گزارش کدهای تمرین اول

درس ماشین لرنینگ

Katayoun kobraei - 99222084

Question 5 : implementing a linear regression model with mean absolute error

همان طور که توقع داشته میشد نتایج لایبرری سایکیت لرن بهتر و دقت بیشتری نسبت به مدل ما دارند. وقتی مدل رگرسیون خطی ساده ما پیاده سازی شد نتایج زیر بدست امد :

```
y_pred_model = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred_model)
print("Custom Linear Regression MAE:", mae)
```

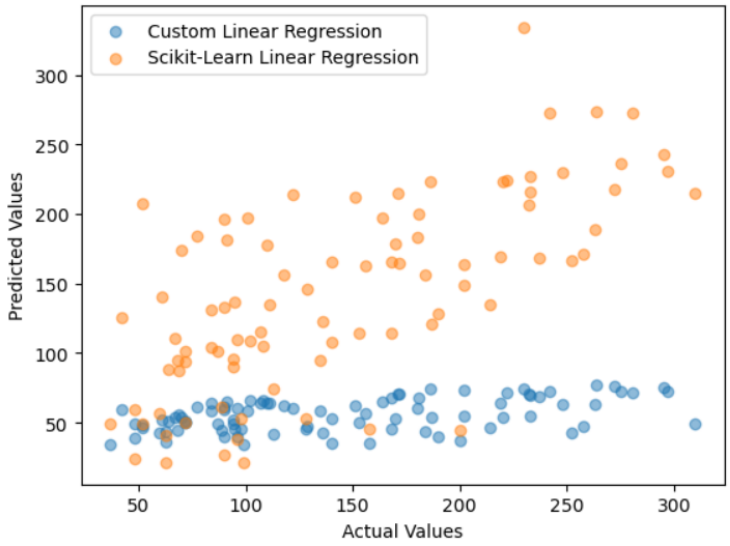
```
Custom Linear Regression MAE: 90.46646960667117
```

این در حالیست که این مقادیر برای مدل اثلی کتابخانه به صورت زیر است:

```
lr_sklearn = LinearRegression()
lr_sklearn.fit(X_train, y_train)
y_pred_sklearn = lr_sklearn.predict(X_test)
mae_sklearn = mean_absolute_error(y_test, y_pred_sklearn)
print("Scikit-Learn Linear Regression MAE:", mae_sklearn)
```

```
Scikit-Learn Linear Regression MAE: 43.48538185702934
```

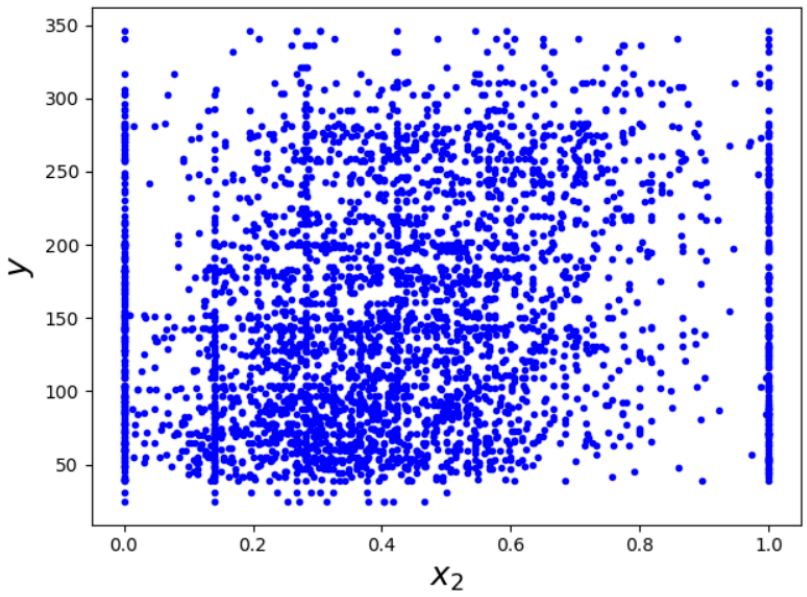
در نهایت اگر این دو مدل را در جدولی باهم مقایسه کنیم میبینیم که نتایج مدل کتابخانه سایکیت لرن بسیار منطقی تر از مدل ما شده است که منطقی است. زیرا مدل ما بسیار ساده و تنها تلاش شده بود ساده ترین الگوریتم در ان پیاده سازی شود. چه بسا با مهندسی فیچر و یا انتخاب لاست فانکشن های دیگر بتوان به مدل بهتری دست یافت.



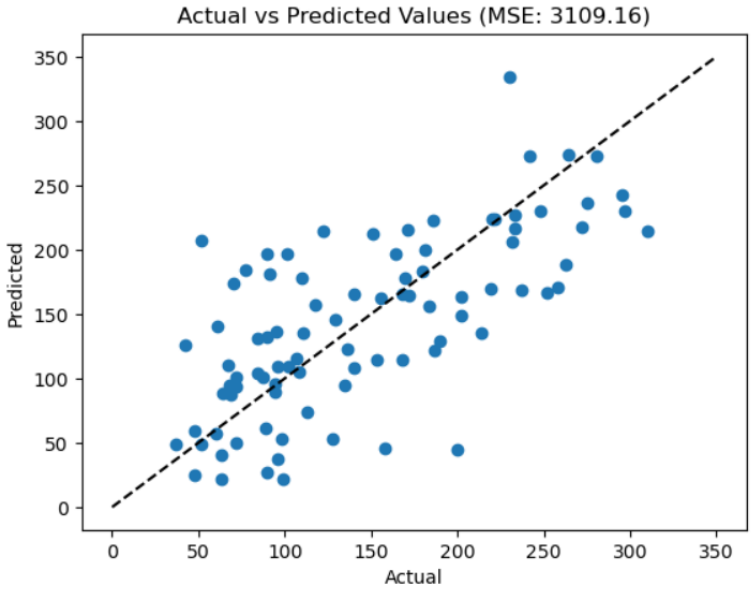
Question 5 : Implementing Linear Regression using the normal equation

در این مسئله با استفاده از معادله نرمال سعی میکنیم به تخمین بهتری برسیم. به طور کلی گردینت دیسنت از این تکنیک، معروف تر و پر کاربرد تر است و محبوبیت بیشتری نیز میان دیتاساینتیست ها دارد. ولی میتوانیم ببینیم که گاهی اوقات تکنیک معادله نرمال هم جواب مطلوبی به ما میرساند.

