01]]
Average MSE across all folds: 181.80639693953248
```

Observing the MSE of polynomial regression using OLS and closed-form, the evaluation model has improved as compared to the linear regression model. The improvization of the model would be due to the existence of non-linear data within the dataset.

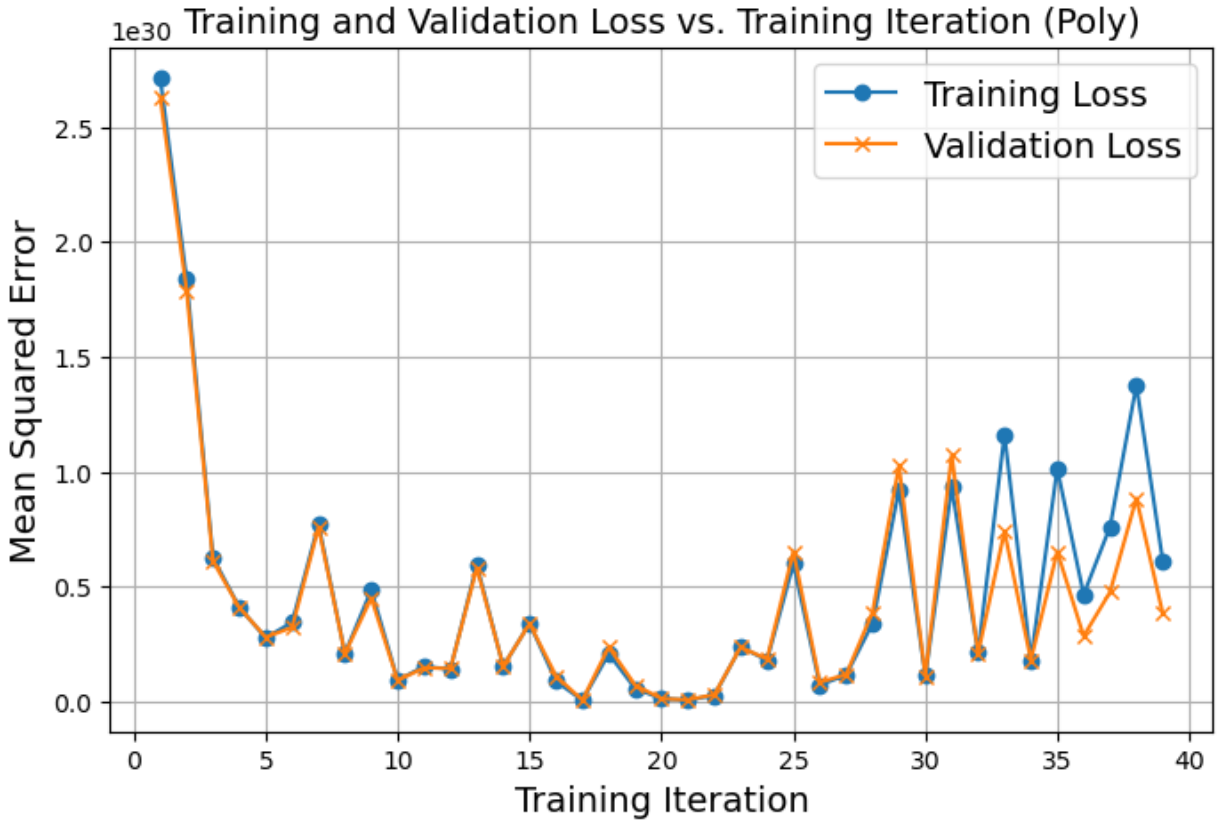
Polynomial regression uses linear model to fit non-linear data.

Ref: Code from chapter 2 of Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd edition).

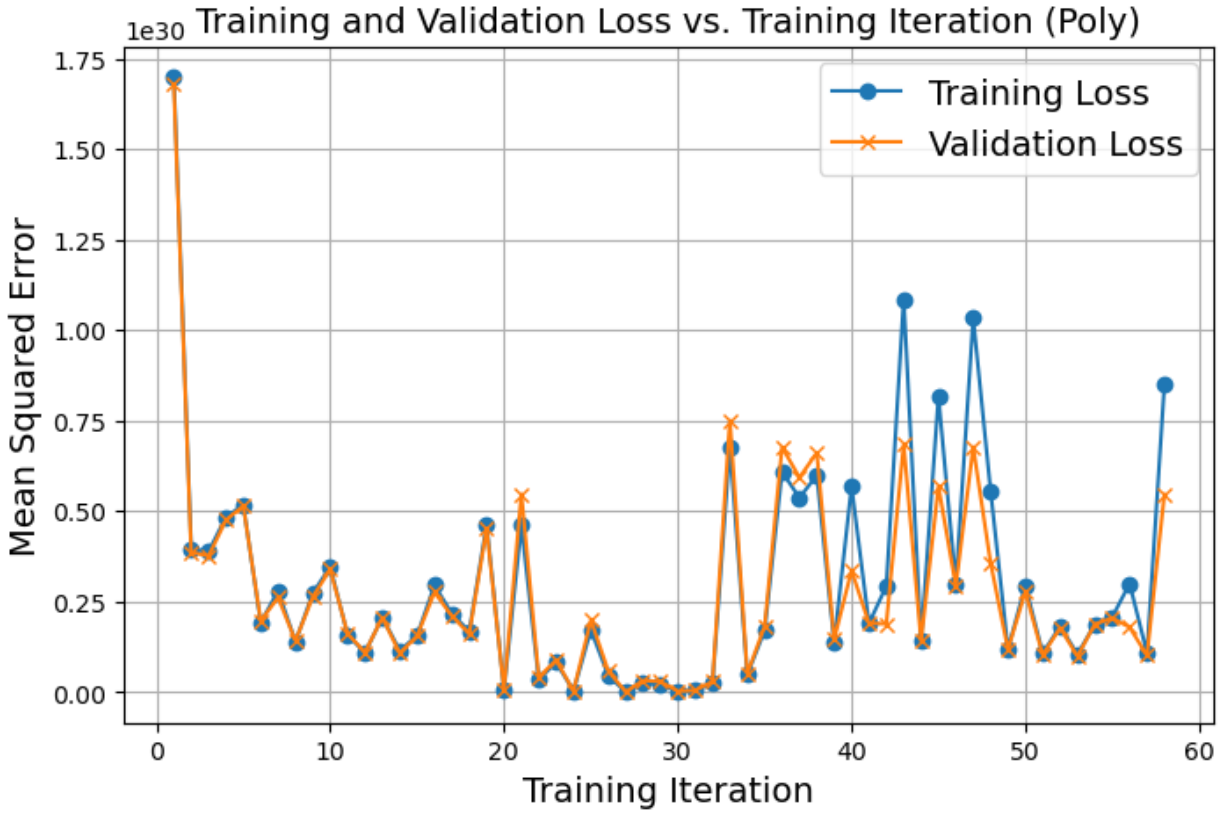
In [74]:
```
import warnings
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold

warnings.filterwarnings("ignore")

# Define the number of folds (k)
k = 4

# Create the SGDRegressor model with appropriate hyperparameters
alphaList_poly = [ 0.01, 0.1,0]
for alpha_poly in alphaList_poly:
    sgd_model_poly = SGDRegressor(loss='squared_error', alpha=alpha_poly, max_

    # Initialize lists to store training and validation loss
    training_loss_poly = []
    validation_loss_poly = []

    # Initialize KFold cross-validator
    kf_poly = KFold(n_splits=k, shuffle=True, random_state=42)

    for train_index_poly, val_index_poly in kf_poly.split(X_train_poly):
        X_train_fold_poly, X_val_fold_poly = X_train_poly[train_index_poly], X_
        y_train_fold_poly, y_val_fold_poly = y_train[train_index_poly], y_trai

        # Number of training iterations
        n_iterations_poly = 100

        # Early stopping parameters
        early_stopping_rounds_poly = 5  # Number of iterations with no improver
        best_val_loss_poly = float('inf')  # Initialize the best validation los
        no_improvement_count_poly = 0  # Initialize the count of iterations wi

        for iteration_poly in range(n_iterations_poly):
            # Fit the model for one iteration (one pass through the training da
            sgd_model_poly.partial_fit(X_train_fold_poly, y_train_fold_poly)

            # Predict on the training data
            y_train_pred_poly = sgd_model_poly.predict(X_train_fold_poly)

            # Calculate training loss (Mean Squared Error) and append to the l
            train_loss_poly = mean_squared_error(y_train_fold_poly, y_train_pre
            training_loss_poly.append(train_loss_poly)

            # Predict on the validation data
            y_val_pred_poly = sgd_model_poly.predict(X_val_fold_poly)

            # Calculate validation loss (Mean Squared Error) and append to the
            val_loss_poly = mean_squared_error(y_val_fold_poly, y_val_pred_poly
            validation_loss_poly.append(val_loss_poly)

            # Check for early stopping
            if val_loss_poly < best_val_loss_poly:
```

```
                best_val_loss_poly = val_loss_poly
                no_improvement_count_poly = 0
            else:
                no_improvement_count_poly += 1

            if no_improvement_count_poly >= early_stopping_rounds_poly:
                print(f"Early stopping at iteration {iteration_poly + 1} due to
                break

    # Plot training and validation loss as a function of training iteration
    plt.figure(figsize=(8, 5))
    plt.plot(range(1, len(training_loss_poly) + 1), training_loss_poly, label=
    plt.plot(range(1, len(validation_loss_poly) + 1), validation_loss_poly, lab
    plt.xlabel('Training Iteration')
    plt.ylabel('Mean Squared Error')
    plt.title('Training and Validation Loss vs. Training Iteration (Poly)')
    plt.legend()
    plt.grid()
    plt.show()

    # Final model evaluation
    y_pred_sgd_poly = sgd_model_poly.predict(X_test_poly)
    final_mse_poly = mean_squared_error(y_test, y_pred_sgd_poly)
    print("Final Mean Squared Error (SGD) (Poly):", final_mse_poly)

warnings.resetwarnings()
```

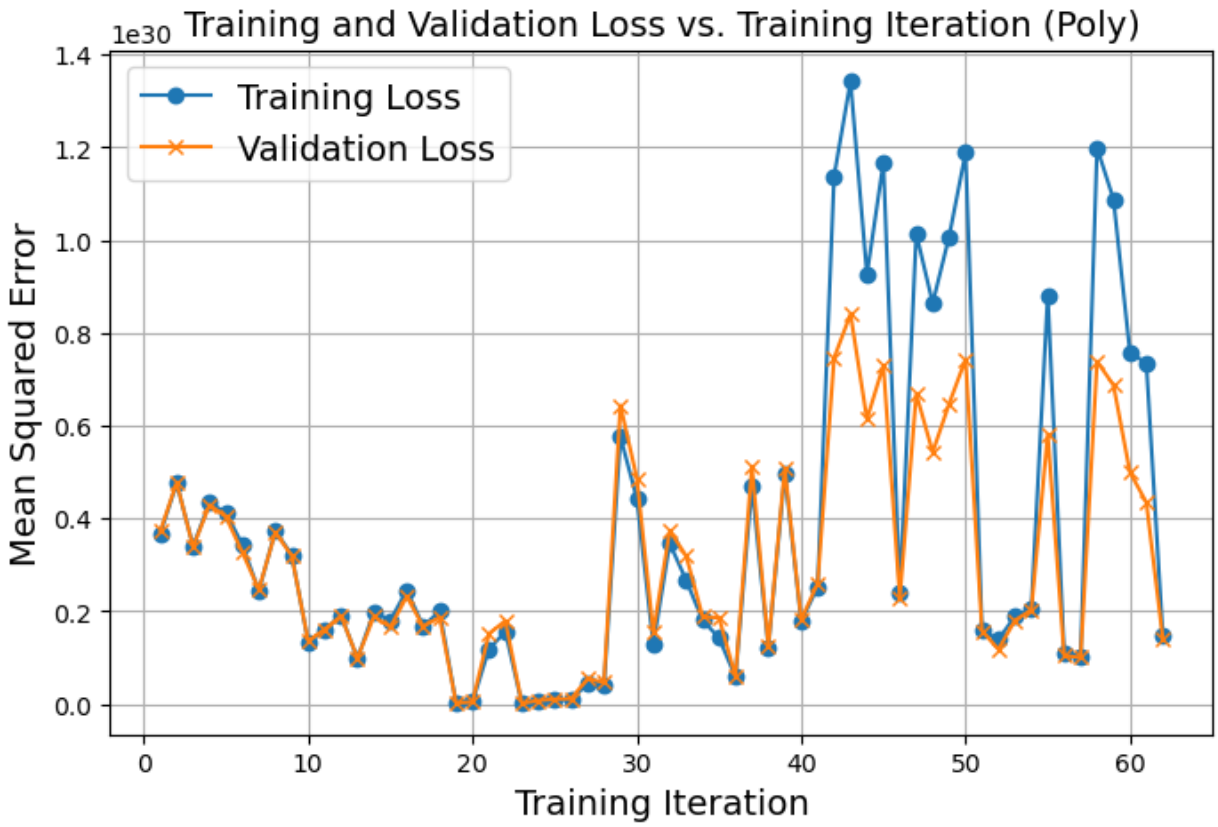
```
Early stopping at iteration 15 due to no improvement in validation loss.
Early stopping at iteration 7 due to no improvement in validation loss.
Early stopping at iteration 9 due to no improvement in validation loss.
Early stopping at iteration 8 due to no improvement in validation loss.
```



```
Final Mean Squared Error (SGD) (Poly): 8.299887368512644e+29
Early stopping at iteration 19 due to no improvement in validation loss.
Early stopping at iteration 13 due to no improvement in validation loss.
Early stopping at iteration 7 due to no improvement in validation loss.
Early stopping at iteration 19 due to no improvement in validation loss.
```




```
Final Mean Squared Error (SGD) (Poly): 1.1367460211060655e+30
Early stopping at iteration 18 due to no improvement in validation loss.
Early stopping at iteration 10 due to no improvement in validation loss.
Early stopping at iteration 13 due to no improvement in validation loss.
Early stopping at iteration 21 due to no improvement in validation loss.
```




```
Final Mean Squared Error (SGD) (Poly): 1.5417238272802845e+29
```

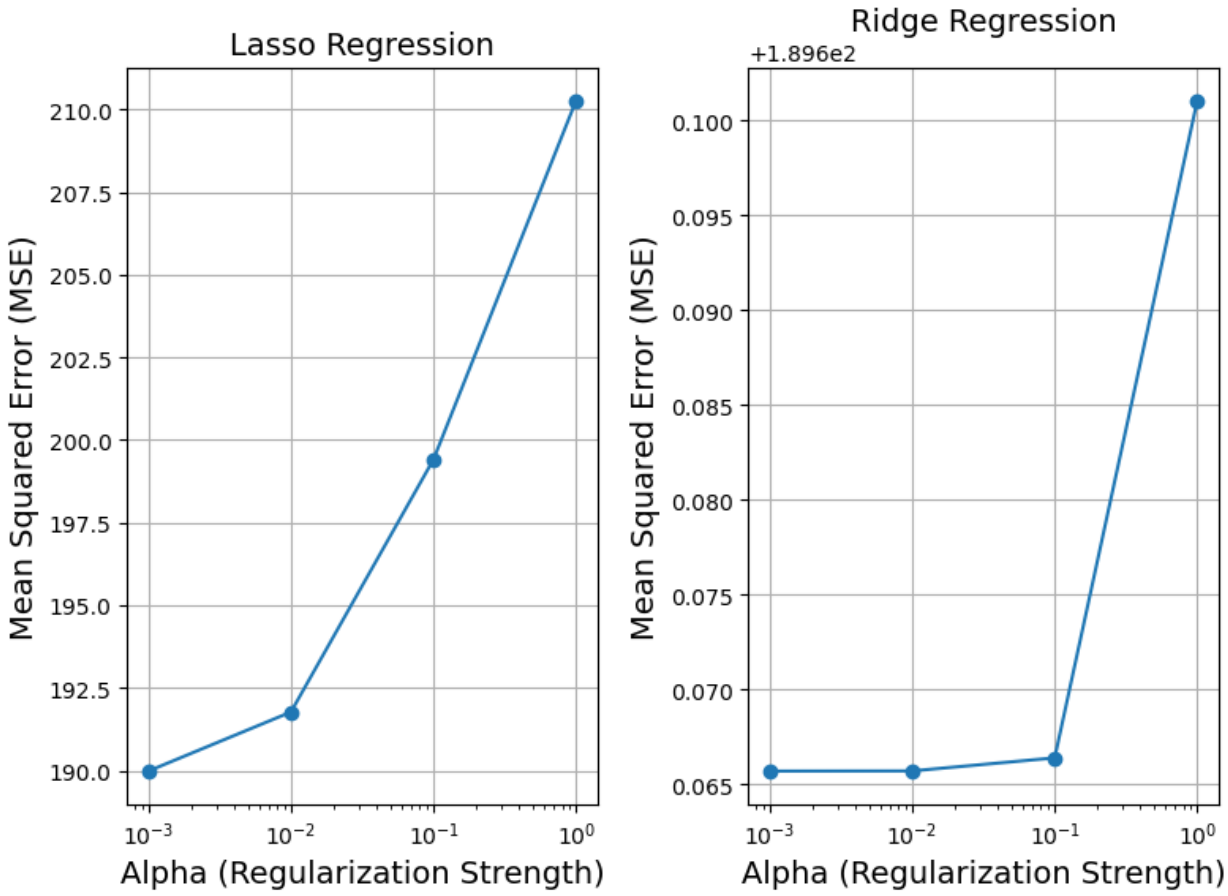
The above graph indicates that the model is overfitting the data. Additionally, comparing the MSE between linear and polynomial regression, we can observe that linear regression performs better. Further, the graphical representation of training and validation loss for the linear model indicated that the model is a better fit as compared the training and validation loss for polynomial regression. One of the reasons for this could be that the updates in SGD are noisy and have a high variance, which can make the optimization process less stable.

Ref: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

In [75]:
```
from sklearn.linear_model import Lasso, Ridge
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

warnings.filterwarnings("ignore")

# Define a list of alpha values to try
alphas = [0.001, 0.01, 0.1, 1.0]

# Initialize empty lists to store the MSE values
lasso_mse_values_poly = []
ridge_mse_values_poly = []

for alpha in alphas:
    lasso_reg_poly = Lasso(alpha=alpha)
    ridge_reg_poly = Ridge(alpha=alpha)

    # Fit the Lasso model
    lasso_reg_poly.fit(X_train_poly, y_train)
```

```
    # Make predictions with the trained Lasso model
    lasso_predictions_poly = lasso_reg_poly.predict(X_train_poly)

    # Calculate MSE for Lasso
    lasso_mse_poly = mean_squared_error(y_train, lasso_predictions_poly)
    lasso_mse_values_poly.append(lasso_mse_poly)

    # Fit the Ridge model
    ridge_reg_poly.fit(X_train_poly, y_train)

    # Make predictions with the trained Ridge model
    ridge_predictions_poly = ridge_reg_poly.predict(X_train_poly)

    # Calculate MSE for Ridge
    ridge_mse_poly = mean_squared_error(y_train, ridge_predictions_poly)
    ridge_mse_values_poly.append(ridge_mse_poly)

    # Print the MSE values for each alpha
    print(f"Alpha = {alpha}")
    print("Lasso Regression Train MSE:", lasso_mse_poly)
    print("Ridge Regression Train MSE:", ridge_mse_poly)
    print("===")

# Create individual plots for each regularization technique
plt.figure(figsize=(12, 6))

# Lasso Regression Plot
plt.subplot(131)
plt.semilogx(alphas, lasso_mse_values_poly, marker='o')
plt.xlabel('Alpha (Regularization Strength)')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Lasso Regression')
plt.grid(True)

# Ridge Regression Plot
plt.subplot(132)
plt.semilogx(alphas, ridge_mse_values_poly, marker='o')
plt.xlabel('Alpha (Regularization Strength)')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Ridge Regression')
plt.grid(True)

plt.tight_layout()
plt.show()

warnings.resetwarnings()
```

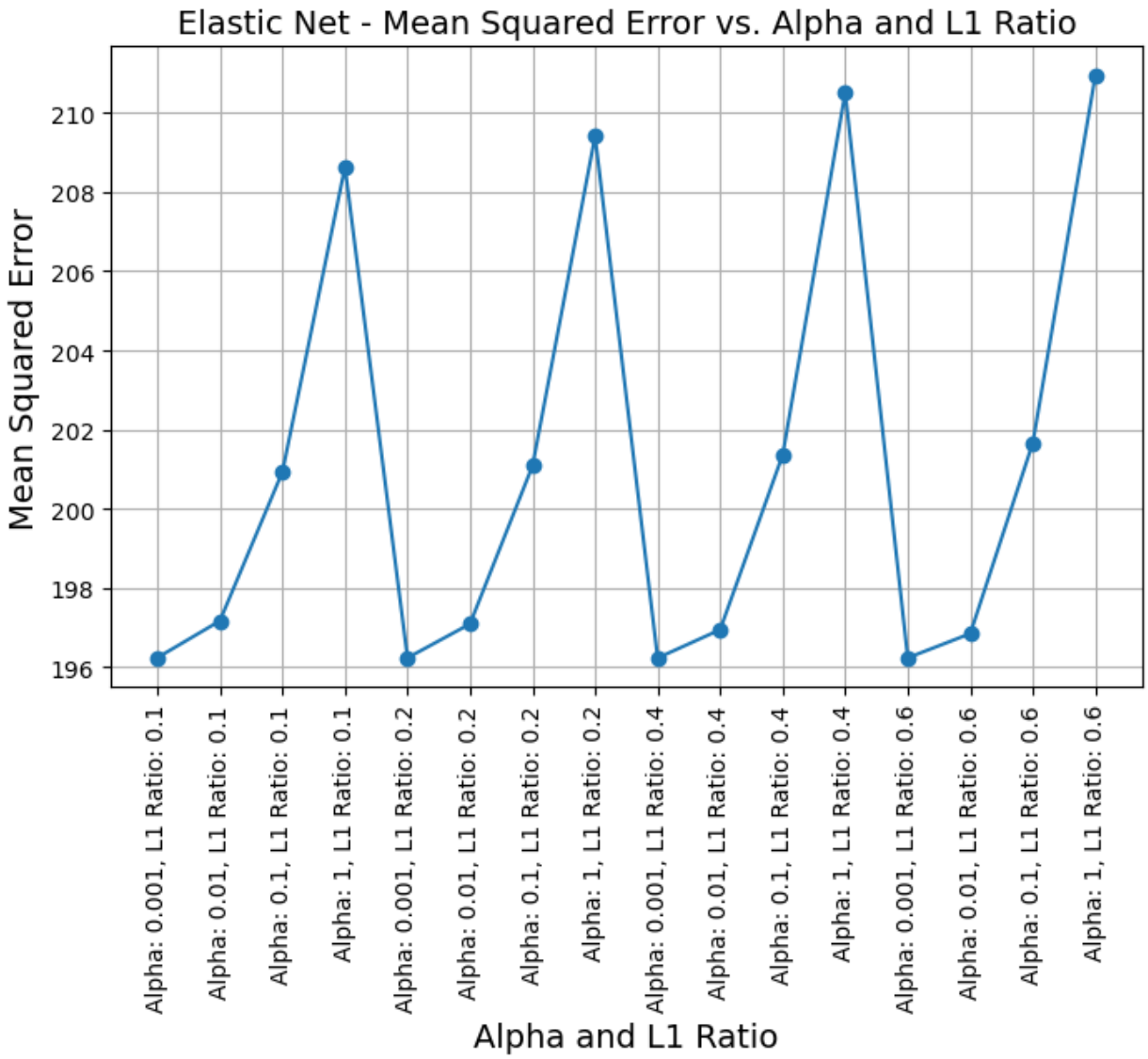
```
Alpha = 0.001
Lasso Regression Train MSE: 189.98092079375363
Ridge Regression Train MSE: 189.6656876338909
===
Alpha = 0.01
Lasso Regression Train MSE: 191.76893835778452
Ridge Regression Train MSE: 189.66569524334707
===
Alpha = 0.1
Lasso Regression Train MSE: 199.41273474379224
Ridge Regression Train MSE: 189.66638471378084
===
Alpha = 1.0
Lasso Regression Train MSE: 210.27029011135812
Ridge Regression Train MSE: 189.70105232503423
===
```



In [76]:
```
import warnings
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold

warnings.filterwarnings("ignore")

# Define the number of folds (k)
k = 4

# Create lists of alpha (regularization strength) and l1_ratio (L1 mixing para
alpha_list = [0.001, 0.01, 0.1, 1]
l1_ratio_list = [0.1, 0.2, 0.4, 0.6]  # Different L1 ratios for Elastic Net
```

```
alpha_l1_list = []  # Store alpha and l1_ratio combinations
elastic_mse_vals = []  # Store Elastic Net MSE values

for l1_ratio in l1_ratio_list:
    for alpha in alpha_list:
        elastic_net_model = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, max_iter

        # Initialize KFold cross-validator
        kf = KFold(n_splits=k, shuffle=True, random_state=42)

        fold_training_losses = []  # Store training losses for each fold
        fold_validation_losses = []  # Store validation losses for each fold

        for train_index, test_index in kf.split(X_train_poly):  # Use the polyn
            X_train_fold, X_val_fold = X_train_poly[train_index], X_train_poly[
            y_train_fold, y_val_fold = y_train[train_index], y_train[test_index

            # Fit the Elastic Net model
            elastic_net_model.fit(X_train_fold, y_train_fold)

            # Predict on the training data
            y_train_pred = elastic_net_model.predict(X_train_fold)

            # Calculate training loss (Mean Squared Error) and append to the li
            train_loss = mean_squared_error(y_train_fold, y_train_pred)
            fold_training_losses.append(train_loss)

            # Predict on the validation data
            y_val_pred = elastic_net_model.predict(X_val_fold)

            # Calculate validation loss (Mean Squared Error) and append to the
            val_loss = mean_squared_error(y_val_fold, y_val_pred)
            fold_validation_losses.append(val_loss)

        # Calculate the mean training and validation loss across folds for this
        mean_training_loss = np.mean(fold_training_losses)
        mean_validation_loss = np.mean(fold_validation_losses)

        alpha_l1_list.append((alpha, l1_ratio))
        elastic_mse_vals.append(mean_validation_loss)

        # Final model evaluation
        elastic_net_model.fit(X_train_poly, y_train)  # Fit on the entire polyn
        y_pred_elastic_net = elastic_net_model.predict(X_test_poly)  # Use poly
        final_mse = mean_squared_error(y_test, y_pred_elastic_net)
        print(f"Alpha: {alpha}, L1 Ratio: {l1_ratio}, Final Mean Squared Error

# Plot training and validation loss as a function of alpha
plt.figure(figsize=(8, 5))
plt.plot(range(len(alpha_l1_list)), elastic_mse_vals, marker='o')
plt.xticks(range(len(alpha_l1_list)), [f"Alpha: {alpha}, L1 Ratio: {l1_ratio}"
plt.xlabel('Alpha and L1 Ratio')
plt.ylabel('Mean Squared Error')
plt.title('Elastic Net - Mean Squared Error vs. Alpha and L1 Ratio')
plt.grid()
plt.show()

warnings.resetwarnings()
```

```
Alpha: 0.001, L1 Ratio: 0.1, Final Mean Squared Error (Elastic Net): 182.04112
214651147
Alpha: 0.01, L1 Ratio: 0.1, Final Mean Squared Error (Elastic Net): 183.588631
95096433
Alpha: 0.1, L1 Ratio: 0.1, Final Mean Squared Error (Elastic Net): 187.8210973
5918561
Alpha: 1, L1 Ratio: 0.1, Final Mean Squared Error (Elastic Net): 195.475077905
42667
Alpha: 0.001, L1 Ratio: 0.2, Final Mean Squared Error (Elastic Net): 182.01796
444057337
Alpha: 0.01, L1 Ratio: 0.2, Final Mean Squared Error (Elastic Net): 183.482647
0309031
Alpha: 0.1, L1 Ratio: 0.2, Final Mean Squared Error (Elastic Net): 187.9670079
7613832
Alpha: 1, L1 Ratio: 0.2, Final Mean Squared Error (Elastic Net): 196.101843255
37935
Alpha: 0.001, L1 Ratio: 0.4, Final Mean Squared Error (Elastic Net): 181.96676
559809575
Alpha: 0.01, L1 Ratio: 0.4, Final Mean Squared Error (Elastic Net): 183.264321
57045372
Alpha: 0.1, L1 Ratio: 0.4, Final Mean Squared Error (Elastic Net): 188.1327277
1029722
Alpha: 1, L1 Ratio: 0.4, Final Mean Squared Error (Elastic Net): 197.013887106
87235
Alpha: 0.001, L1 Ratio: 0.6, Final Mean Squared Error (Elastic Net): 181.91935
557706879
Alpha: 0.01, L1 Ratio: 0.6, Final Mean Squared Error (Elastic Net): 183.045972
24565546
Alpha: 0.1, L1 Ratio: 0.6, Final Mean Squared Error (Elastic Net): 188.2166466
288771
Alpha: 1, L1 Ratio: 0.6, Final Mean Squared Error (Elastic Net): 197.482854656
57548
```



Analyzing the MSE values across Ridge, Lasso and Elastic-Net, the model performs best at the lowest alpha value. However, the ideal model is also dependent on multi-collinearity of the dataset. While Elastic-Net needs to ideally be used along with Lasso and Ridge, Lasso and elastic-net doesnot work well with multi-collinear features. Since, multi-collinearity does not exists in this case, Lasso and elastic-net can be used as the regularization technique.

Ref: https://www.geeksforgeeks.org/lasso-vs-ridge-vs-elastic-net-ml/

In [77]:
```
X_train_poly.shape
```

Out[77]:
```
(15083, 230)
```

Trial one: Batch size - 20, learning rate - 0.01

In [78]:
```
i = 1
gradient_clip_threshold = 1.0  # Adjust this threshold as needed

for train_index, test_index in kf.split(X_train_poly):
    X_train_fold_poly, X_test_fold_poly = X_train_poly[train_index], X_train_po
```

```
    y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]

    # Add bias column to fold data
    X_train_fold_poly = np.c_[np.ones((len(X_train_fold_poly), 1)), X_train_fol
    X_test_fold_poly = np.c_[np.ones((len(X_test_fold_poly), 1)), X_test_fold_p

    lr = 0.01
    n_iter = 100
    batch_size = 20
    theta_poly = np.random.randn(231, 1)  # Initialize coefficients (matching t

    # Train the model using minibatch gradient descent with gradient clipping
    theta_poly, cost_history_poly = minibatch_gradient_descent(X_train_fold_pol

    print('Predictions for fold:', i)
    print('Intercept value:', theta_poly[0, 0], '\nWeight values:', theta_poly[

    # Use the validation data and mean square error to evaluate performance
    N = len(y_test_fold)
    y_hat_poly = np.dot(X_test_fold_poly, theta_poly)
    mse_val_poly = (1 / N) * (np.sum(np.square(y_hat_poly - y_test_fold)))
    print('Mean Square Error:', round(mse_val_poly, 3), '\n')

    fig, ax = plt.subplots(figsize=(12, 4))
    plt.title('Fold: ' + str(i))
    ax.set_ylabel('Cost')
    ax.set_xlabel('Iterations')
    _ = ax.plot(range(n_iter), cost_history_poly, 'b.')

    i += 1
```

```
Predictions for fold: 1
Intercept value: -0.22909053395679796
Weight values: [[ 0.27587841]
 [ 0.98600359]
 [-0.46282628]
 [ 1.20389164]
 [ 0.26371619]
 [ 0.39367081]
 [ 0.33552568]
 [-0.54997448]
 [-1.77350975]
 [ 0.99772074]
 [ 0.07817275]
 [-0.87170693]
 [-0.65326657]
 [-3.16322769]
 [-0.58800814]
 [-0.28227726]
 [ 0.21899142]
 [-0.91141269]
 [-0.5385676 ]
 [ 0.87837922]
 [-0.2824135 ]
 [-1.5935594 ]
 [-0.89857353]
 [-0.79695992]
 [ 0.25865048]
 [-0.90809488]
 [-2.15242228]
 [-3.09334718]
 [ 0.10993886]
 [-1.36200838]
 [ 1.99564857]
 [-0.39359987]
 [-1.14921232]
 [-0.55028554]
 [-0.91894303]
 [ 0.89576955]
 [-1.29744339]
 [ 0.49558487]
 [-0.52793715]
 [-1.25397122]
 [-0.68382899]
 [-0.14506495]
 [-1.3286707 ]
 [ 0.08693019]
 [ 0.47205133]
 [-1.55586634]
 [ 0.23290245]
 [ 0.30775012]
 [-2.09428005]
 [-0.95122795]
 [-2.68805415]
 [-0.38111071]
 [-2.58663409]
 [-1.64194423]
 [-2.80183177]
 [-0.20265281]
 [-0.68523008]
 [-1.55423012]
```

```
[-1.23885802]
[ 0.94060306]
[-1.20601077]
[-0.60175921]
[-3.05811761]
[-0.89572433]
[-1.00907461]
[-0.22838021]
[-1.08674554]
[-1.18057565]
[-0.42884054]
[-2.12983832]
[-1.95069479]
[-2.59110552]
[-0.57717914]
[ 0.38287894]
[-1.66687887]
[-1.21301095]
[-2.67825955]
[-0.47994589]
[-0.41586853]
[-1.67070502]
[-0.09616271]
[-1.97117876]
[-1.68728709]
[-0.9283836 ]
[-1.20986292]
[-0.56255365]
[-0.50321184]
[-1.72625625]
[-0.5252441 ]
[-1.8269357 ]
[-1.43858157]
[-0.77812023]
[-0.16206167]
[-0.69906611]
[-0.3793546 ]
[-1.48928432]
[-1.51754014]
[-1.27455859]
[-2.66805086]
[ 0.33298682]
[ 0.63024949]
[ 0.16704497]
[-0.19989626]
[-0.12241148]
[-1.37184001]
[-0.66165721]
[-0.50295377]
[-0.47959578]
[-1.70993349]
[-0.38854866]
[-1.5882683 ]
[-1.74804169]
[-1.49380637]
[-0.35087747]
[-0.48240444]
[-0.97483974]
[-1.50472234]
[-0.69243804]
```

```
[-1.37336356]
[ 1.22657343]
[-0.24632481]
[-1.4904309 ]
[ 2.53977265]
[-1.05181493]
[-0.39597868]
[-1.9467493 ]
[-1.65391209]
[-0.61957036]
[ 0.77325749]
[ 0.17767963]
[-0.25240211]
[-1.3897155 ]
[-0.1416172 ]
[-1.9389325 ]
[ 0.14346287]
[-1.49397334]
[-0.09802852]
[ 0.79297879]
[-0.43779954]
[ 0.72767461]
[ 0.87441567]
[-0.14924823]
[-0.53409951]
[ 0.83749929]
[-0.39291569]
[-0.81847461]
[ 0.42617568]
[-0.22819606]
[ 0.49347708]
[ 0.28413048]
[-2.06177398]
[-1.58054255]
[ 0.69460723]
[-0.82549383]
[-0.06179199]
[-0.96060787]
[ 1.08832178]
[-0.89372452]
[-0.74585785]
[ 0.10434753]
[ 0.36190716]
[ 0.32956222]
[ 0.16459484]
[ 0.00933581]
[-1.80262908]
[-1.77634805]
[-0.71481999]
[-0.18286769]
[-1.79827917]
[-1.53577715]
[ 0.02739514]
[-1.68134886]
[-0.2646968 ]
[-1.97431266]
[ 0.90060452]
[-1.95669037]
[-0.85565105]
[-0.54201557]
```