اگر برای دیتا و مقادیر تخمین زده شده داشته باشیم:



میبینیم که مدل ما توانسته تخمین تقریبا خوبی با استفاده از این تکنیک بزند:



Question 10 : implementing Forward and Backward Feature selection algorithms

میدانیم یکی از تکنیک های مهم در ماشین لرنینگ مهندسی فیچرها است. دو الگوریتم معروف برای این کار پیاده سازی شده است. همان طور که توقع میرود در نهایت فیچر های مهم تری بدست می اید و در عین حال از اورفیتینگ و اندرفیتینگ نیز جلوگیری میشود. در تابع فوروارد تمامی فیچر هارا داریم و هر بار یک فیچر انتخاب میشود و این کار تا زمانی ادامه میابد که دیگر فیچری قابل حذف شدن نباشد.

الگوریتم ما به گونه ای کار میکند که فیچر های انتخاب شده خود را چه در مرحله فوروارد و چه در مرحله بکوارد برمیگرداند و در عین حال مقدار "ام اس ای" را نیز محاسبه میکند:

testing

In [60]:
```
selected_features, best_mse = forward_feature_selection(X, y)
print("Forward features = ", selected_features)
print("MSE = ", best_mse)
```

```
Forward features =  [2, 8, 3, 4, 1, 5, 7, 9, 6, 0]
MSE =  2859.6903987680657
```

In [61]:
```
selected_features, best_mse = forward_feature_selection(X, y)
print("Forward features = ", selected_features)
print("MSE = ", best_mse)
```

```
Forward features =  [2, 8, 3, 4, 1, 5, 7, 9, 6, 0]
MSE =  2859.6903987680657
```

Question 12 :

در این بخش از تمارین سعی شده است مدل رگرسیون برای دیتاست Online News Popularity پیاده سازی شود. این دیتاست این یک مجموعه داده در دسترس عموم است که معمولاً در تحقیقات یادگیری ماشین و داده کاوی برای پیش‌بینی محبوبیت مقالات خبری آنلاین استفاده می‌شود. مجموعه داده شامل انواع مختلفی از ویژگی های مرتبط با مقالات خبری، مانند تعداد کلمات، اشتراک گذاری رسانه های اجتماعی، کلمات کلیدی و موارد دیگر است. متغیر هدف در مجموعه داده معمولاً تعداد اشتراک‌هایی است که یک مقاله خبری در رسانه‌های اجتماعی دریافت می‌کند، که می‌تواند برای پیش‌بینی محبوبیت یا ویروسی بودن مقالات خبری استفاده شود.

مجموعه داده محبوبیت اخبار آنلاین اغلب برای کارهایی مانند رگرسیون، طبقه بندی و انتخاب ویژگی استفاده می شود. معمولاً برای توسعه و ارزیابی مدل‌های یادگیری ماشینی برای پیش‌بینی محبوبیت مقالات خبری آنلاین بر اساس ویژگی‌های مختلف استفاده می‌شود. در این راه روی داده پیش پردازش هایی انجام می دهیم که دیتا برای کار مناسب تر شود. در این بین مدل با پیاده سازی روش هایی ازحالت پایه در امده و پیشرفته تر میشود. در نهایت هم نتیجه نهایی را چاپ کرده و میتوانیم ان را با مدل های معروف تر مقایسه کنیم.

هدف از این تمرین اشنایی کامل با فرایند های پیش پردازش و در نهایت بهترین راه برای رسیدن به دیتای بهتر برای اموزش با استفاده از راه های نمایش دیتا، پیدا کردن بهترین الگوریتم اموزش برای این کارو روش های مختلف برای ترین نتیجه بهتر و در نهایت زیاد کردن دقت و نشان دادن نتایج اموزش خواهد بود. در این راستا از لاس فانکشن های مختلف استفاده میشود تا به بیشترین دقت برسیم.

در ادامه گزارش به پیاده سازی کد میپردازیم که چگونه میتوان به هدف مسئله با این روش ها برسیم. در اغاز دیتاست موردنظر را ایمپورت میکنیم.

importing data

In [4]:
```
df = pd.read_csv("OnlineNewsPopularity.csv")
```

سپس بعد از ان به پیش پردازش های دیتا میپردازیم. پیش پردازش دیتا یک مرحله مهم در فرآیند تجزیه و تحلیل داده ها است زیرا به ما کمک می کند تا ویژگی های داده ها را درک کنیم، هر گونه مشکل کیفیت داده را شناسایی کنیم، و دیدی را راجب دیتا به دست آوریم که می تواند مهندسی ویژگی، انتخاب مدل و ارزیابی مدل را ارائه دهد.از نمایش اطلاعات اولیه شروع میکنیم تا هرگونه تغییر مورد نیاز را پیش از پیاده سازی مدل انجام دهیم.