```
 [-0.01828822]
 [-0.77841797]
 [ 1.64607161]
 [-1.14583802]
 [ 0.05612485]
 [-1.03159373]
 [-1.48670454]
 [-0.71780676]
 [-0.40911068]
 [ 0.51360055]
 [ 0.53545435]
 [-1.85026409]
 [-0.05711182]
 [ 0.48206002]
 [-0.78849123]
 [-1.16347371]
 [-1.00702292]
 [-0.84264401]
 [ 0.05175535]
 [-1.00948132]
 [-1.65026609]
 [-1.53477937]
 [-1.62314895]
 [-0.5914304 ]
 [-0.37645346]
 [ 1.0411559 ]
 [-0.48341332]
 [-0.39290542]
 [-0.12979723]
 [-0.34247322]
 [-1.03552674]
 [-1.65372408]
 [-0.10467987]
 [-1.86193003]
 [ 0.30758193]
 [ 0.72905941]
 [ 0.92942143]
 [ 0.76329135]
 [ 1.71111755]
 [ 1.29036804]
 [-2.92136303]
 [-0.48330102]
 [-1.76625463]
 [-1.76971474]
 [ 0.07926474]
 [-0.46950118]
 [-0.33676118]
 [-1.98641792]
 [-0.11482903]
 [-0.79940642]
 [-0.31143347]
 [-3.26919762]]
Mean Square Error: 1010761.684

Predictions for fold: 2
Intercept value: 0.18238014956524806
Weight values: [[ 1.19234382e+00]
 [-1.47533620e+00]
 [ 1.05352885e+00]
 [-8.04699547e-01]
```

```
[ 6.75032653e-01]
[ 2.47960244e-01]
[ 1.27312744e+00]
[-2.02750594e+00]
[-1.02496865e+00]
[-5.01927375e-02]
[-4.58663145e-01]
[ 1.13697533e+00]
[ 3.59173182e-01]
[ 1.05462798e+00]
[ 2.72968075e-01]
[-1.48067406e+00]
[ 2.22251399e-01]
[-3.96202430e-01]
[-1.25901421e+00]
[ 1.07111397e+00]
[ 2.27494543e-01]
[ 1.82502315e-01]
[-3.35168646e-01]
[ 2.05546442e-01]
[-7.42825097e-01]
[ 6.51058060e-01]
[-2.22776535e-01]
[ 1.57300079e+00]
[-1.41975610e+00]
[ 6.81720048e-01]
[-1.60090608e+00]
[ 9.83270707e-01]
[ 9.43127098e-02]
[ 6.03598706e-01]
[-2.15347230e-01]
[ 1.22055473e+00]
[-2.59762862e-04]
[-4.85732424e-01]
[-1.13491170e+00]
[-8.13683133e-01]
[ 1.77118704e+00]
[-7.13479027e-01]
[ 6.11995415e-01]
[-5.32789328e-01]
[-4.33905552e-01]
[ 1.54295265e+00]
[-1.16172172e+00]
[-1.06327162e+00]
[-8.68016607e-02]
[ 4.35361433e-01]
[-2.63501023e-01]
[ 1.02483119e-01]
[-1.14943888e+00]
[ 1.13828830e+00]
[-7.97976153e-02]
[-9.05064593e-02]
[ 1.12259717e+00]
[ 1.62188673e+00]
[-4.11540785e-01]
[-2.07397690e-02]
[ 5.22966990e-01]
[ 4.01768430e-01]
[ 4.45971531e-01]
[-5.37758874e-01]
```

```
[-3.78374865e-01]
[ 9.74965814e-01]
[ 5.59180598e-01]
[ 1.54181895e+00]
[ 3.60243415e-01]
[ 9.37062971e-01]
[-8.36723224e-01]
[-3.53742179e-01]
[ 5.77104011e-01]
[ 5.10977707e-02]
[-1.68437854e-01]
[-9.20921683e-01]
[ 1.07166004e+00]
[-1.01420843e+00]
[ 1.90369994e+00]
[ 5.59529969e-01]
[-1.88175657e+00]
[ 1.35056410e+00]
[ 8.56251855e-01]
[ 1.13011582e-01]
[-8.60882138e-01]
[ 2.11488599e-01]
[ 6.28599446e-01]
[ 9.58264802e-01]
[-8.04539060e-01]
[-6.78370497e-01]
[-1.11096617e+00]
[ 7.88107327e-01]
[ 1.41311176e+00]
[-2.47062567e+00]
[-1.52023623e+00]
[ 7.48544346e-01]
[ 3.02875005e-01]
[-9.99122998e-01]
[-1.52377633e+00]
[ 2.25620163e-01]
[ 3.24907555e-01]
[ 1.31831327e+00]
[-1.35785882e-01]
[ 9.90776037e-02]
[ 3.01722631e-01]
[ 9.15790353e-01]
[-2.64566326e-01]
[-3.11388456e-01]
[ 2.55750777e-01]
[-3.20497541e+00]
[-6.01293752e-01]
[ 3.93943811e-02]
[ 1.86839561e-01]
[ 8.10445743e-01]
[-1.15057660e+00]
[-1.39235319e+00]
[-1.42469280e+00]
[ 1.00330347e+00]
[-7.05615561e-02]
[-5.91989258e-02]
[-9.72586712e-01]
[ 8.58881778e-01]
[ 1.26665112e-01]
[-3.20648388e-01]
```

```
[-1.24833416e+00]
[ 1.29374130e+00]
[-2.81753864e-01]
[ 2.32226102e-04]
[-1.98979393e+00]
[ 2.92067944e-01]
[-1.30158499e-01]
[ 6.61061282e-01]
[-2.24138214e-01]
[ 2.93164776e-01]
[ 1.23459531e+00]
[ 5.37387977e-01]
[ 1.60382723e+00]
[ 1.16504820e+00]
[ 4.13903984e-01]
[-4.11492935e-01]
[-1.23892554e+00]
[ 1.62113306e+00]
[ 1.70512631e+00]
[ 8.93643411e-01]
[ 1.30634086e+00]
[-5.84020549e-02]
[ 7.04298508e-01]
[ 4.80426493e-01]
[ 1.28064334e+00]
[-2.07065675e-01]
[-1.30270024e+00]
[-8.61119295e-02]
[ 1.28712430e+00]
[ 2.52708934e-01]
[-1.17864578e+00]
[ 3.48608998e-01]
[ 3.47578018e-01]
[-6.19653841e-01]
[-1.70052304e+00]
[-1.05491655e-01]
[-1.99448565e+00]
[-1.87380736e+00]
[ 6.25249441e-01]
[ 3.14689974e-01]
[ 1.10251578e+00]
[-3.50218198e-01]
[-1.47602944e+00]
[ 1.71387784e+00]
[ 2.10975862e-01]
[ 7.82484807e-01]
[ 3.40169742e-02]
[ 7.34095097e-01]
[ 7.11667082e-01]
[ 4.90421043e-01]
[-3.48980836e-01]
[-2.00516121e+00]
[ 2.65162361e-01]
[ 1.00547707e+00]
[ 1.11496071e+00]
[ 8.28954997e-01]
[ 2.85586304e-01]
[ 3.06422919e-01]
[ 1.72224611e-01]
[-3.11607637e-03]
```

```
 [-8.75925986e-01]
 [-9.95740616e-02]
 [-1.20633250e+00]
 [ 2.29702206e-01]
 [ 7.40439791e-01]
 [-6.30564340e-01]
 [ 7.89057613e-01]
 [ 8.92992210e-02]
 [ 7.39101242e-01]
 [ 5.65062734e-02]
 [ 9.18353977e-01]
 [-5.95858028e-01]
 [-4.27932128e-01]
 [ 2.59843805e-01]
 [ 6.94885122e-01]
 [-1.19353916e-01]
 [-1.94117223e-01]
 [-2.41188565e+00]
 [ 1.20601260e+00]
 [-6.86277517e-01]
 [ 1.27434193e+00]
 [ 1.19898139e+00]
 [ 3.78049133e-01]
 [ 2.14946667e+00]
 [-2.43063667e+00]
 [ 5.53255894e-01]
 [-1.34313845e+00]
 [-5.05798033e-01]
 [-4.31363952e-01]
 [ 9.54002718e-02]
 [-4.24435760e-01]
 [-1.07651485e+00]
 [ 9.21405636e-01]
 [ 5.89041639e-01]
 [-1.39871699e+00]
 [ 9.99843815e-01]
 [-3.12360648e-01]
 [-2.29580130e-01]
 [-3.51144025e-01]
 [-5.42000996e-01]
 [ 6.64116770e-03]
 [-5.12014140e-01]
 [ 4.54489196e-01]
 [ 1.93030009e+00]
 [ 7.86232888e-01]
 [-1.22914859e+00]]
Mean Square Error: 2447.489

Predictions for fold: 3
Intercept value: -0.9774674484880297
Weight values: [[ 1.15002375e+00]
 [-1.43245045e+00]
 [-4.99076574e-01]
 [-1.34505125e-01]
 [-2.37496720e+00]
 [-2.13309400e+00]
 [ 6.70576492e-01]
 [ 4.43179228e-02]
 [-5.56236912e-01]
 [-1.57898315e-01]
```

```
[ 1.39596269e-01]
[ 3.19752005e-01]
[ 1.22533930e+00]
[-9.85937414e-01]
[-1.66709565e+00]
[-3.27851010e-01]
[ 3.56343797e-01]
[ 1.05788478e-01]
[-2.26513356e-03]
[-5.14850769e-01]
[-1.14072103e+00]
[ 5.89376611e-03]
[-1.22435844e-01]
[ 6.35429873e-01]
[-1.11920234e-01]
[ 6.52889126e-01]
[ 1.67634137e-01]
[-4.37147748e-01]
[-8.59592616e-01]
[-1.19758805e+00]
[-1.27197207e+00]
[-2.31675243e-01]
[-2.89991441e-01]
[-1.56846790e+00]
[ 1.64525208e+00]
[-1.08934289e+00]
[ 8.41893961e-03]
[ 3.97550960e-01]
[ 9.80044593e-02]
[ 2.28248741e-01]
[ 2.17932929e-01]
[-2.35641786e+00]
[ 4.67944931e-01]
[-1.49902723e+00]
[ 1.02098701e+00]
[ 6.74876321e-01]
[ 1.12647028e+00]
[-1.35051210e+00]
[ 1.17190046e+00]
[ 7.78059986e-01]
[ 5.01015565e-01]
[ 7.90979885e-01]
[ 1.21509059e+00]
[ 5.48269379e-01]
[ 1.35805169e+00]
[ 1.64023197e-01]
[ 2.08849495e+00]
[ 4.37574882e-01]
[-1.44471729e+00]
[-3.72778969e-02]
[ 6.82174991e-01]
[-2.05142094e-01]
[ 8.26121404e-01]
[ 4.68518948e-01]
[ 8.58302862e-01]
[-7.02341360e-03]
[-1.08212938e+00]
[-1.25275093e+00]
[-4.88370527e-01]
[-9.15343544e-03]
```

```
[ 4.18948981e-01]
[-8.74031173e-01]
[ 5.16911263e-01]
[ 8.17561728e-01]
[-1.26182537e+00]
[ 1.09211271e+00]
[ 1.18969623e+00]
[-5.57882178e-01]
[-2.04495762e-01]
[-5.46716616e-01]
[-1.33641173e+00]
[-7.34418697e-01]
[ 3.89073531e-01]
[ 1.82477623e+00]
[ 2.29565192e-01]
[-8.43811575e-02]
[-1.41322813e-01]
[-1.44143727e+00]
[ 2.20635418e+00]
[-6.35939615e-01]
[ 9.05559375e-01]
[-6.73041435e-01]
[-2.85229641e-01]
[-9.99506103e-01]
[ 1.13752554e+00]
[ 6.50545470e-01]
[ 4.40653923e-01]
[ 8.29860499e-01]
[ 3.92972936e-01]
[ 1.10542271e+00]
[ 1.31821636e-02]
[-8.50682958e-01]
[-1.70049978e+00]
[-1.86796810e+00]
[-6.78717175e-02]
[ 3.86729224e-01]
[-1.13106033e+00]
[-5.80228382e-02]
[ 7.70791530e-01]
[-1.22574750e+00]
[ 4.76631321e-01]
[-3.14015450e-01]
[ 1.57485831e-01]
[ 5.28124480e-01]
[-7.84426770e-01]
[ 6.61494555e-01]
[-1.13743149e+00]
[-2.01675610e+00]
[ 2.43308335e-02]
[ 3.19392523e-01]
[-5.84244082e-01]
[-2.27193078e-01]
[-1.02204256e+00]
[ 1.57732077e+00]
[-4.50786128e-01]
[ 1.70263818e-01]
[ 2.56008146e-02]
[-3.05670107e-01]
[-1.21001062e+00]
[ 3.13895033e-01]
```

```
[-1.48184727e+00]
[-2.37673632e+00]
[-2.31871300e-01]
[ 1.12710260e+00]
[ 7.67196269e-01]
[-5.83596640e-01]
[ 3.43008333e-01]
[ 2.07072179e+00]
[-1.21201836e+00]
[ 2.79504338e-01]
[ 2.16575424e-01]
[ 6.76673579e-01]
[-1.66040392e+00]
[ 4.18128547e-01]
[-6.17920689e-01]
[ 2.39458470e+00]
[-1.76589536e+00]
[ 1.30358272e+00]
[ 2.63401999e+00]
[ 3.40059763e-01]
[-5.09110720e-01]
[ 7.99147819e-02]
[-5.73250510e-01]
[-9.07747473e-01]
[ 7.17737099e-01]
[-3.60191910e-01]
[ 2.32790654e-01]
[-1.15448929e-01]
[ 1.00172224e+00]
[ 8.89983135e-01]
[-3.73932206e-01]
[-1.38570376e+00]
[-1.56105200e+00]
[-6.75927169e-01]
[ 5.12687893e-01]
[-1.15112700e-02]
[ 8.13506952e-01]
[-6.23950451e-01]
[-5.58539336e-01]
[ 1.86849058e+00]
[-6.47051182e-01]
[-1.64898415e+00]
[-4.61965941e-01]
[ 2.38767295e-01]
[ 9.65146269e-02]
[-1.34901525e+00]
[-3.15278291e-01]
[-2.10921313e-01]
[ 1.33501665e+00]
[-3.11046291e-01]
[ 1.14595787e-01]
[-4.52508976e-01]
[ 6.09582145e-01]
[ 1.10840794e+00]
[-1.39740636e+00]
[ 8.93510202e-01]
[-5.61438864e-01]
[ 1.40169272e+00]
[-8.96607707e-03]
[-1.12174681e+00]
```

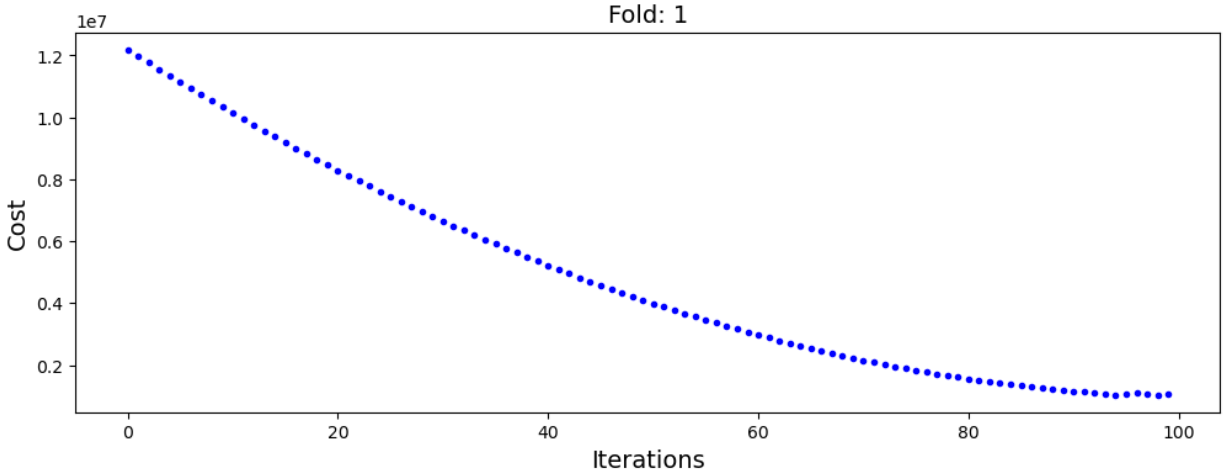
```
 [ 1.83121112e+00]
 [-1.28572641e+00]
 [-1.75100441e+00]
 [ 1.21159191e+00]
 [ 2.48493268e-01]
 [-1.43727958e+00]
 [ 4.58890729e-01]
 [ 2.76227142e-01]
 [-2.20912873e+00]
 [-2.09934289e-01]
 [-8.74239067e-01]
 [ 9.47998333e-01]
 [-1.83497790e+00]
 [-1.57822988e+00]
 [-5.64293117e-01]
 [-1.21848604e-01]
 [ 2.11636187e-03]
 [-2.42068920e-01]
 [ 1.37207090e+00]
 [-4.87630605e-02]
 [-1.45294164e+00]
 [ 9.33396722e-02]
 [ 3.92076676e-01]
 [-3.77745958e-01]
 [ 1.19857101e+00]
 [ 3.56512194e-01]
 [ 6.00475046e-01]
 [ 3.73682638e-01]
 [-1.40563342e+00]
 [ 1.34867733e+00]
 [-1.08136205e+00]
 [-5.40472052e-01]
 [ 1.23158284e+00]
 [ 7.06273346e-01]
 [-5.63454098e-01]
 [-2.17693481e-01]
 [ 3.48710411e-01]
 [ 7.67329890e-01]
 [-9.94460488e-01]
 [ 8.32163763e-01]]
Mean Square Error: 4129.166

Predictions for fold: 4
Intercept value: 1.4396125581477883
Weight values: [[ 3.39532868e-01]
 [ 5.36789889e-01]
 [ 1.61133323e+00]
 [ 3.12853170e-01]
 [-7.53458760e-01]
 [ 7.40041574e-01]
 [ 1.46846340e+00]
 [-1.22325859e+00]
 [ 5.10823598e-01]
 [ 1.43662985e+00]
 [ 4.45304931e-01]
 [-1.36906682e+00]
 [ 1.02153485e-01]
 [-3.04119449e-01]
 [-1.10101247e+00]
 [-1.53517123e-01]
```

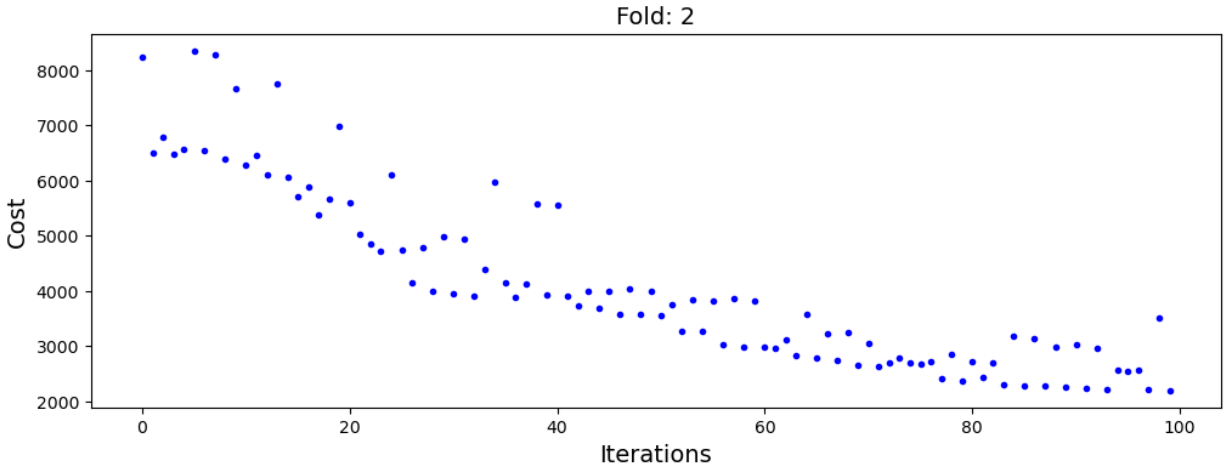
```
[-1.42500186e-01]
[-1.35693540e+00]
[-6.22158325e-01]
[-3.10623766e-01]
[ 1.16959999e+00]
[ 4.50824654e-01]
[-2.16979903e-03]
[-1.11995250e+00]
[ 1.43695774e+00]
[ 1.87211173e+00]
[ 9.70673531e-01]
[-3.65226019e-01]
[ 8.32433515e-01]
[ 1.00736669e+00]
[-3.96573751e-02]
[ 4.57582543e-01]
[ 1.01500099e+00]
[-1.21597741e-02]
[ 1.83936463e+00]
[ 2.84700474e-01]
[ 4.89021665e-01]
[ 4.05405863e-01]
[-1.38061589e-01]
[ 1.20778117e+00]
[ 7.27403015e-01]
[ 6.59901938e-01]
[-1.18641444e+00]
[ 1.43342145e-02]
[-1.26532033e-01]
[-1.38659968e-01]
[-4.67489866e-01]
[-5.18740795e-01]
[-4.92782227e-02]
[ 1.25709243e+00]
[ 1.08437292e+00]
[-1.14790089e+00]
[ 6.29002404e-01]
[-4.11843295e-01]
[-2.75610676e-01]
[ 1.73837526e+00]
[ 3.40284229e-02]
[-1.40246802e+00]
[ 9.99344908e-01]
[-2.35038741e-01]
[-7.70841743e-01]
[ 1.51101874e+00]
[ 1.58717154e+00]
[-5.26670405e-01]
[ 8.93529326e-01]
[ 4.03866355e-01]
[-1.72688945e+00]
[-1.79403502e-01]
[-5.16206404e-01]
[-1.21685789e+00]
[ 4.63946802e-01]
[ 6.02275404e-01]
[ 9.93532981e-01]
[-4.20779335e-01]
[ 7.17311872e-01]
[ 3.56642134e-01]
```

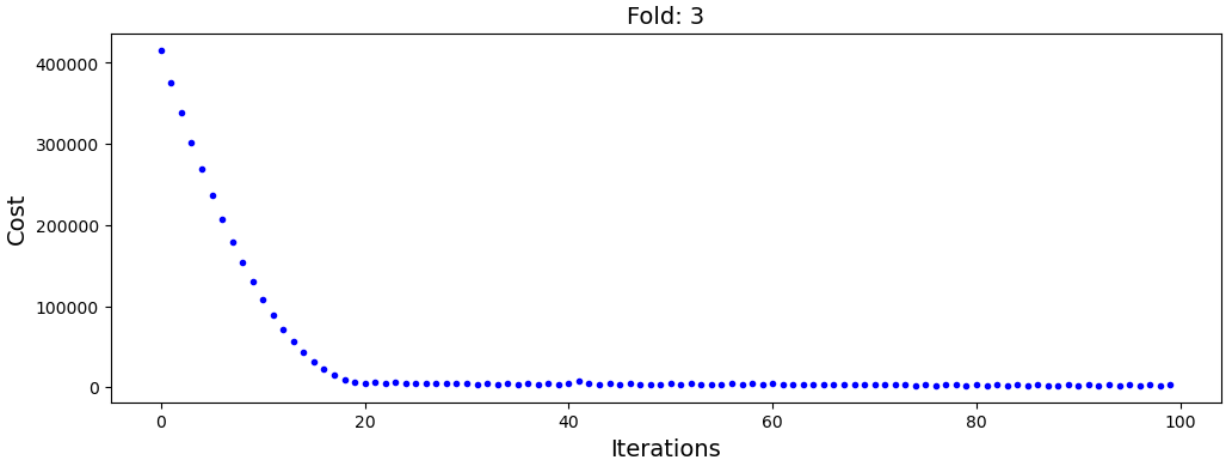
```
[ 1.05864877e+00]
[-7.90722339e-02]
[-1.86816577e+00]
[-1.17401543e-01]
[ 2.30233485e-01]
[-1.19171762e+00]
[ 1.97535492e+00]
[ 9.23881153e-01]
[ 1.67810513e-01]
[ 4.43838834e-01]
[-3.74475780e-01]
[ 2.44347969e+00]
[-7.26469865e-01]
[-8.07172802e-01]
[ 6.32305324e-01]
[ 1.19911335e+00]
[-6.26633135e-01]
[-4.24239435e-01]
[-3.28897755e-01]
[-2.01399772e+00]
[-1.78813887e+00]
[-2.16845379e+00]
[ 2.40374909e+00]
[-7.35771943e-01]
[ 1.61846019e+00]
[-1.29906303e+00]
[-2.72380258e+00]
[-1.26600478e+00]
[ 1.15199506e+00]
[-1.04207821e+00]
[ 5.68221845e-01]
[ 1.09488504e+00]
[-6.19008454e-01]
[ 1.65036194e-01]
[ 6.12009899e-01]
[-2.91176924e-01]
[ 1.72813525e+00]
[ 1.25426991e+00]
[ 4.87147027e-01]
[ 2.37774618e+00]
[ 4.88715482e-01]
[ 1.50052784e-01]
[ 1.54114999e+00]
[ 1.18150863e+00]
[-1.42408125e-01]
[ 1.42677601e+00]
[ 2.10641626e+00]
[-7.77946255e-01]
[ 8.09820239e-01]
[ 2.33756523e-01]
[-1.88372272e-01]
[ 5.08166926e-01]
[-2.39107136e+00]
[ 5.36862859e-01]
[ 1.62024124e+00]
[-1.11361496e-01]
[ 7.71872626e-01]
[-2.10857745e-02]
[-8.48208775e-01]
[ 4.09008029e-01]
```

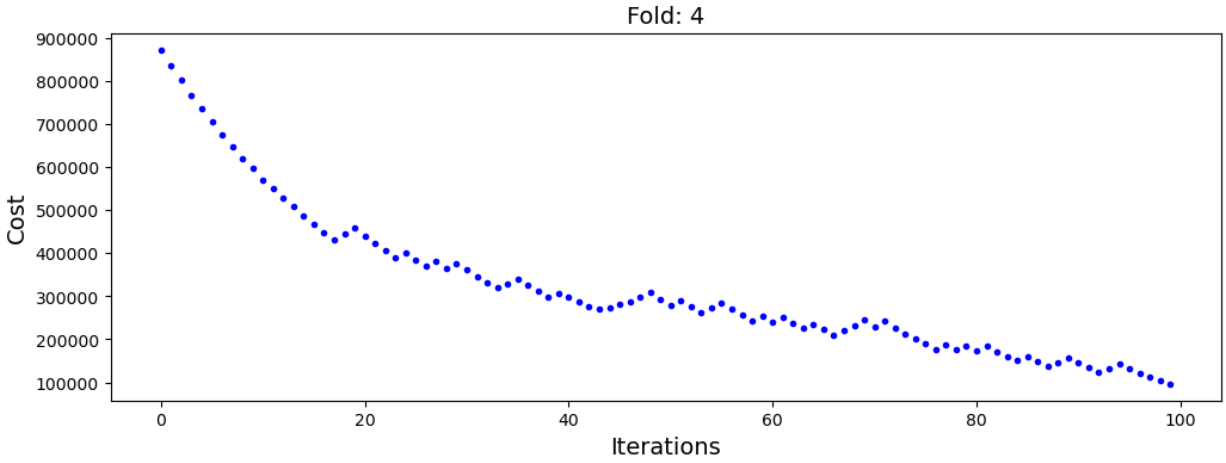
```
[ 1.43208376e-01]
[ 7.12890058e-02]
[-1.17342665e+00]
[-6.29951310e-01]
[-1.25506742e+00]
[ 6.00088750e-01]
[ 1.51498920e+00]
[-2.35901659e+00]
[ 9.68998549e-01]
[-6.82502353e-01]
[-6.33436896e-01]
[-4.04979202e-02]
[ 1.77237723e+00]
[-1.14330359e+00]
[-1.02998868e+00]
[-3.71404805e-01]
[-1.13690747e+00]
[-1.21189726e+00]
[ 2.17241919e-01]
[ 2.95803464e-01]
[-1.17024108e+00]
[-8.40277544e-02]
[-3.92824665e-02]
[ 7.56138742e-01]
[-1.90233262e+00]
[-5.22613414e-01]
[-9.52252955e-01]
[ 5.93597788e-01]
[ 3.60701258e-01]
[-1.96003516e-01]
[ 2.61442476e-01]
[-6.64357062e-01]
[ 9.36649767e-01]
[-1.69694064e-01]
[-6.41820948e-01]
[ 6.37740531e-01]
[-1.82882138e+00]
[ 1.55732551e+00]
[-7.06518922e-01]
[ 8.87020271e-01]
[-1.94487372e-02]
[-6.44770302e-01]
[ 2.22911212e+00]
[ 8.49654178e-01]
[-5.03759533e-01]
[-6.55704257e-01]
[-3.61166222e+00]
[ 2.00176770e+00]
[ 2.28189987e-03]
[-2.57626296e-01]
[-1.15072958e-01]
[ 2.95877707e-01]
[ 4.65558652e-01]
[-8.77610042e-01]
[ 4.12025390e-01]
[-1.29474631e+00]
[ 1.18713934e+00]
[ 2.07010351e+00]
[-9.05327732e-01]
[-5.80470294e-01]
```

```
 [ 2.62574336e+00]
 [-1.48116386e+00]
 [-4.30039333e-01]
 [-8.39282550e-01]
 [-5.18095798e-01]
 [ 4.71338057e-01]
 [-8.50543534e-01]
 [ 1.10187700e+00]
 [-1.65328091e+00]
 [ 2.74719390e-01]
 [-2.79482235e-01]
 [-7.72889697e-03]
 [ 8.07406802e-01]
 [-1.34304737e+00]
 [-7.45436358e-01]
 [-1.61713495e+00]
 [ 6.20066571e-01]
 [ 8.60536325e-01]
 [-5.00918464e-01]
 [-1.14886961e+00]
 [ 9.52744484e-01]
 [-3.11155345e-01]
 [ 2.68994323e-01]
 [-4.71904344e-01]
 [ 1.45079791e+00]
 [ 4.58020395e-01]
 [-1.59292101e+00]
 [ 6.71738022e-02]
 [-6.80904489e-01]
 [-9.91677037e-01]
 [-4.27001088e-01]
 [ 1.41089021e-03]
 [-7.19155252e-01]
 [ 6.54378121e-01]]
Mean Square Error: 62672.773
```









Trial one: Batch size - 42, learning rate - 0.01

In [79]:
```
i = 1
gradient_clip_threshold = 1.0  # Adjust this threshold as needed

for train_index, test_index in kf.split(X_train_poly):
    X_train_fold_poly, X_test_fold_poly = X_train_poly[train_index], X_train_po
    y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]

    # Add bias column to fold data
    X_train_fold_poly = np.c_[np.ones((len(X_train_fold_poly), 1)), X_train_fol
    X_test_fold_poly = np.c_[np.ones((len(X_test_fold_poly), 1)), X_test_fold_p
```

```
    lr = 0.01
    n_iter = 100
    batch_size = 42
    theta_poly = np.random.randn(231, 1)  # Initialize coefficients (matching t

    # Train the model using minibatch gradient descent with gradient clipping
    theta_poly, cost_history_poly = minibatch_gradient_descent(X_train_fold_pol

    print('Predictions for fold:', i)
    print('Intercept value:', theta_poly[0, 0], '\nWeight values:', theta_poly[

    # Use the validation data and mean square error to evaluate performance
    N = len(y_test_fold)
    y_hat_poly = np.dot(X_test_fold_poly, theta_poly)
    mse_val_poly = (1 / N) * (np.sum(np.square(y_hat_poly - y_test_fold)))
    print('Mean Square Error:', round(mse_val_poly, 3), '\n')

    fig, ax = plt.subplots(figsize=(12, 4))
    plt.title('Fold: ' + str(i))
    ax.set_ylabel('Cost')
    ax.set_xlabel('Iterations')
    _ = ax.plot(range(n_iter), cost_history_poly, 'b.')

    i += 1
```

```
Predictions for fold: 1
Intercept value: 1.7059262520917533
Weight values: [[-5.95047416e-01]
 [ 3.90183665e-02]
 [ 9.26841627e-01]
 [ 1.04923305e+00]
 [-7.79937976e-02]
 [ 1.17511143e+00]
 [-7.33148426e-01]
 [ 7.90049951e-01]
 [-1.40290113e+00]
 [ 1.82287789e+00]
 [ 2.47220411e-01]
 [-1.46698272e-01]
 [-2.49780280e-01]
 [-4.45108707e-01]
 [ 4.39733394e-01]
 [ 4.06816245e-01]
 [ 1.48472136e+00]
 [ 1.29958363e+00]
 [-2.32731924e+00]
 [-9.86614610e-02]
 [ 7.42966245e-01]
 [ 2.21208676e+00]
 [-2.73794995e-01]
 [-1.89800978e-01]
 [-8.56594648e-01]
 [ 2.72640503e-01]
 [-4.72318084e-01]
 [-4.64178122e-02]
 [ 3.33171013e-01]
 [ 4.37946492e-01]
 [-1.04848513e+00]
 [ 1.95262965e+00]
 [ 2.38020549e-01]
 [ 1.71191159e+00]
 [ 2.08184467e-01]
 [ 2.11364769e+00]
 [-6.65954411e-01]
 [-1.45833471e+00]
 [ 8.54817453e-01]
 [ 2.32296715e-01]
 [ 7.64022389e-01]
 [ 3.97088352e-01]
 [ 6.12962952e-01]
 [ 4.75295235e-01]
 [ 5.25927954e-01]
 [-1.94213528e-02]
 [-7.17340940e-01]
 [ 2.26320785e+00]
 [ 1.34761334e+00]
 [-5.54099690e-01]
 [-2.53415702e-01]
 [ 2.33355165e+00]
 [ 5.79299195e-01]
 [ 1.40962283e+00]
 [ 2.39229629e-01]
 [ 6.54724957e-01]
 [-1.25636061e+00]
 [ 9.00327879e-01]
```

```
[ 1.82162068e+00]
[ 1.37707813e-01]
[-1.49940488e+00]
[ 8.15271414e-01]
[-1.24343494e+00]
[-4.18506910e-01]
[-3.83432714e-01]
[-8.26603924e-01]
[ 1.07230472e+00]
[ 5.14350068e-01]
[ 1.22428254e+00]
[-7.34437826e-01]
[ 1.11330761e+00]
[ 1.02106844e+00]
[-8.89222677e-01]
[-1.88537307e-01]
[ 1.70747824e+00]
[ 7.89523469e-01]
[ 1.04423667e+00]
[-1.53906005e+00]
[-7.32960987e-01]
[ 6.18573038e-01]
[ 1.01074103e+00]
[ 1.13034178e+00]
[ 1.08380578e+00]
[ 1.39493993e+00]
[ 1.03703025e+00]
[-1.03845341e+00]
[ 1.20111061e+00]
[ 3.70886345e-01]
[ 3.11646855e+00]
[ 8.48926880e-02]
[ 4.74772309e-01]
[-2.01508045e+00]
[ 1.90405345e+00]
[-2.96448692e-01]
[-7.86211519e-01]
[-7.93977394e-01]
[-9.56114502e-01]
[-4.07769886e-01]
[ 2.08455761e+00]
[-8.21822729e-01]
[ 1.13903840e+00]
[ 2.68602772e-01]
[-1.26285746e+00]
[ 4.48178739e-01]
[ 5.33908794e-01]
[-2.87970816e-01]
[ 1.81697022e-02]
[-2.42729580e+00]
[-6.94169258e-01]
[ 1.56003392e-02]
[ 1.62819733e+00]
[-9.14276904e-01]
[ 5.11069210e-01]
[ 8.51866490e-01]
[-1.79546590e-01]
[ 3.55716465e-01]
[-2.87679890e-02]
[ 1.11786513e+00]
```