In [5]:
```
print("Dataset shape: ", df.shape)
print("Columns: ", df.columns)
```

```
Dataset shape:  (39644, 61)
Columns:  Index(['url', ' timedelta', ' n_tokens_title', ' n_tokens_content',
       ' n_unique_tokens', ' n_non_stop_words', ' n_non_stop_unique_tokens',
       ' num_hrefs', ' num_self_hrefs', ' num_imgs', ' num_videos',
       ' average_token_length', ' num_keywords', ' data_channel_is_lifestyle',
       ' data_channel_is_entertainment', ' data_channel_is_bus',
       ' data_channel_is_socmed', ' data_channel_is_tech',
       ' data_channel_is_world', ' kw_min_min', ' kw_max_min', ' kw_avg_min',
       ' kw_min_max', ' kw_max_max', ' kw_avg_max', ' kw_min_avg',
       ' kw_max_avg', ' kw_avg_avg', ' self_reference_min_shares',
       ' self_reference_max_shares', ' self_reference_avg_sharess',
       ' weekday_is_monday', ' weekday_is_tuesday', ' weekday_is_wednesday',
       ' weekday_is_thursday', ' weekday_is_friday', ' weekday_is_saturday',
       ' weekday_is_sunday', ' is_weekend', ' LDA_00', ' LDA_01', ' LDA_02',
       ' LDA_03', ' LDA_04', ' global_subjectivity',
       ' global_sentiment_polarity', ' global_rate_positive_words',
       ' global_rate_negative_words', ' rate_positive_words',
       ' rate_negative_words', ' avg_positive_polarity',
       ' min_positive_polarity', ' max_positive_polarity',
       ' avg_negative_polarity', ' min_negative_polarity',
       ' max_negative_polarity', ' title_subjectivity',
       ' title_sentiment_polarity', ' abs_title_subjectivity',
       ' abs_title_sentiment_polarity', ' shares'],
      dtype='object')
```

میتوانیم در کنار این خلاصه ای از تایپ فیچرها و میانگین و ... شان داشته باشیم. سپس از روش توصیف برای نمایش آمار خلاصه ستون‌های عددی در مجموعه داده استفاده می‌کنیم و مقادیر گمشده را با استفاده از روش (Is_null() بررسی میکنیم.

```
In [6]: df.describe()
```

Out[6]:

	timedelta	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_unique_tokens	num_hrefs	num_self_hrefs	num_
count	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.00
mean	354.530471	10.398749	546.514731	0.548216	0.996469	0.689175	10.883690	3.293638	4.54
std	214.163767	2.114037	471.107508	3.520708	5.231231	3.264816	11.332017	3.855141	8.30
min	8.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	164.000000	9.000000	246.000000	0.470870	1.000000	0.625739	4.000000	1.000000	1.00
50%	339.000000	10.000000	409.000000	0.539226	1.000000	0.690476	8.000000	3.000000	1.00
75%	542.000000	12.000000	716.000000	0.608696	1.000000	0.754630	14.000000	4.000000	4.00
max	731.000000	23.000000	8474.000000	701.000000	1042.000000	650.000000	304.000000	116.000000	128.00

8 rows × 60 columns

find missing values

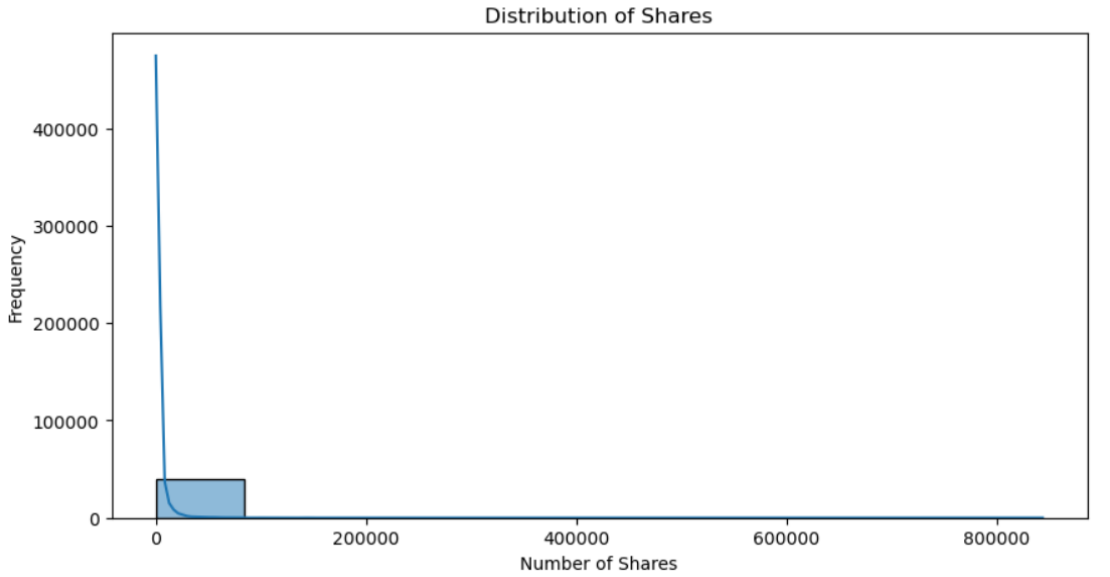
```
In [9]: print("Missing values: ")
        df.isnull().sum()
```

```
Missing values:

Out[9]: url                            0
        timedelta                      0
        n_tokens_title                 0
        n_tokens_content               0
        n_unique_tokens                0
                                      ..
        title_subjectivity             0
        title_sentiment_polarity       0
        abs_title_subjectivity         0
        abs_title_sentiment_polarity   0
        shares                         0
        Length: 61, dtype: int64
```

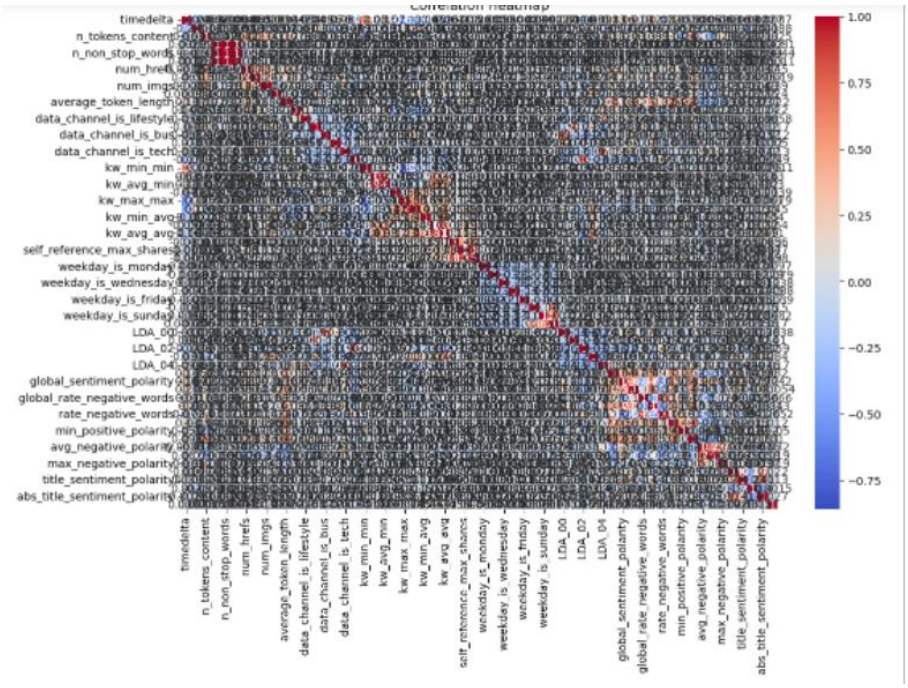
همان طور که میبینیم دیتاست ما مقدار میسینگ ندارد پس اماده پیش پردازش میباشد.

در مرحله بعد، ما از تکنیک‌های تجسم داده‌ها، مانند هیستوگرام و هیت مپ، برای به دست آوردن بینشی در مورد توزیع متغیر هدف (تعداد اشتراک‌ها) و همبستگی بین ویژگی‌ها استفاده می‌کنیم.

```
plt.figure(figsize=(10, 5))
sns.histplot(df[' shares'], bins=10, kde=True)
plt.xlabel('Number of Shares')
plt.ylabel('Frequency')
plt.title('Distribution of Shares')
plt.show()
```




```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

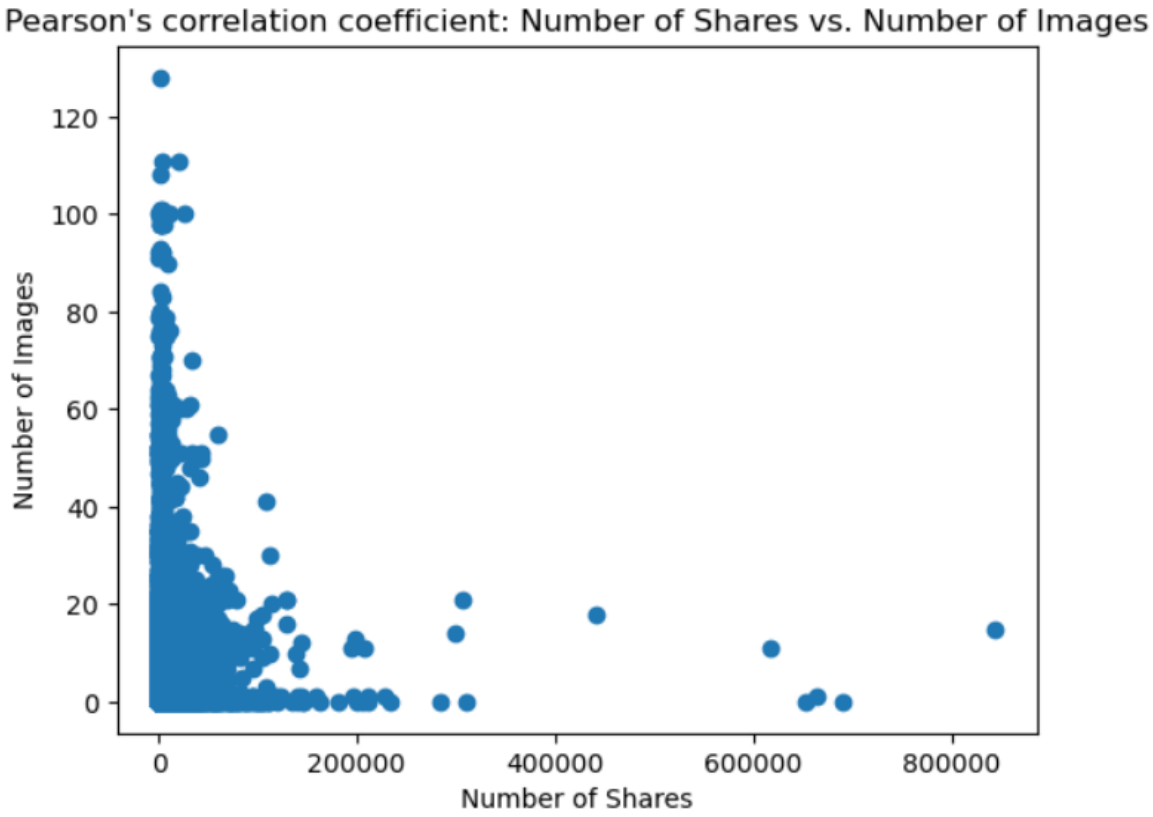


حال نوبت به ازمون های فرض اماری میرسد. در اینجا پنج بار تست در سه نوع تست انجام شده است:

1. Pearson's correlation coefficient:

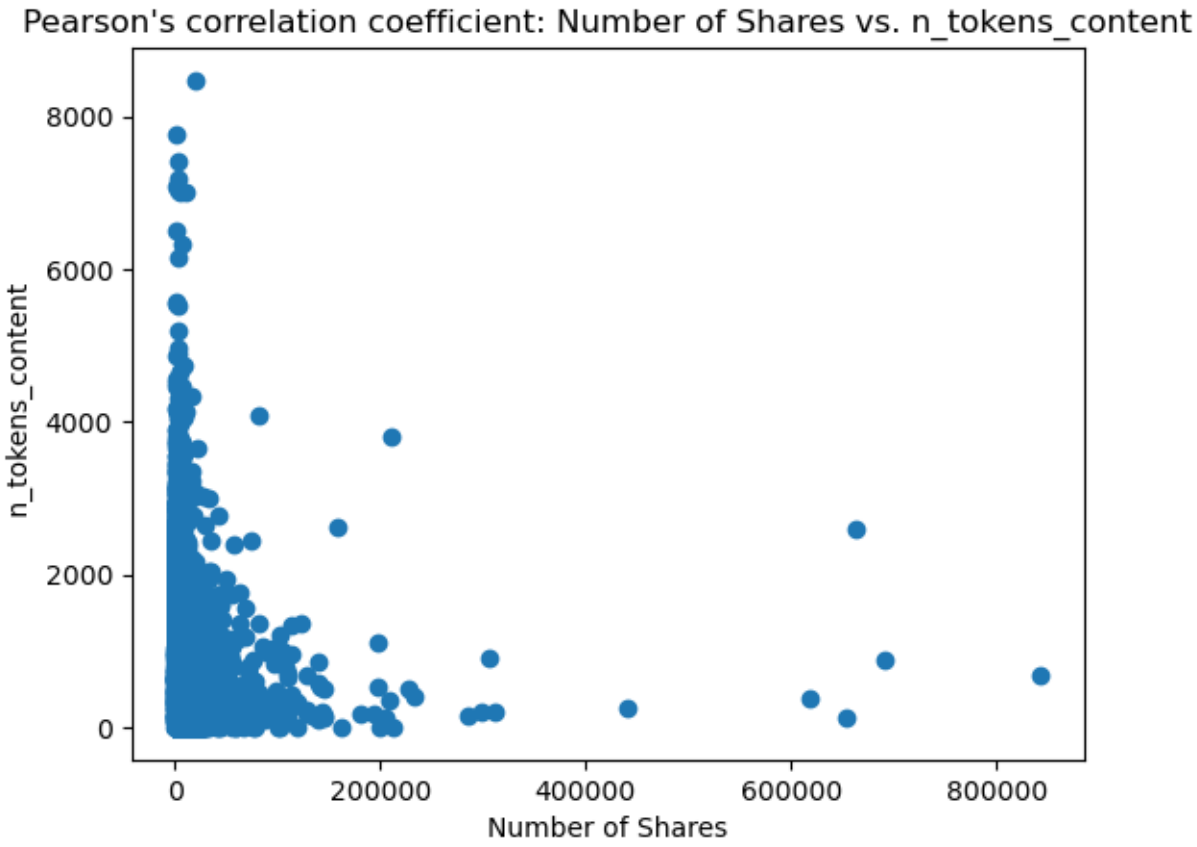
این ازمون برای ارزیابی قدرت و جهت رابطه خطی بین دو متغیر عددی (به عنوان مثال، تعداد اشتراک‌ها و تعداد تصاویر). این می تواند به تعیین اینکه آیا یک همبستگی آماری معنی دار بین دو متغیر عددی وجود دارد یا خیر کمک می کند. در اینجا از ضریب همبستگی پیرسون می توان برای اندازه گیری قدرت و جهت رابطه خطی بین دو متغیر پیوسته (تعداد عکس ها و تعداد اشتراک ها) و تعیین اینکه آیا همبستگی معنی داری وجود دارد استفاده کرد.