```
[ 1.87632713e+00]
[ 2.09237681e-01]
[ 6.18400521e-01]
[ 2.24923884e+00]
[-1.96890466e-02]
[-2.71586284e-02]
[ 1.11048879e+00]
[ 2.62719141e-01]
[ 7.91504843e-02]
[-4.56679143e-01]
[ 5.03003843e-01]
[-5.60553548e-01]
[-8.69703233e-02]
[ 1.44847882e+00]
[ 4.40176880e-01]
[ 2.30309607e-01]
[ 2.84719744e+00]
[ 3.25846654e-01]
[ 1.70687976e-02]
[ 2.48726679e+00]
[ 6.85613991e-01]
[ 1.67916827e+00]
[-9.30245150e-01]
[-6.21047236e-02]
[-7.48343651e-01]
[-8.34700802e-01]
[-1.97496244e-01]
[ 2.34019423e+00]
[ 5.01773035e-01]
[ 3.40077428e-01]
[ 1.50687216e+00]
[-1.08352575e+00]
[ 6.43383517e-01]
[-8.18112348e-01]
[ 9.20225564e-01]
[-1.50935351e+00]
[-1.27829450e+00]
[-1.41946870e-01]
[-9.50488220e-01]
[-1.41867620e-02]
[ 1.63648680e+00]
[-1.37335162e+00]
[ 9.18192440e-01]
[-2.94394136e-01]
[ 5.89648584e-01]
[-1.40279254e+00]
[ 1.03491061e+00]
[-1.27831348e+00]
[-2.14982762e+00]
[-1.35813832e+00]
[ 9.41405434e-01]
[-3.34773306e-01]
[ 4.04951493e-01]
[ 2.76853305e+00]
[-1.15915894e+00]
[-2.91578220e-02]
[ 1.21790102e+00]
[ 1.52240406e-02]
[-3.96497213e-01]
[-1.34168610e+00]
```

```
 [-5.74431373e-01]
 [-9.98210543e-01]
 [ 2.54037916e-01]
 [-3.65382051e-01]
 [-1.23711291e+00]
 [ 8.00982041e-01]
 [-7.13990103e-02]
 [-5.70916537e-02]
 [ 7.44144592e-01]
 [ 1.09789624e+00]
 [-7.30855498e-01]
 [-2.19186080e-01]
 [ 1.66338150e-01]
 [-7.07501046e-01]
 [ 9.20189525e-01]
 [ 1.27287363e+00]
 [ 1.60235028e+00]
 [-7.83375046e-01]
 [ 1.41018036e+00]
 [-1.62063604e-01]
 [ 2.15714894e-03]
 [ 2.95830134e-02]
 [-1.64284061e+00]
 [ 2.35224181e-01]
 [ 1.51755302e+00]
 [ 6.65908194e-01]
 [ 1.70086020e+00]
 [ 1.81863001e+00]
 [ 1.24981129e+00]
 [-2.24192580e+00]
 [-8.49787190e-01]
 [ 6.40051621e-01]
 [ 6.68517041e-01]
 [ 1.09713482e+00]
 [-7.68180641e-02]
 [-5.97686926e-01]
 [ 1.67555148e+00]
 [ 1.04973323e+00]
 [ 3.59344198e-01]
 [-4.68893451e-01]
 [ 8.10240138e-01]
 [-6.04932498e-01]
 [-4.55705923e-01]
 [-3.20591457e+00]
 [-1.61550780e-01]
 [ 5.76550885e-01]
 [-2.03631409e-01]
 [-2.71533813e-01]
 [ 9.72995576e-01]
 [ 1.57267228e+00]
 [-8.20065635e-01]
 [ 2.48578553e-01]]
Mean Square Error: 31507.4

Predictions for fold: 2
Intercept value: -0.5264191140385722
Weight values: [[-8.95879795e-01]
 [ 1.16013791e+00]
 [-1.15051517e-01]
 [-1.51600654e+00]
```

```
[-2.13814215e+00]
[-2.26195223e+00]
[-6.30255989e-02]
[ 5.19452215e-01]
[ 7.95389887e-01]
[ 5.41585586e-01]
[ 8.01713481e-01]
[ 2.72497230e-01]
[ 4.21297794e-01]
[-2.41129645e-01]
[ 9.06332959e-01]
[-7.04751458e-01]
[ 1.68202093e+00]
[ 2.32701970e+00]
[-1.43076143e+00]
[-6.20284613e-01]
[-5.40211635e-01]
[ 9.75303310e-03]
[ 5.84962811e-01]
[-1.44395768e+00]
[ 1.24684505e+00]
[ 8.45546545e-01]
[ 8.26045038e-01]
[-4.50026817e-01]
[ 2.71754507e-01]
[-6.87648920e-01]
[-6.92074394e-02]
[-1.35235301e+00]
[-2.42232277e-01]
[-1.13821010e+00]
[ 4.47561524e-01]
[-1.31778147e+00]
[ 6.15973451e-01]
[ 8.93833295e-01]
[-7.66559376e-02]
[ 1.12746048e-01]
[-6.56188986e-01]
[ 3.67195000e-01]
[-2.17836929e+00]
[ 1.35109000e+00]
[ 1.93135130e+00]
[ 3.32581282e-01]
[-7.45445365e-01]
[ 3.88085718e-01]
[ 6.93425978e-01]
[ 1.42776773e+00]
[ 2.94964217e-02]
[ 7.08601040e-01]
[ 2.59853104e+00]
[-1.08908583e-01]
[ 1.08947761e+00]
[ 5.48939141e-01]
[ 1.04556788e+00]
[ 1.55708570e+00]
[-4.48256213e-01]
[-2.00898306e-01]
[ 7.95475346e-01]
[-6.40041301e-01]
[ 7.43052810e-01]
[ 5.84112855e-01]
```

```
[-3.66697804e-01]
[-9.53754429e-01]
[-4.61685139e-01]
[-1.41339164e-01]
[ 1.78968946e-01]
[ 1.06093154e-01]
[ 4.88177488e-01]
[ 5.26066801e-01]
[ 2.12228101e-01]
[ 7.35144896e-01]
[-1.05705915e+00]
[ 1.40986168e+00]
[-5.46134206e-01]
[-4.45688508e-01]
[ 6.09712202e-01]
[ 1.21419203e+00]
[-3.06301926e-01]
[ 4.75616298e-01]
[ 1.51459267e+00]
[ 7.10054380e-01]
[-1.27336855e-01]
[ 4.11213176e-01]
[ 2.76530865e-01]
[-3.43298761e-01]
[-1.13348814e+00]
[ 7.24147658e-01]
[ 6.71520308e-02]
[-1.48667890e-01]
[ 1.27787297e+00]
[ 6.57708586e-01]
[ 4.14688296e-01]
[ 1.16954862e+00]
[-5.92852879e-01]
[ 1.42542660e+00]
[-2.56412127e-02]
[-6.83041051e-02]
[-7.42374946e-01]
[-1.22206027e+00]
[ 9.46484780e-01]
[ 1.06117850e+00]
[ 9.34444006e-01]
[ 6.33667886e-01]
[ 6.53242411e-01]
[-1.06793729e+00]
[-1.20003661e+00]
[ 1.15156694e+00]
[ 7.06472037e-01]
[-6.33274639e-01]
[-2.17089866e-01]
[ 1.44368520e+00]
[ 7.14405422e-01]
[ 7.45393088e-01]
[-1.12454827e+00]
[-1.67425545e-01]
[ 1.11370698e-01]
[-7.07646353e-01]
[ 9.02920097e-01]
[-6.53746932e-01]
[ 8.98445909e-01]
[ 1.18922275e-01]
```

```
[ 9.19201349e-01]
[ 2.97168896e+00]
[ 1.00809448e+00]
[ 3.93422211e-01]
[-8.51614872e-01]
[ 1.12128254e+00]
[-1.06089691e+00]
[-3.39399597e-02]
[ 1.00486277e+00]
[-5.64926877e-01]
[ 6.01594147e-01]
[ 1.27147061e+00]
[-2.42914947e-01]
[-1.63515999e-01]
[ 1.71103448e+00]
[ 1.55284494e+00]
[ 1.41614525e+00]
[ 5.65328089e-01]
[ 1.17582202e+00]
[ 4.02080697e-01]
[ 3.21495740e-01]
[ 9.18364250e-01]
[-3.00095734e-03]
[ 2.62369082e-01]
[ 7.20932606e-01]
[-1.87432346e-01]
[-7.57296883e-01]
[-5.16280087e-01]
[-1.50156503e+00]
[-1.55488947e+00]
[ 7.86555460e-01]
[ 4.14782821e-02]
[ 3.00192570e+00]
[-3.59898525e-01]
[-2.89976960e-01]
[ 2.73027265e-01]
[ 2.79594935e-01]
[-1.09841663e+00]
[-9.62826046e-01]
[-8.17083598e-01]
[-4.85429685e-01]
[ 6.93849524e-01]
[-3.80535274e-01]
[-1.90408202e+00]
[-1.52166046e-02]
[ 1.22500459e+00]
[-1.34885596e+00]
[ 2.05234748e+00]
[-4.62029291e-02]
[ 7.88861074e-01]
[ 2.05420113e+00]
[ 7.22663151e-01]
[-1.43762412e+00]
[ 6.09554209e-01]
[ 3.07289663e-03]
[ 1.77442892e+00]
[-5.84463753e-01]
[-9.48769733e-01]
[ 4.29679649e-01]
[ 1.26505925e+00]
```

```
 [-2.17511466e-01]
 [-1.86952352e-01]
 [ 2.23868382e+00]
 [-9.96728814e-01]
 [-9.03057489e-01]
 [-1.66350951e+00]
 [ 1.27480162e+00]
 [ 8.36550423e-01]
 [-1.14752148e+00]
 [ 3.17588529e-02]
 [ 5.16412548e-01]
 [ 1.15864009e+00]
 [ 1.12954168e+00]
 [-1.08823003e+00]
 [ 5.04670917e-01]
 [ 2.73515494e+00]
 [ 2.33677868e-01]
 [-3.18271488e-01]
 [-6.53038643e-02]
 [-6.57937989e-01]
 [ 2.17763114e-01]
 [ 1.20164086e+00]
 [ 1.10852124e+00]
 [-1.28386487e+00]
 [ 2.93697187e-01]
 [ 1.37043358e+00]
 [ 1.26676993e+00]
 [ 1.06211444e+00]
 [ 9.39601658e-01]
 [-5.11654189e-01]
 [ 8.69193584e-01]
 [ 1.77003574e-01]
 [-1.78759683e+00]
 [ 1.25274468e+00]
 [-1.39156254e+00]
 [-1.38006756e+00]
 [ 3.04340094e-01]
 [-6.94269896e-01]
 [-9.19746079e-01]
 [ 2.61948298e+00]
 [ 9.67203283e-01]
 [ 2.63734349e-01]
 [-1.76669508e+00]
 [ 9.70364317e-02]
 [ 1.03988462e+00]
 [ 5.30385418e-01]]
Mean Square Error: 50713.088

Predictions for fold: 3
Intercept value: 0.4186874048523137
Weight values: [[ 0.18089966]
 [-0.40877028]
 [ 2.09549397]
 [ 2.17137835]
 [ 0.0863522 ]
 [ 0.58966792]
 [ 1.01514536]
 [ 0.90104866]
 [ 1.03912417]
 [ 1.35513069]
```

```
[ 0.42665327]
[ 1.56905873]
[-1.72513417]
[ 0.23178317]
[-0.14745027]
[ 0.08065046]
[-0.46233623]
[-1.03495398]
[ 0.83762673]
[ 0.58584943]
[-0.72342199]
[ 0.19881885]
[-0.46810494]
[ 0.27478162]
[-2.26928375]
[ 0.07263756]
[-0.02411735]
[ 1.5050147 ]
[-0.28725994]
[ 0.2942774 ]
[-0.45789014]
[-1.39941203]
[-0.44067084]
[ 0.13668803]
[-0.43384257]
[-0.04269355]
[ 0.68241202]
[ 1.94414461]
[-0.36898125]
[ 1.18556331]
[ 1.03992104]
[ 0.45317371]
[-0.02911262]
[-0.24337472]
[-1.32587183]
[ 0.72666908]
[-1.54829331]
[-0.26194147]
[ 0.68488549]
[ 1.34372873]
[ 0.12790157]
[-1.26842198]
[ 0.6664901 ]
[ 0.16083345]
[ 0.14517728]
[-0.22572137]
[ 1.19129378]
[-0.29050574]
[ 0.6341014 ]
[ 0.21015788]
[-0.55766368]
[-0.157272  ]
[-1.64041279]
[ 0.05359903]
[ 0.03801006]
[-0.64486387]
[ 0.37875686]
[-0.76585778]
[-0.57491841]
[ 0.2844833 ]
```

```
[-0.72637938]
[ 1.1522688 ]
[ 1.26912362]
[ 0.37782385]
[ 0.83158729]
[-0.8445873 ]
[-0.36876045]
[-0.53424228]
[ 1.74116049]
[ 1.31543844]
[ 1.0282623 ]
[ 1.30236144]
[-1.25095727]
[-0.34551487]
[ 0.06245304]
[ 0.73008645]
[ 0.05581511]
[ 2.4619028 ]
[-0.68047905]
[-0.66700898]
[ 0.66639076]
[ 0.296726  ]
[ 0.36475788]
[ 1.08043543]
[ 0.75314553]
[ 1.76315864]
[ 0.53075595]
[-0.24297707]
[ 1.00131452]
[ 0.70277343]
[ 0.45154415]
[ 1.48349388]
[ 0.2459966 ]
[-0.01412951]
[-1.43145688]
[ 0.0599727 ]
[-0.39510147]
[-0.40553536]
[-0.12428777]
[-0.03846668]
[ 1.76821586]
[ 0.68010232]
[ 2.54205574]
[ 1.87648807]
[ 0.14975907]
[-0.25941849]
[ 2.83149397]
[-2.42488308]
[ 3.38775169]
[-0.39650182]
[-0.94947274]
[ 0.38079729]
[-0.11498207]
[-0.310289  ]
[ 0.81338581]
[ 0.58388048]
[ 0.41374757]
[-1.82810864]
[ 0.49862519]
[-0.46630677]
```

```
[ 1.25444528]
[-1.55664881]
[-0.93959486]
[-0.91901898]
[-0.14800158]
[ 1.77543913]
[ 0.7973625 ]
[-0.10409445]
[ 0.26592401]
[-1.05623399]
[ 0.45083548]
[ 0.37290145]
[ 0.84295827]
[ 0.93238789]
[-0.04779224]
[ 1.32878832]
[-2.74655584]
[-0.67875689]
[ 0.28244209]
[-0.89577227]
[ 0.99328611]
[-0.80154937]
[ 0.60726491]
[-0.5335792 ]
[-0.96063763]
[ 0.08343656]
[-0.31145844]
[ 0.25423559]
[ 0.54790889]
[-0.2448477 ]
[ 0.53618011]
[ 0.85597302]
[ 0.17867652]
[ 0.71021092]
[ 0.48815675]
[ 1.97896089]
[ 0.8331921 ]
[-0.2647333 ]
[-0.90266758]
[-1.99258697]
[-0.25438813]
[ 0.98061343]
[ 0.2423945 ]
[-0.29565015]
[ 1.04636574]
[-2.05028683]
[ 0.32464419]
[-0.47363995]
[ 0.18409684]
[-0.15616695]
[ 1.16643983]
[ 0.95593736]
[ 0.8921112 ]
[ 0.64648993]
[ 0.1496233 ]
[ 0.92828247]
[ 1.10481112]
[-0.06833207]
[-0.33692576]
[ 1.36350394]
```

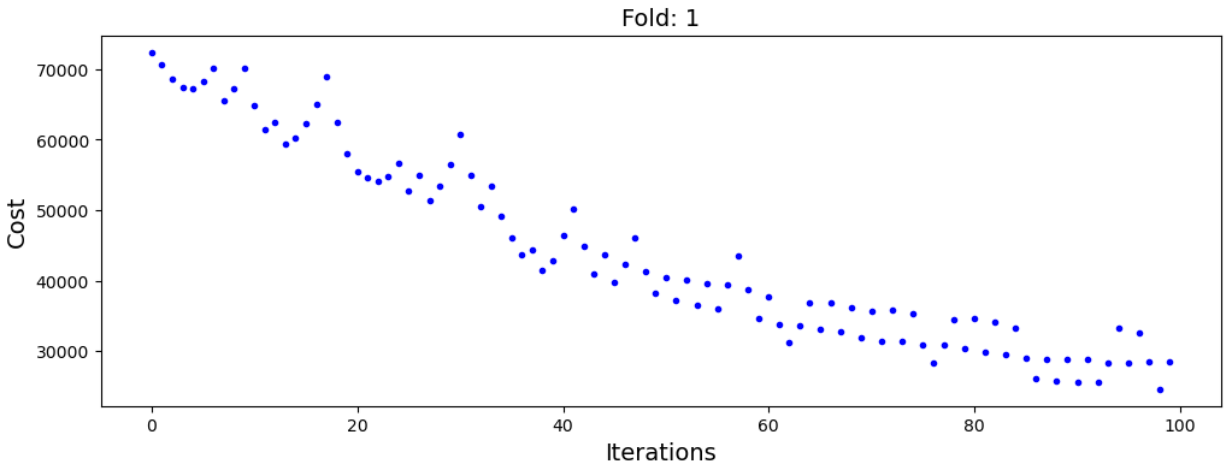
```
 [ 1.83055918]
 [-0.5775227 ]
 [ 0.7644562 ]
 [ 0.1657654 ]
 [ 0.49349573]
 [-1.97750032]
 [-0.10677487]
 [-1.50301872]
 [ 0.16876404]
 [-1.3768519 ]
 [-2.15338637]
 [-0.44055286]
 [-0.07193727]
 [ 1.20292287]
 [ 0.08758386]
 [ 1.01530332]
 [-0.82299045]
 [ 0.19153776]
 [-1.62501608]
 [-0.6675933 ]
 [-0.89795189]
 [ 0.9272575 ]
 [ 2.48001984]
 [ 0.25359843]
 [ 1.11956404]
 [ 1.73113652]
 [ 1.13474712]
 [ 0.89914937]
 [-0.32337219]
 [ 1.1995996 ]
 [ 0.57218809]
 [ 0.08218283]
 [ 0.35307484]
 [-0.68818642]
 [ 0.07949848]
 [-1.21309143]
 [-1.58378835]
 [ 0.35766969]
 [ 1.25791782]
 [-1.15148648]]
Mean Square Error: 62117.438