```
corr_coeff, p_value = stats.pearsonr(df[' shares'], df[' num_imgs'])

# Scatter plot of num_shares vs. num_images
plt.scatter(df[' shares'], df[' num_imgs'])
plt.xlabel('Number of Shares')
plt.ylabel('Number of Images')
plt.title("Pearson's correlation coefficient: Number of Shares vs. Number of Images")
plt.show()
```



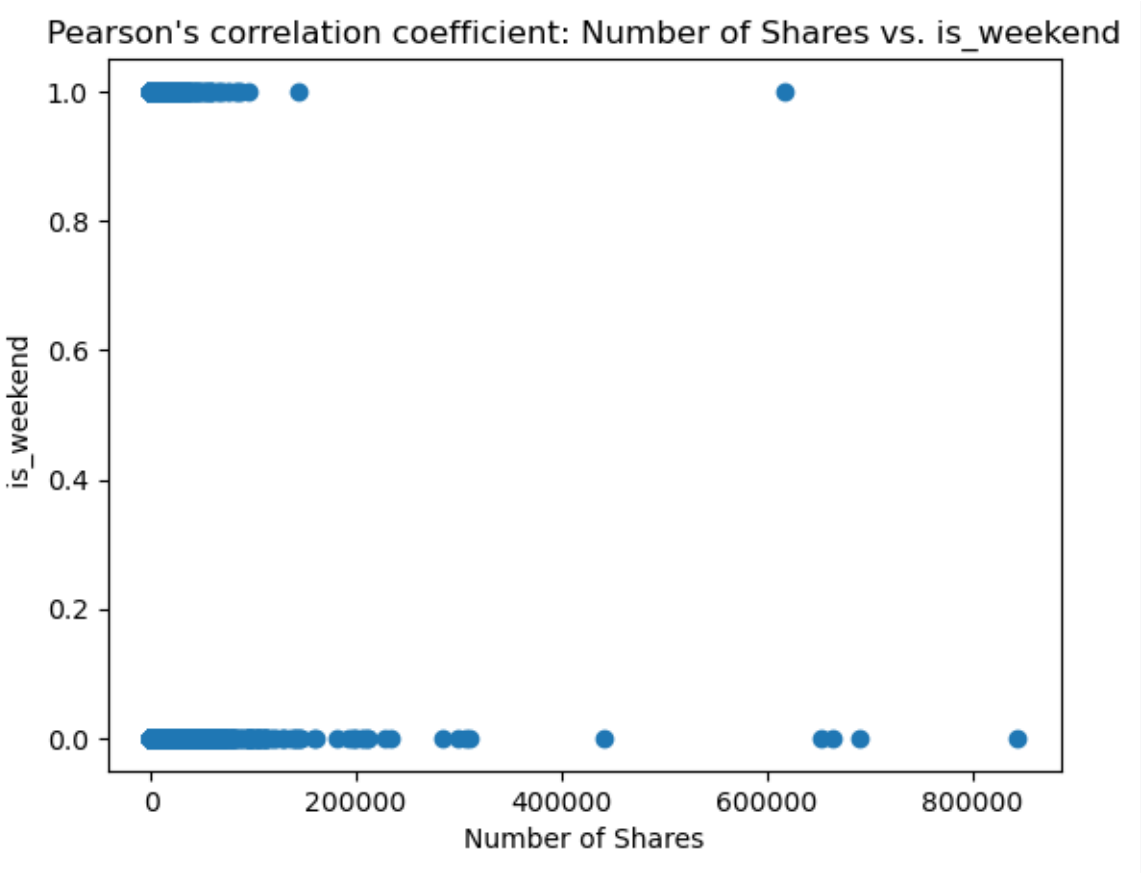
که در اینجا میبینی که ارتباط بالایی وجود دارد که اگر در مقادیر پایین از عکس استفاده شود میتوان تعداد اشتراکات زیادی داشت.

```
Pearson's correlation coefficient:
Correlation coefficient: 0.002458984345090837
p-value: 0.6244249147493814
```



دراینجا نیز میبنیم هرچه تعداد توکن بیشتر تعداد به اشتراک گذاری های بیشتری نیز خواهد داشت.

```
weekday_data = df[df[" is_weekend"] == 1][" shares"]
weekend_data = df[df[" is_weekend"] == 0][" shares"]

plt.scatter(df[' shares'], df[' is_weekend'])
plt.xlabel('Number of Shares')
plt.ylabel(' is_weekend')
plt.title("Pearson's correlation coefficient: Number of Shares vs. is_weekend")
plt.show()
```

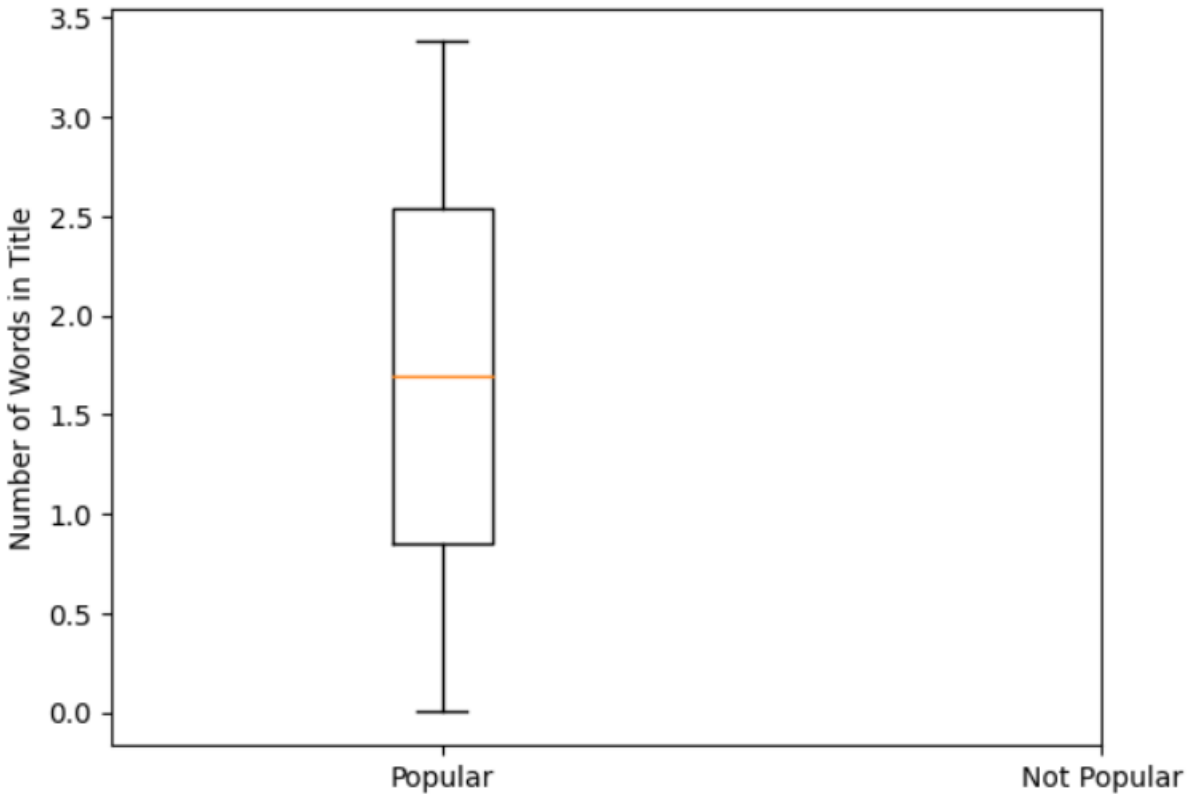


2. Independent t-test:

یک آزمون برای مقایسه میانگین‌های دو گروه مستقل برای تعیین تفاوت معنی‌دار آنها با یکدیگر استفاده می‌شود. این ازمون معمولاً زمانی استفاده می شود که دو گروه مجزا وجود دارد که به یکدیگر مرتبط نیستند و می خواهیم تعیین کنیم که آیا تفاوت آماری معنی داری بین میانگین یک متغیر خاص در این دو گروه وجود دارد یا خیر.