Predictions for fold: 4
Intercept value: 0.4004429067951868
Weight values: [[-0.61915978]
 [-0.71020215]
 [ 0.46325539]
 [ 1.24674394]
 [-0.82638091]
 [-0.12572263]
 [ 0.62350681]
 [ 1.37014871]
 [ 0.43260361]
 [ 0.03431532]
 [ 0.45400619]
 [ 0.09263289]
 [ 0.79562472]
 [-0.01074257]
 [ 2.33332746]
 [ 0.29322376]
```

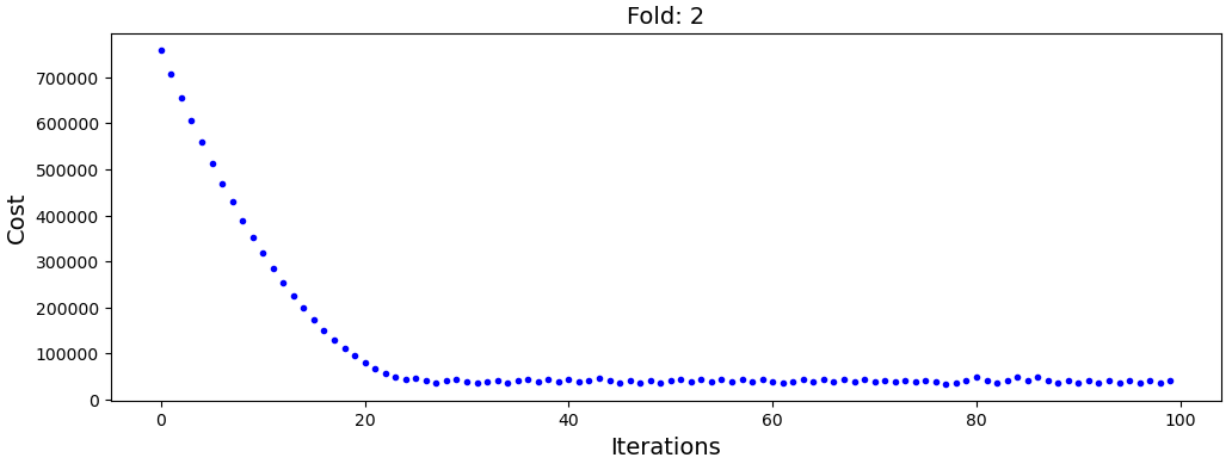
```
[ 0.63282701]
[ 1.26550367]
[ 3.39145029]
[ 2.10347005]
[ 0.93078848]
[-0.62927961]
[ 0.09852907]
[ 1.14572784]
[ 0.53773449]
[ 0.15895257]
[-1.0260235 ]
[-1.39263837]
[ 1.76996275]
[ 0.06886932]
[ 1.02040099]
[ 1.13306155]
[ 1.91221435]
[-0.54458641]
[ 0.13836044]
[ 0.34042863]
[-0.47938531]
[ 0.5766038 ]
[-0.02329846]
[-0.67109887]
[ 0.03851803]
[ 0.36973084]
[ 0.10696811]
[ 0.33054578]
[ 1.32573934]
[ 0.47864464]
[ 2.12178721]
[ 0.53149169]
[ 0.15063281]
[ 2.04588703]
[ 0.68906161]
[ 0.0219669 ]
[-0.71201301]
[-0.31065481]
[ 0.57099266]
[ 0.53515287]
[-0.06948886]
[ 0.98997144]
[ 0.73821347]
[-0.30952945]
[ 1.04438127]
[-0.64966736]
[ 1.04299741]
[ 1.78445334]
[ 0.88044633]
[ 0.41808828]
[ 1.76036513]
[ 1.6747808 ]
[-0.08698516]
[ 0.18486416]
[ 0.18975273]
[ 1.94464332]
[ 2.03269669]
[ 1.65846966]
[-0.13345466]
[-0.67621681]
```

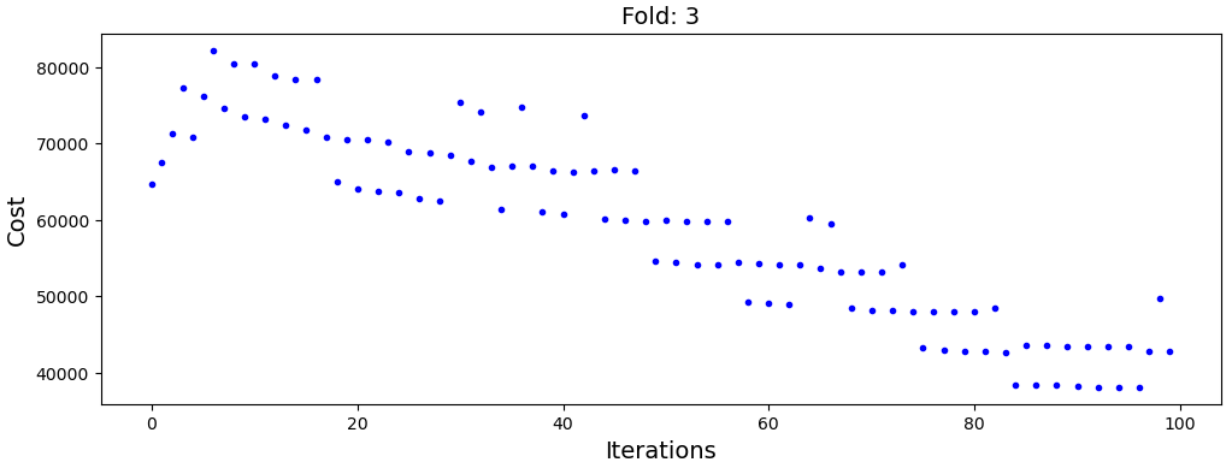
```
[ 0.53179123]
[ 2.37617575]
[-1.19349305]
[ 1.33662391]
[ 0.89275431]
[ 0.00766809]
[-0.21413571]
[-0.26361437]
[ 1.64452158]
[-0.08530748]
[ 0.67756201]
[ 1.74153441]
[-0.89437202]
[ 0.38552338]
[ 0.33831051]
[ 0.00593049]
[ 0.81991046]
[-0.48319649]
[ 0.22920909]
[ 0.95339115]
[ 0.93906636]
[ 1.01495427]
[ 2.38668462]
[-0.01566612]
[ 1.85633951]
[-0.84123237]
[ 0.35473195]
[ 0.8978981 ]
[ 1.30569405]
[ 0.65478321]
[ 1.13666527]
[ 0.63488208]
[ 0.7770562 ]
[ 1.32822261]
[-0.5777389 ]
[ 0.15397003]
[ 0.76615972]
[-1.29059261]
[ 1.18253832]
[ 0.70963377]
[ 0.41959552]
[ 1.62270456]
[ 1.23122227]
[-0.53642643]
[ 0.73466507]
[ 0.24389817]
[ 0.28206006]
[ 1.47020075]
[ 0.95765188]
[ 0.99331207]
[ 0.29982951]
[-1.78328786]
[-0.14166694]
[ 0.34767226]
[ 2.19122209]
[-0.47486993]
[-0.46535981]
[-0.42776197]
[ 1.88014205]
[ 1.84619495]
```

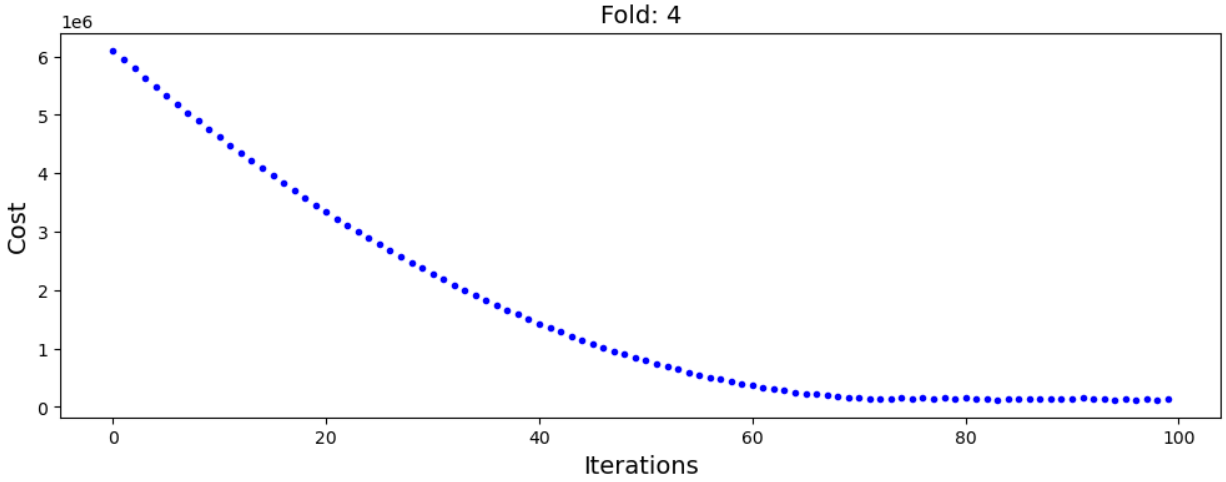
```
[-0.66376623]
[ 1.00226894]
[-1.52318328]
[-0.69854071]
[ 0.04830707]
[-1.29950249]
[-3.12628141]
[ 0.2953297 ]
[ 2.5999189 ]
[ 0.80678151]
[ 1.53963188]
[ 0.40235188]
[-1.22778995]
[-0.04252908]
[ 0.16413598]
[ 0.2463848 ]
[ 0.77695823]
[-0.2538653 ]
[ 2.58418497]
[ 1.40507506]
[ 1.22504934]
[ 0.99445668]
[ 0.54767549]
[ 1.14696563]
[-0.96817168]
[-0.27591664]
[-0.21475409]
[-0.25556808]
[ 0.44056477]
[ 0.47413314]
[ 2.9620863 ]
[ 0.9920247 ]
[-1.16602492]
[ 0.46334552]
[ 1.76685138]
[-0.3359008 ]
[-1.89880927]
[-0.12037061]
[-1.0851294 ]
[ 0.91930108]
[ 1.75027578]
[ 0.90108036]
[-0.01831293]
[-0.71852747]
[ 1.08692736]
[-0.46372349]
[-0.1307956 ]
[ 0.94394128]
[ 0.76626357]
[ 0.03747044]
[-1.29355604]
[ 1.09847771]
[-0.80537861]
[-2.09201301]
[ 0.49965412]
[ 0.63825152]
[-0.88764243]
[ 1.90832701]
[ 1.90190375]
[ 0.41019313]
```

```
 [-0.97728614]
 [ 0.75337037]
 [ 0.4200678 ]
 [ 0.47101657]
 [ 1.70781744]
 [ 0.514059  ]
 [ 0.27510055]
 [ 0.52378621]
 [ 0.00697781]
 [ 0.5280009 ]
 [ 0.45734857]
 [ 1.02713621]
 [ 0.78666837]
 [ 1.32420877]
 [-0.20748563]
 [ 3.07424286]
 [-1.89771481]
 [ 2.09301454]
 [-1.12854509]
 [-0.08386357]
 [-0.85871143]
 [ 0.7001315 ]
 [-1.53914194]
 [ 0.53441243]
 [-0.25882313]
 [-0.01221407]
 [ 1.8804599 ]
 [ 0.94777003]
 [-0.21673782]
 [ 1.75063586]
 [ 0.44025798]
 [ 1.85341852]
 [ 1.3060048 ]
 [ 1.09293777]]
Mean Square Error: 70944.725
```









Trial one: Batch size - 68, learning rate - 0.01

In [80]:
```
i = 1
gradient_clip_threshold = 1.0  # Adjust this threshold as needed

for train_index, test_index in kf.split(X_train_poly):
    X_train_fold_poly, X_test_fold_poly = X_train_poly[train_index], X_train_po
    y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]

    # Add bias column to fold data
    X_train_fold_poly = np.c_[np.ones((len(X_train_fold_poly), 1)), X_train_fol
    X_test_fold_poly = np.c_[np.ones((len(X_test_fold_poly), 1)), X_test_fold_p
```

```
    lr = 0.01
    n_iter = 100
    batch_size = 68
    theta_poly = np.random.randn(231, 1)  # Initialize coefficients (matching t

    # Train the model using minibatch gradient descent with gradient clipping
    theta_poly, cost_history_poly = minibatch_gradient_descent(X_train_fold_pol

    print('Predictions for fold:', i)
    print('Intercept value:', theta_poly[0, 0], '\nWeight values:', theta_poly[

    # Use the validation data and mean square error to evaluate performance
    N = len(y_test_fold)
    y_hat_poly = np.dot(X_test_fold_poly, theta_poly)
    mse_val_poly = (1 / N) * (np.sum(np.square(y_hat_poly - y_test_fold)))
    print('Mean Square Error:', round(mse_val_poly, 3), '\n')

    fig, ax = plt.subplots(figsize=(12, 4))
    plt.title('Fold: ' + str(i))
    ax.set_ylabel('Cost')
    ax.set_xlabel('Iterations')
    _ = ax.plot(range(n_iter), cost_history_poly, 'b.')

    i += 1
```

```
Predictions for fold: 1
Intercept value: 0.39611187695705263
Weight values: [[-0.95838268]
 [ 0.9313907 ]
 [ 0.34713063]
 [ 1.29041725]
 [ 0.70152562]
 [ 0.67793376]
 [ 0.11695157]
 [ 0.25227536]
 [ 1.08905871]
 [ 1.11038872]
 [ 0.47227226]
 [ 0.82908777]
 [-0.98207275]
 [-0.77754674]
 [ 1.36420818]
 [-0.15033595]
 [ 0.48861073]
 [ 0.67030883]
 [ 0.74451223]
 [ 1.68179952]
 [ 1.42032524]
 [-0.42775485]
 [-1.04928996]
 [ 1.35329975]
 [-0.36996784]
 [ 2.0326408 ]
 [ 1.38293523]
 [ 0.69617427]
 [ 0.507377  ]
 [ 0.56220369]
 [ 0.06571664]
 [ 1.36528749]
 [-0.59022152]
 [ 1.4778068 ]
 [-0.85862408]
 [ 1.51165122]
 [ 0.75989187]
 [-2.55554739]
 [-0.00841138]
 [ 0.8329222 ]
 [ 2.00988959]
 [-0.36037338]
 [-0.57328933]
 [-0.5511655 ]
 [ 0.32022375]
 [ 1.47345431]
 [ 0.21666764]
 [ 0.95389186]
 [-0.85669793]
 [-0.10412629]
 [ 1.8462778 ]
 [-0.24661133]
 [ 0.52200382]
 [-1.29311795]
 [ 0.6276826 ]
 [ 0.43277822]
 [ 0.86461535]
 [ 0.8711814 ]
```

```
[-0.73424615]
[-0.24734663]
[ 0.49369518]
[ 0.74724896]
[ 0.28504404]
[ 1.01941329]
[ 2.04303485]
[ 1.01965092]
[ 2.252606  ]
[ 2.32958895]
[ 0.46538065]
[ 0.49284085]
[ 2.77569295]
[ 0.92982237]
[-0.25071867]
[ 0.09487579]
[ 1.39187134]
[-1.29046976]
[ 1.66660385]
[-0.17932045]
[ 0.11010069]
[ 0.38237204]
[-0.86170372]
[-0.3396362 ]
[ 1.29124142]
[ 0.45516807]
[ 2.73270148]
[ 0.87945616]
[ 0.54342863]
[ 1.3046877 ]
[ 1.04613241]
[ 1.72597913]
[-1.54664075]
[ 0.79380664]
[-0.01782659]
[ 0.30502682]
[ 1.73377993]
[-0.62310471]
[ 1.72335056]
[ 1.04429933]
[ 0.15135023]
[ 0.33899874]
[ 0.59769026]
[ 2.28227512]
[ 1.70066261]
[-0.19769171]
[ 1.10351658]
[-0.35236915]
[ 0.68121621]
[ 0.17083655]
[ 0.83538145]
[ 0.8013149 ]
[ 1.17178527]
[ 0.97556021]
[ 0.89647054]
[ 0.82031807]
[ 0.90642842]
[ 0.97825855]
[ 1.92947756]
[ 1.90183771]
```

```
[ 1.14259094]
[ 0.3013301 ]
[ 0.5102349 ]
[ 2.21076011]
[-0.6220833 ]
[-1.16244491]
[ 0.00634546]
[ 0.98489502]
[-0.91125578]
[ 1.84845517]
[-0.71629001]
[ 1.6901264 ]
[ 0.13739205]
[ 0.06437222]
[ 0.76596361]
[ 1.78800844]
[ 0.89400367]
[ 1.56961057]
[ 0.34826254]
[-0.34789988]
[ 0.09650157]
[-0.39373515]
[-0.89900691]
[ 2.06907111]
[-1.55496529]
[-2.17218802]
[ 0.82704949]
[ 1.90086826]
[ 0.76010219]
[ 0.47537163]
[-0.16980174]
[-2.01760725]
[ 0.47152744]
[ 0.06004842]
[-0.05519064]
[ 1.49724245]
[ 0.10087755]
[ 1.06105088]
[-0.18402655]
[ 0.87137108]
[-1.10009998]
[ 0.59955193]
[ 0.4794906 ]
[-0.48097258]
[ 2.02974171]
[-1.94212434]
[-0.61145253]
[ 0.87254713]
[ 1.49957399]
[ 1.43615615]
[-0.17646777]
[ 0.40774643]
[ 0.33443481]
[ 1.21575674]
[ 0.34061647]
[-0.42043045]
[-0.96998048]
[ 0.00944584]
[ 1.05077745]
[-0.44609419]
```