```
t_stat, p_value = stats.ttest_ind(weekday_data, weekend_data)
print("Independent t-test:")
print("t-statistic:", t_stat)
print("p-value:", p_value)

plt.boxplot([t_stat, p_value])
plt.xticks([1, 2], ['Popular', 'Not Popular'])
plt.ylabel('Number of Words in Title')
plt.show()
```

```
Independent t-test:
t-statistic: 3.3769109636398387
p-value: 0.0007337519086551708
```



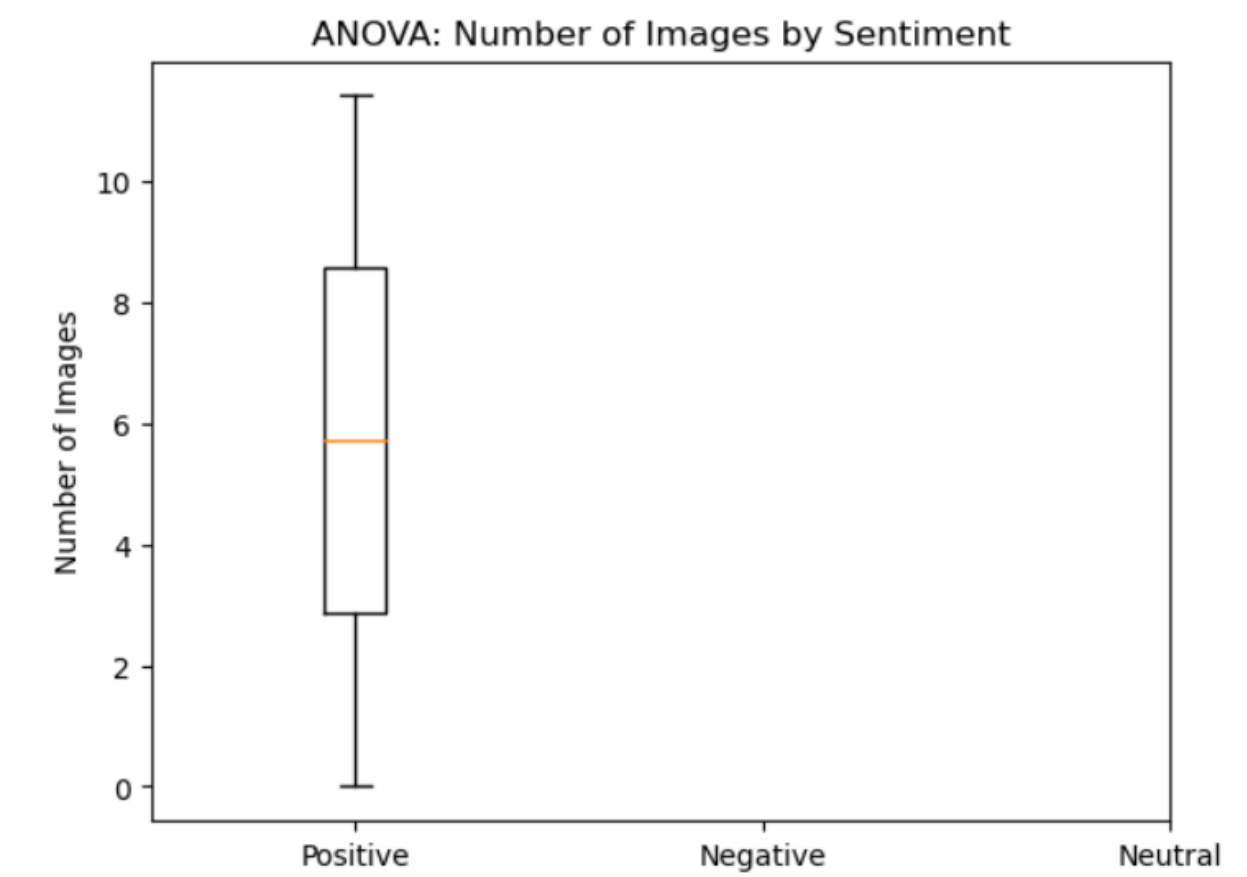
3. One-way ANOVA:

این آزمون برای مقایسه میانگین یک متغیر عددی در چندین گروه (به عنوان مثال، دسته مقاله یا احساسات). این می تواند به تعیین اینکه آیا تفاوت آماری معنی داری در میانگین متغیر عددی در بین گروه ها وجود دارد یا خیر کمک کند. در اینجا می توان برای مقایسه میانگین های بیش از دو گروه (دسته های مقاله) و تعیین اینکه آیا تفاوت معنی داری وجود دارد استفاده کرد.

```
f_stat, p_value = stats.f_oneway(*[df[df[" is_weekend"] == day][" shares"] for day in days_of_week])
print("One-way ANOVA:")
print("F-statistic:", f_stat)
print("p-value:", p_value)

# Plot boxplots for the three groups
plt.boxplot([f_stat, p_value])
plt.xticks([1, 2, 3], ['Positive', 'Negative', 'Neutral'])
plt.ylabel('Number of Images')
plt.title('ANOVA: Number of Images by Sentiment')
plt.show()
```

```
One-way ANOVA:
F-statistic: 11.403527656350944
p-value: 0.0007337519086632175
```

.



در ادامه به پیاده سازی مدل اصلی میرویم. پس از ایمپورت کردن کتابخانه‌های لازم و مجموعه داده و استخراج ویژگی‌ها و متغیر هدف شروع می‌کنیم. سپس، با استفاده از تابع train_test_split داده ها را به مجموعه های آموزشی و آزمایشی تقسیم می کنیم. در مرحله بعد، یک مدل رگرسیون خطی را با استفاده از مقداردهی می کنیم، مدل را با داده های آموزشی با استفاده از الگوریتم های مختلفی فیت میکنیم و با استفاده از روش پیش بینی روی مجموعه آزمون پیش بینی می کنیم. در نهایت، عملکرد مدل را با استفاده از میانگین مربعات خطا و امتیاز مربع "ار" به عنوان معیارهای ارزیابی ارزیابی می‌کنیم.

باید ابتدا متغیر های "یو ار ال" و "اشتراکات" یا همان هدف استیمیشن را از دیتاست کنار بگداریم و بعد دیتا ترین و تست را تشکیل دهیم.

Extract features and target variable

In [18]:
```
X = df.iloc[:, 2:-1]
y = df[' shares']
```

Split the data into training and testing sets

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Initialize and fit a linear regression model

In [20]:
```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[20]:
```
LinearRegression()
```

حال یک بار به صورت ساده و بدون اینکه الگوریتم خاصی برسیم مدل را پیاده سازی کرده و تست میکنیم.