```
 [ 1.15065828]
 [ 0.07484167]
 [ 0.58138942]
 [ 1.13805772]
 [ 0.25012332]
 [ 0.08901476]
 [ 0.35410456]
 [ 1.82063197]
 [-0.2346914 ]
 [ 0.73109343]
 [ 0.36447313]
 [-0.30520569]
 [ 1.35661825]
 [ 0.91360968]
 [ 1.30505758]
 [ 1.10022578]
 [ 0.12707402]
 [-0.65578856]
 [ 0.68375648]
 [-0.69736003]
 [ 0.82817277]
 [-1.25930773]
 [ 0.66798144]
 [-0.98220224]
 [-0.60133157]
 [ 1.55766279]
 [-1.232789  ]
 [ 0.08202366]
 [-0.68835664]
 [-0.59412365]
 [ 0.98961913]
 [ 0.86682079]
 [ 1.70982018]
 [ 1.32313822]
 [ 0.98419175]
 [ 0.35697342]
 [ 1.74748877]
 [ 0.49549199]
 [ 0.17173793]
 [ 1.6216335 ]
 [-0.53482704]
 [-0.89007202]
 [-0.61022132]
 [ 0.85649002]
 [ 0.16816044]
 [-0.47546002]
 [-0.39753269]
 [ 1.6637206 ]
 [ 0.03239022]
 [ 0.52730709]
 [ 0.23164922]
 [ 2.17437247]]
Mean Square Error: 140208.974

Predictions for fold: 2
Intercept value: 1.478864813807668
Weight values: [[-9.99601333e-01]
 [-1.49045279e+00]
 [-3.20746129e-02]
 [-9.84843054e-03]
```

```
[-9.00240728e-01]
[-9.09645777e-01]
[ 2.89521951e-01]
[-3.89200572e-02]
[-6.00609931e-02]
[ 2.99853112e-01]
[-3.00477202e+00]
[-9.14521846e-01]
[ 1.40281382e+00]
[-9.54821406e-01]
[-5.62203329e-01]
[ 8.87562060e-01]
[-7.82777461e-01]
[-7.21530124e-02]
[ 3.97449894e-01]
[-6.25654591e-02]
[-6.19876486e-01]
[ 8.40701003e-01]
[ 4.78676299e-01]
[-1.54491145e+00]
[-5.31606101e-02]
[-6.77111833e-01]
[-1.62048643e+00]
[-1.33886059e+00]
[-9.93372455e-01]
[-6.88216817e-01]
[-2.33027674e+00]
[-8.37121734e-01]
[-2.78813219e+00]
[-9.54210424e-01]
[-1.22573608e+00]
[-2.76520460e-01]
[-3.07526993e+00]
[ 1.53931969e+00]
[-7.38264384e-01]
[-5.69927044e-01]
[-2.45219862e+00]
[-8.58362803e-01]
[-2.34855063e-02]
[ 1.66840896e+00]
[-5.10975526e-01]
[ 2.09177109e-01]
[ 1.48081234e+00]
[ 5.78677979e-01]
[ 7.27521153e-01]
[-5.71071605e-01]
[ 2.67051922e-01]
[-2.37567907e+00]
[ 9.06083366e-02]
[-1.16918228e+00]
[-5.07841752e-01]
[-2.90942246e-01]
[-2.39962658e+00]
[-1.26142880e+00]
[ 3.71716980e-02]
[ 3.50381914e-01]
[-3.50540183e-01]
[-8.37340366e-01]
[-1.08917755e+00]
[ 5.75790636e-01]
```

```
[-4.75606497e-02]
[-8.54680656e-01]
[-6.19636712e-01]
[-1.23772528e+00]
[ 2.03566695e-01]
[-8.67573940e-02]
[ 5.78247183e-01]
[-9.51972701e-01]
[-1.21556401e-01]
[-1.64858408e+00]
[-2.98782965e-01]
[-1.64219012e+00]
[-9.17444647e-01]
[ 1.95548272e-02]
[-1.40357486e-01]
[-8.04177203e-01]
[-2.23161544e+00]
[ 3.48330224e-01]
[ 1.25834838e-01]
[-2.42769680e+00]
[-8.17671604e-01]
[ 4.09368937e-01]
[-6.11227120e-01]
[-1.99951116e+00]
[-1.77436003e+00]
[-6.23139715e-01]
[ 9.09782524e-01]
[ 7.48554919e-01]
[ 4.33634132e-01]
[-1.48935518e+00]
[ 1.22823806e+00]
[-1.17791196e-01]
[ 1.52216575e+00]
[-1.96374976e+00]
[-1.47824860e+00]
[-6.93733316e-01]
[ 5.08693202e-01]
[ 2.94063084e-01]
[-2.14748711e+00]
[ 4.94649668e-01]
[ 9.52102965e-01]
[-1.05378919e+00]
[ 6.89354525e-01]
[-3.96086219e-01]
[-1.08147377e-01]
[-9.84129684e-01]
[ 4.04118317e-01]
[-7.56813126e-01]
[-5.61499712e-01]
[ 7.83772521e-01]
[-1.08006841e+00]
[ 3.56587057e-02]
[-6.14451034e-02]
[ 1.74409826e+00]
[-7.92260960e-01]
[-1.08662243e+00]
[ 3.40529676e-01]
[-1.04612384e+00]
[ 5.49996127e-03]
[-9.13166346e-02]
```

```
[-5.88786769e-01]
[-2.44805183e+00]
[-1.13363566e+00]
[ 8.39574489e-02]
[-5.61614093e-01]
[-1.23702993e+00]
[-9.03270375e-01]
[ 2.05189898e-01]
[-1.12548782e+00]
[-2.40210155e-01]
[-6.11921011e-01]
[-1.47129243e+00]
[-1.43068298e+00]
[ 7.55635685e-01]
[-9.48848539e-01]
[ 6.45922904e-01]
[-4.89297526e-01]
[-8.39456386e-01]
[ 4.31807035e-01]
[-6.92805198e-01]
[-4.68914419e-01]
[ 6.07513401e-01]
[ 1.93690855e-01]
[-5.48133999e-01]
[-6.39059452e-01]
[ 8.52389619e-01]
[ 3.37095843e-02]
[-1.62769389e+00]
[ 2.88672373e-03]
[-6.48513657e-02]
[ 2.25447328e+00]
[ 7.56286612e-02]
[ 1.71157167e-01]
[-3.63547459e-01]
[ 2.96091450e+00]
[ 1.21984956e+00]
[-1.09698649e+00]
[-5.89237848e-02]
[ 8.09806774e-02]
[ 2.17992404e+00]
[-1.21072613e+00]
[-1.61898753e-01]
[ 3.45032191e-01]
[ 9.60998627e-01]
[ 6.07302164e-01]
[ 7.39059580e-02]
[ 1.89987257e-01]
[-1.86602783e+00]
[ 3.27296637e-01]
[-1.74490154e+00]
[ 4.44341733e-01]
[-8.93705490e-01]
[-6.84986108e-01]
[-3.47808915e-01]
[-7.16935395e-01]
[ 1.09766326e+00]
[-9.42919737e-01]
[-3.45074443e+00]
[ 2.03757613e-01]
[ 1.11901911e+00]
```

```
 [-1.27282682e+00]
 [-1.00870528e+00]
 [ 1.15200117e+00]
 [-1.10510468e+00]
 [-4.98584973e-01]
 [ 1.90655124e+00]
 [-1.71565279e+00]
 [ 5.72953305e-01]
 [-1.55387680e-02]
 [ 1.69590881e+00]
 [ 4.07196823e-01]
 [-8.89748846e-01]
 [-5.54483608e-01]
 [ 7.29388745e-01]
 [-9.86432776e-01]
 [ 1.66131832e-01]
 [-2.24702807e-01]
 [ 7.74524449e-02]
 [-7.54590507e-01]
 [-1.28701940e+00]
 [-1.25912376e+00]
 [ 2.79025412e-01]
 [ 6.51020218e-01]
 [-1.19911325e-01]
 [-3.91720227e-01]
 [ 9.47985284e-01]
 [ 1.21275499e+00]
 [-5.16653705e-01]
 [ 2.95653973e-01]
 [-6.95353894e-01]
 [ 1.37890597e-01]
 [-1.40701804e+00]
 [ 1.46453426e+00]
 [-1.12094179e+00]
 [-4.63254191e-01]
 [-4.75011330e-01]
 [-1.84123670e+00]
 [ 1.29027200e+00]
 [-1.24432953e+00]
 [-3.69606415e-01]
 [ 6.99302549e-01]
 [-7.06444922e-01]
 [ 1.26631213e+00]
 [ 1.50276209e-01]
 [ 4.49805260e-01]
 [-9.34380693e-01]]
Mean Square Error: 183300.032

Predictions for fold: 3
Intercept value: 1.7918634892751069
Weight values: [[ 1.40073304]
 [-1.71346565]
 [-0.37627611]
 [-0.01276363]
 [-0.63390914]
 [-0.94097391]
 [ 0.69929048]
 [-0.58678154]
 [-1.72029443]
 [ 0.10816967]
```

```
[-0.25670454]
[-1.13744789]
[ 0.42957837]
[-0.7131022 ]
[ 0.34895887]
[-2.93001782]
[ 0.25816994]
[-0.62928086]
[-0.94248545]
[-1.70596322]
[-0.08434851]
[-0.46175278]
[ 0.38250283]
[-1.92245323]
[ 0.01125205]
[-1.8072341 ]
[-0.72867928]
[-1.45259851]
[-0.97927444]
[-0.7652196 ]
[-2.01152594]
[-1.13544213]
[ 1.76745938]
[ 0.69162515]
[-1.55696359]
[ 0.19651013]
[ 0.16341774]
[ 0.66838158]
[-0.4452143 ]
[ 0.56319585]
[-0.34103295]
[ 1.40778832]
[ 0.58624043]
[ 0.6726417 ]
[-0.06420562]
[-0.68699733]
[-0.95408937]
[ 0.49306625]
[-1.48628552]
[-1.01922006]
[ 1.00883646]
[ 0.9089523 ]
[-0.67728116]
[-0.17409423]
[-0.06136805]
[-0.08344664]
[-1.48401467]
[-1.10712209]
[-0.05502707]
[ 0.12158418]
[ 0.48013598]
[-0.95329611]
[-0.56358206]
[ 0.18850906]
[ 1.33361969]
[-1.61793653]
[ 1.2614347 ]
[-0.47407977]
[-0.92360887]
[ 0.37551115]
```

```
[-0.50181028]
[ 0.76467074]
[ 0.00697954]
[-1.20871235]
[-0.93191658]
[-0.66463593]
[ 0.36467422]
[-0.0849595 ]
[ 3.15779642]
[-1.54404141]
[-0.5871636 ]
[ 0.78732393]
[-0.45533119]
[-0.0156129 ]
[-0.10655116]
[ 0.41366995]
[ 0.79722284]
[-0.04282266]
[-0.66108342]
[-0.70190173]
[-0.56977505]
[-0.57438603]
[-0.83476724]
[-0.64290057]
[-0.06076477]
[-0.31485317]
[-0.27448484]
[ 0.20201563]
[ 0.57837768]
[-0.54694615]
[-0.82622932]
[ 0.82146153]
[ 1.24569704]
[ 0.11431463]
[ 0.91765386]
[-0.42880021]
[-0.43425032]
[ 0.54830044]
[ 1.47872584]
[ 0.18064546]
[-2.99199023]
[-0.70795415]
[-0.78357649]
[ 0.00686854]
[-0.73607513]
[-0.85101278]
[ 1.68958434]
[ 0.10560292]
[ 0.47246701]
[-0.42032814]
[ 0.1928946 ]
[ 0.16479023]
[ 1.53828515]
[-0.82321491]
[-0.01869519]
[ 0.50451274]
[-0.04148124]
[ 0.22837728]
[-0.69380133]
[-0.6757828 ]
```

```
[-0.01688169]
[ 0.71540076]
[-1.66855016]
[-0.49978945]
[-0.0316297 ]
[-0.66728069]
[ 0.24225274]
[ 0.05470294]
[-2.72749079]
[-0.17630461]
[ 0.75300748]
[-0.0103641 ]
[ 0.49753544]
[ 0.35853006]
[-0.18923423]
[-0.19865265]
[-1.3744126 ]
[-0.44254217]
[-1.29178063]
[-1.15394651]
[ 1.59086029]
[-2.04681448]
[-0.37976355]
[ 0.61411041]
[ 0.53254694]
[-0.01065545]
[-0.25601471]
[-2.6344308 ]
[ 0.15928605]
[ 0.57075682]
[-0.02810353]
[-1.06945289]
[ 0.62181725]
[-1.0681825 ]
[-2.82599017]
[ 0.41668639]
[ 0.49745475]
[ 0.07384027]
[ 0.42432972]
[-0.87214349]
[-0.2373232 ]
[-1.43986769]
[-1.28043441]
[-0.37093315]
[-0.12642824]
[-0.95306271]
[ 0.45907643]
[-0.32094226]
[-1.80405505]
[ 0.26758562]
[-1.86538868]
[ 0.32457666]
[-2.28272625]
[-0.64044751]
[-0.46015358]
[ 0.60962608]
[-1.46638265]
[ 0.71341958]
[ 1.50484024]
[-0.58124833]
```

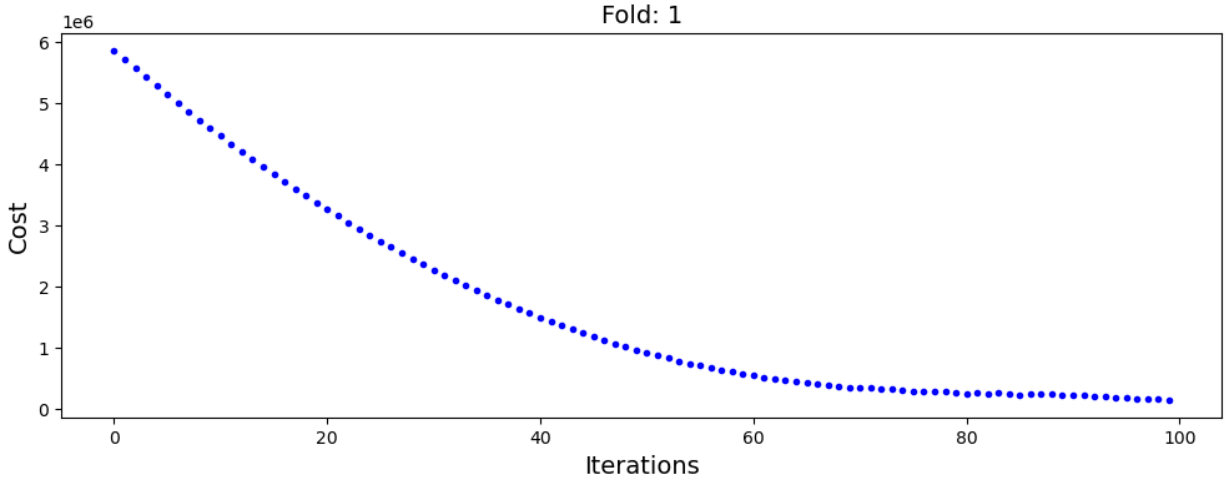
```
 [ 0.47959225]
 [ 1.43002714]
 [ 1.08083697]
 [ 1.04703631]
 [-0.17604484]
 [ 0.03359269]
 [-0.81430008]
 [-1.63126446]
 [ 1.49608641]
 [-0.28448812]
 [ 0.91254893]
 [-0.87105306]
 [-0.44700487]
 [ 0.4088752 ]
 [ 0.93646982]
 [ 1.03645729]
 [ 1.32449053]
 [-0.96801312]
 [-0.59392183]
 [ 0.69316163]
 [-0.97031487]
 [-0.78017283]
 [-0.35515691]
 [ 0.66742487]
 [-0.41211165]
 [ 0.20605519]
 [-0.09559741]
 [ 1.89629116]
 [-0.56312264]
 [ 0.05078357]
 [-1.59737538]
 [ 0.67581793]
 [ 1.90287723]
 [-1.45561148]
 [-0.72722761]
 [ 0.15296957]
 [ 0.32653484]
 [-1.36319493]
 [-0.65534119]
 [-1.20513976]]
Mean Square Error: 38336.489

Predictions for fold: 4
Intercept value: 0.6153748631096175
Weight values: [[ 7.56471200e-01]
 [ 3.44463021e-01]
 [ 1.64823448e-01]
 [ 9.56235594e-01]
 [-7.06538355e-01]
 [ 8.30835538e-01]
 [-1.25348121e+00]
 [ 2.17469337e-01]
 [-5.83425357e-01]
 [-7.45296886e-01]
 [ 8.69387999e-01]
 [ 1.12737639e+00]
 [-2.31006634e+00]
 [ 3.11897043e-01]
 [-2.51131340e-01]
 [-2.19269900e-01]
```

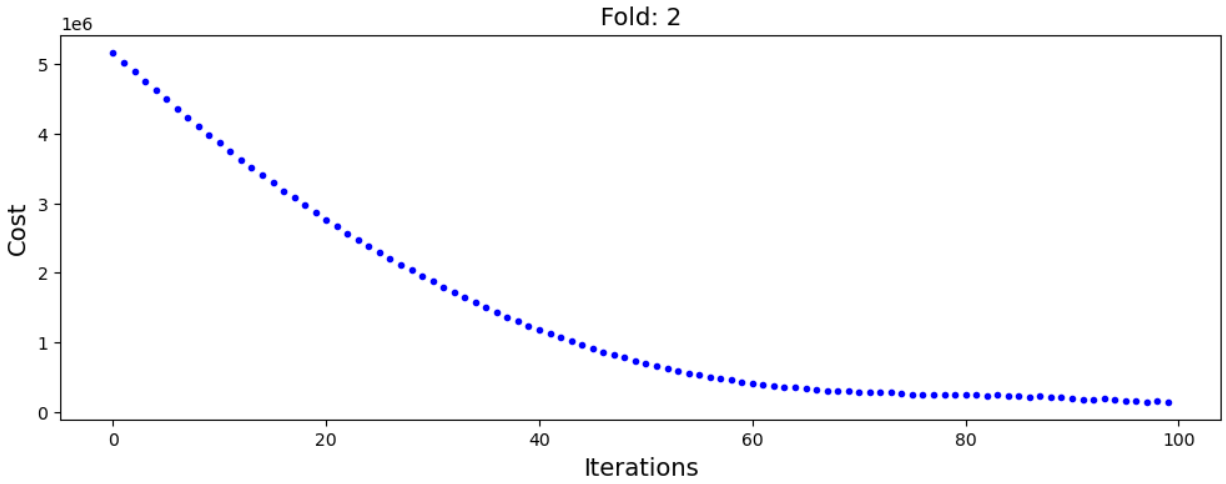
```
[-4.93399611e-01]
[-1.59331604e+00]
[-2.09377194e-01]
[-5.73348600e-01]
[ 1.28836604e+00]
[-4.34035296e-01]
[-8.66261669e-01]
[ 1.14410730e+00]
[ 2.27793400e-01]
[-5.94494357e-01]
[-1.51205383e+00]
[-1.69291507e+00]
[-5.13698791e-01]
[ 1.08711824e+00]
[ 3.00863717e-01]
[-1.10712218e+00]
[ 1.17524075e+00]
[-4.42495461e-01]
[-1.79987964e-01]
[-1.31705374e+00]
[-1.05396185e+00]
[ 1.09347696e-01]
[ 4.60611603e-01]
[-2.36858160e+00]
[-2.01849225e+00]
[-1.18098082e+00]
[-1.32661831e+00]
[-1.90848977e+00]
[ 4.64829625e-01]
[ 1.16524170e+00]
[-2.42265330e-01]
[-7.98312865e-01]
[-5.08773745e-01]
[ 2.62360308e-01]
[-8.94123337e-04]
[-5.99908792e-01]
[-2.47155826e+00]
[ 1.00936886e+00]
[-3.91545602e-01]
[ 5.75874525e-01]
[-2.69184571e-01]
[-1.32624268e+00]
[-6.19362353e-01]
[ 9.23415013e-02]
[ 5.32280273e-01]
[-2.20546164e-01]
[ 1.14040279e+00]
[-4.40793491e-01]
[ 3.48160780e-01]
[ 5.04703298e-01]
[-1.53925613e-01]
[-5.45284752e-02]
[-2.12202990e-02]
[-5.18319747e-01]
[ 7.93671868e-01]
[-5.69416914e-01]
[-1.61545299e+00]
[-1.55467454e+00]
[ 2.83671899e-01]
[-4.94751828e-01]
```

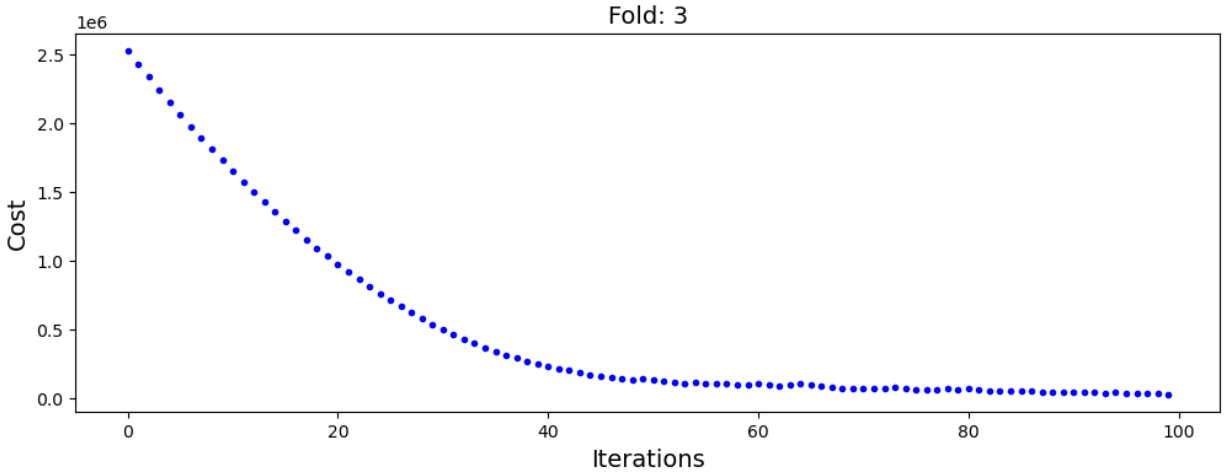
```
[ 6.27225513e-02]
[-3.25582545e-01]
[ 7.87743993e-01]
[ 3.57747904e-01]
[-6.09288843e-01]
[-1.32685364e+00]
[-1.94951006e-01]
[-3.83093253e-01]
[ 3.45772354e-01]
[ 2.12806945e-01]
[ 1.34263987e-01]
[-2.01128663e-01]
[-3.18138347e-01]
[-1.08264635e+00]
[-1.50374470e+00]
[-4.96076724e-01]
[-5.46113918e-01]
[-4.66727522e-01]
[ 5.10721710e-01]
[ 1.30895913e-01]
[-1.99332363e+00]
[-2.14979197e+00]
[-8.88706567e-01]
[ 5.55019602e-01]
[-5.47842459e-01]
[ 1.37390061e+00]
[-3.20706345e-01]
[-8.28300056e-01]
[ 1.11387663e+00]
[ 2.37819694e-01]
[ 6.97674761e-01]
[-3.92108963e-01]
[-2.43544570e-01]
[-1.92137967e-01]
[ 6.19653954e-01]
[ 6.46693210e-01]
[-3.26431187e+00]
[ 1.49008274e+00]
[ 4.84201862e-01]
[ 4.49564185e-02]
[-9.78143082e-02]
[-1.21803513e+00]
[-1.14970082e+00]
[-8.08360754e-01]
[-4.88047813e-01]
[-2.31309323e-01]
[-2.50115402e+00]
[-4.76544564e-01]
[-3.78365301e+00]
[-1.24156459e+00]
[ 6.10848161e-01]
[-9.21841813e-01]
[-1.56661355e+00]
[ 6.03938475e-01]
[-1.12924943e+00]
[ 1.52811353e+00]
[-1.60506276e+00]
[-7.65006561e-01]
[ 1.65984559e-01]
[-6.79467855e-01]
```

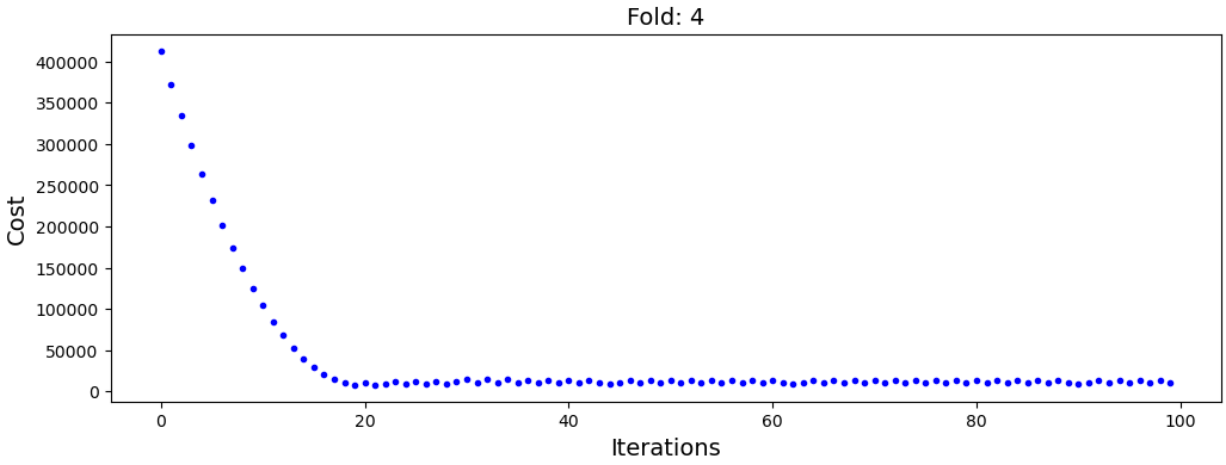
```
[-1.26039416e-01]
[ 3.47712381e-01]
[-1.61534744e+00]
[ 3.61454540e-01]
[ 4.44417802e-02]
[ 1.42524126e+00]
[-1.02838708e+00]
[-1.56705079e+00]
[-3.07491824e+00]
[-3.73725539e-02]
[ 7.42284792e-01]
[ 1.13918318e+00]
[ 1.25502626e+00]
[ 7.66297872e-02]
[-5.40943227e-01]
[-5.34674164e-01]
[-1.76234387e+00]
[ 7.21522860e-02]
[-1.04384377e+00]
[ 1.44069362e+00]
[-1.02046670e+00]
[-1.03829933e+00]
[-1.13272252e+00]
[-3.82558546e-02]
[ 9.63175559e-01]
[-5.62616375e-01]
[-8.13389096e-01]
[ 1.53379777e-01]
[-2.65661397e-01]
[-2.50659938e-01]
[-5.76187295e-01]
[-6.25434653e-01]
[-3.33525037e-01]
[-6.93692040e-01]
[-3.92033027e-02]
[ 9.97534311e-01]
[ 3.67422170e-01]
[-6.32162549e-01]
[ 5.24290282e-01]
[-4.18274706e-01]
[ 2.82944735e-01]
[ 7.96221794e-01]
[ 1.43790626e+00]
[ 1.11403565e+00]
[ 1.99460108e-01]
[ 9.04505510e-01]
[ 1.07061622e-01]
[ 6.45675584e-01]
[ 7.22803064e-01]
[-1.28913647e+00]
[-4.12986298e-01]
[-1.54319454e+00]
[ 1.11005508e+00]
[ 2.78619984e+00]
[ 2.05719473e-01]
[-4.06036114e-01]
[ 1.66515297e+00]
[ 6.64966961e-01]
[ 1.34113182e+00]
[ 9.31740357e-02]
```

```
 [ 9.08709599e-01]
 [-3.06182018e-01]
 [-1.92082318e-03]
 [-9.61951192e-01]
 [-3.18330895e-02]
 [ 1.58725080e+00]
 [-7.44528030e-01]
 [-6.06112339e-01]
 [-9.14956405e-01]
 [-3.90745963e-01]
 [-1.18206389e-01]
 [-1.23907441e+00]
 [-2.01063718e+00]
 [ 2.84693545e-01]
 [-3.10659049e-02]
 [ 4.95732343e-01]
 [ 1.62084716e+00]
 [ 9.15810588e-01]
 [ 1.19401721e+00]
 [ 8.37166875e-01]
 [ 7.55091169e-01]
 [ 5.16972109e-01]
 [ 8.46039848e-01]
 [ 4.25420833e-01]
 [-1.31348313e+00]
 [-4.63476614e-01]
 [ 3.37519713e-01]
 [ 6.10478031e-01]
 [ 1.59286481e-01]
 [ 9.58841516e-01]
 [ 7.37446743e-01]
 [ 5.88075634e-01]
 [-1.96897950e-01]
 [ 4.24016398e-01]]
Mean Square Error: 5939.832
```









While the data converges for all the batches, the mse values for these models is significantly higher as compared to the linear regression models.

Question G: Make predictions of the labels on the test data, using the trained model with chosen hyperparameters. Summarize performance using the appropriate evaluation metric. Discuss the results. Include thoughts about what further can be explored to increase performance.

In [81]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet, SG
from sklearn.metrics import mean_squared_error

warnings.filterwarnings("ignore")

# Create a DataFrame to store MSE values
mse_table = pd.DataFrame(columns=['Model', 'MSE'])

# Linear Regression
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
linear_reg_predictions = linear_reg.predict(X_test)
linear_reg_mse = mean_squared_error(y_test, linear_reg_predictions)
mse_table = mse_table.append({'Model': 'Linear Regression', 'MSE': linear_reg_m

# Ridge Regression
ridge_reg = Ridge(alpha=0.001)
ridge_reg.fit(X_train, y_train)
ridge_reg_predictions = ridge_reg.predict(X_test)
ridge_reg_mse = mean_squared_error(y_test, ridge_reg_predictions)
mse_table = mse_table.append({'Model': 'Ridge Regression', 'MSE': ridge_reg_mse

# Lasso Regression
lasso_reg = Lasso(alpha=0.001)
lasso_reg.fit(X_train, y_train)
lasso_reg_predictions = lasso_reg.predict(X_test)
lasso_reg_mse = mean_squared_error(y_test, lasso_reg_predictions)
mse_table = mse_table.append({'Model': 'Lasso Regression', 'MSE': lasso_reg_mse

# Elastic Net Regression
elastic_net = ElasticNet(alpha=0.001, l1_ratio=0.1)
elastic_net.fit(X_train, y_train)
elastic_net_predictions = elastic_net.predict(X_test)
elastic_net_mse = mean_squared_error(y_test, elastic_net_predictions)
mse_table = mse_table.append({'Model': 'Elastic Net', 'MSE': elastic_net_mse},

# SGDRegressor (Stochastic Gradient Descent)
sgd_reg = SGDRegressor(alpha=0.001, max_iter=1000, random_state=42)
sgd_reg.fit(X_train, y_train)
sgd_reg_predictions = sgd_reg.predict(X_test)
sgd_reg_mse = mean_squared_error(y_test, sgd_reg_predictions)
mse_table = mse_table.append({'Model': 'SGD Regression', 'MSE': sgd_reg_mse}, i

# Display the MSE values
print(mse_table)

warnings.resetwarnings()
```

```
              Model           MSE
0  Linear Regression  2.435591e+02
1   Ridge Regression  2.435591e+02
2   Lasso Regression  2.435578e+02
3        Elastic Net  2.435872e+02
4     SGD Regression  9.138920e+11
```

In [82]:
```
warnings.filterwarnings("ignore")

# Create a DataFrame to store MSE values
```

```
mse_table_poly = pd.DataFrame(columns=['Model', 'MSE_poly'])

# Linear Regression
linear_reg_poly = LinearRegression()
linear_reg_poly.fit(X_train_poly, y_train)
linear_reg_predictions_poly = linear_reg_poly.predict(X_test_poly)
linear_reg_mse_poly = mean_squared_error(y_test, linear_reg_predictions_poly)
mse_table_poly = mse_table_poly.append({'Model': 'Linear Regression', 'MSE_poly

# Ridge Regression
ridge_reg_poly = Ridge(alpha=0.001)
ridge_reg_poly.fit(X_train_poly, y_train)
ridge_reg_predictions_poly = ridge_reg_poly.predict(X_test_poly)
ridge_reg_mse_poly = mean_squared_error(y_test, ridge_reg_predictions_poly)
mse_table_poly = mse_table_poly.append({'Model': 'Ridge Regression', 'MSE_poly

# Lasso Regression
lasso_reg_poly = Lasso(alpha=0.001)
lasso_reg_poly.fit(X_train_poly, y_train)
lasso_reg_predictions_poly = lasso_reg_poly.predict(X_test_poly)
lasso_reg_mse_poly = mean_squared_error(y_test, lasso_reg_predictions_poly)
mse_table_poly = mse_table_poly.append({'Model': 'Lasso Regression', 'MSE_poly

# Elastic Net Regression
elastic_net_poly = ElasticNet(alpha=0.001, l1_ratio=0.1)  # Adjust l1_ratio as
elastic_net_poly.fit(X_train_poly, y_train)
elastic_net_predictions_poly = elastic_net_poly.predict(X_test_poly)
elastic_net_mse_poly = mean_squared_error(y_test, elastic_net_predictions_poly)
mse_table_poly = mse_table_poly.append({'Model': 'Elastic Net', 'MSE_poly': ela

# SGDRegressor (Stochastic Gradient Descent)
sgd_reg_poly = SGDRegressor(alpha=0.001, max_iter=1000, random_state=42)
sgd_reg_poly.fit(X_train_poly, y_train)
sgd_reg_predictions_poly = sgd_reg_poly.predict(X_test_poly)
sgd_reg_mse_poly = mean_squared_error(y_test, sgd_reg_predictions_poly)
mse_table_poly = mse_table_poly.append({'Model': 'SGD Regression', 'MSE_poly':

# Display the MSE values
print(mse_table_poly)

warnings.resetwarnings()
```

```
               Model      MSE_poly
0  Linear Regression  1.816824e+02
1   Ridge Regression  1.816743e+02
2   Lasso Regression  1.812181e+02
3        Elastic Net  1.817695e+02
4     SGD Regression  3.165407e+29
```

Summarizing results:

Analyzing the evaluation metric across linear and polynomial regression models, we notice that the MSE is better for polynomial regression as compared to linear regression. This could be because the data contains non-linear. Further, given that polynomial regression works better when there are multiple features. It is capable of finding relationships between features, which is something a plain linear regression model cannot do. Thus, indicating that polynomial regression is a better fit for this dataset.

Additionally, we can observe that Lasso performs better in both linear and polynomial regression models. This could indicate that the dataset is noisy. This fact needs to be taken into consideration while predicting future car prices.

Future work:

- Analyze which features could be dropped in order to improve performance
- Hypermetre tuning methods such as grid search can be used to improve the performance of the model.
- Test the models with other regression models such as Random Forest and other bagging or boosting techniques.

Ref: Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd edition).

Additional resources used:

- Kaggle notebook: https://www.kaggle.com/code/gadigevishalsai/car-price-prediction-challenge-eda-regression/notebook#Car-Price-Prediction
- Data Source: https://www.kaggle.com/datasets/deepcontractor/car-price-prediction-challenge/data