Initialize and fit a linear regression model

In [20]:
```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

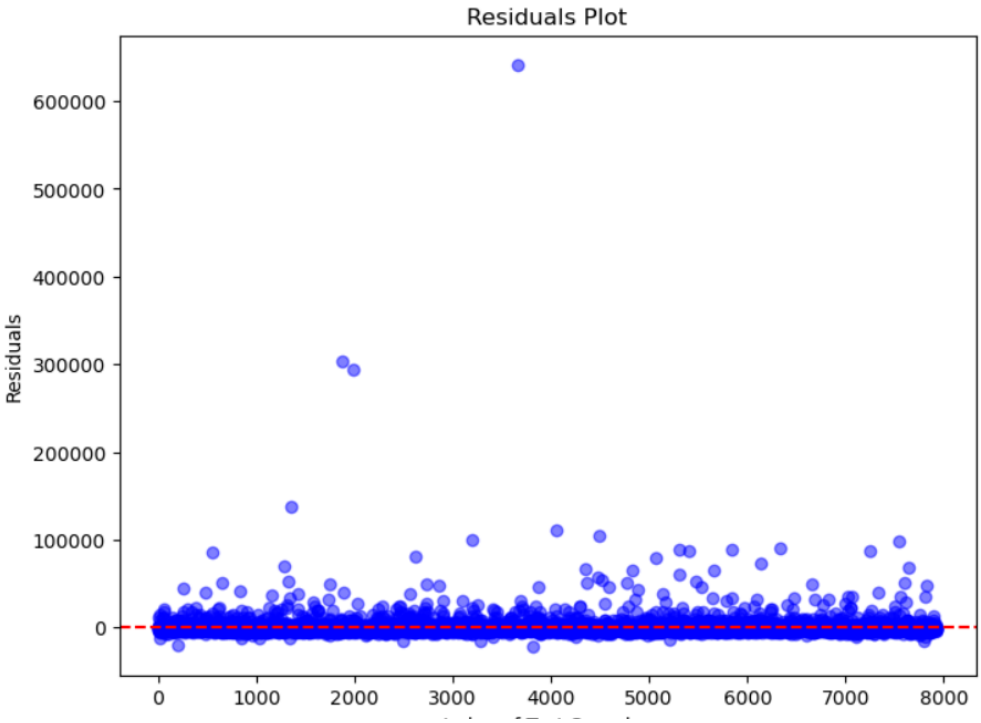
Out[20]:
```
LinearRegression()
```

Make predictions on the test set

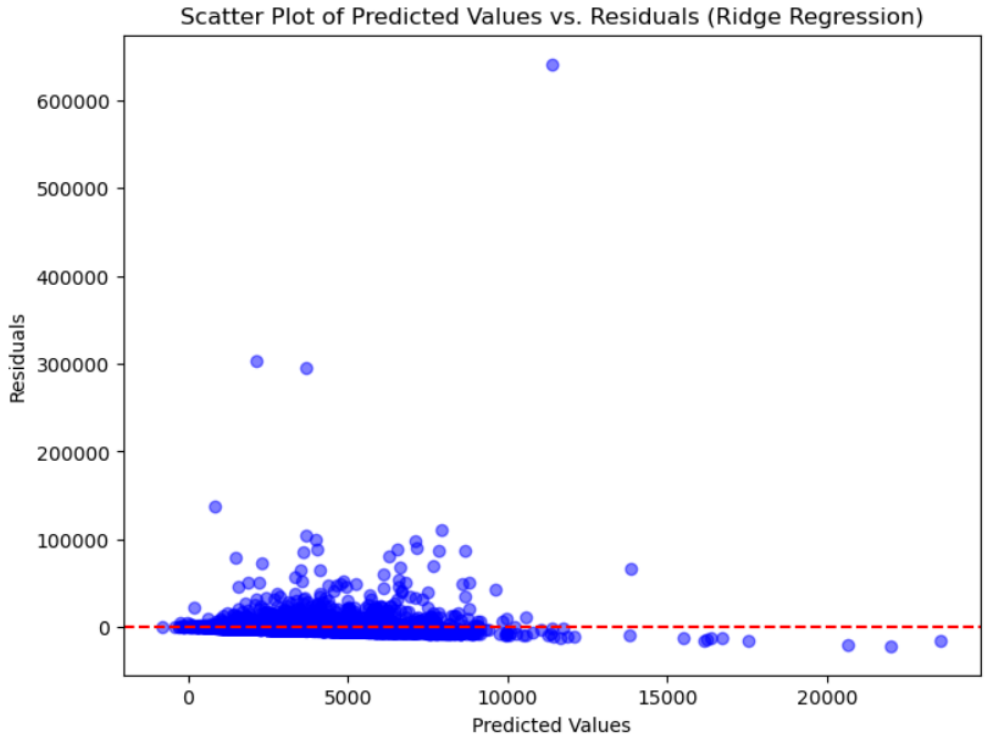
In [21]:
```
y_pred = regressor.predict(X_test)
```

Plot residuals

In [22]:
```
y_residuals = y_test - y_pred

plt.figure(figsize=(8, 6))
plt.scatter(np.arange(len(y_residuals)), y_residuals, c='blue', alpha=0.5)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Index of Test Samples')
plt.ylabel('Residuals')
plt.title('Residuals Plot')
plt.show()
```

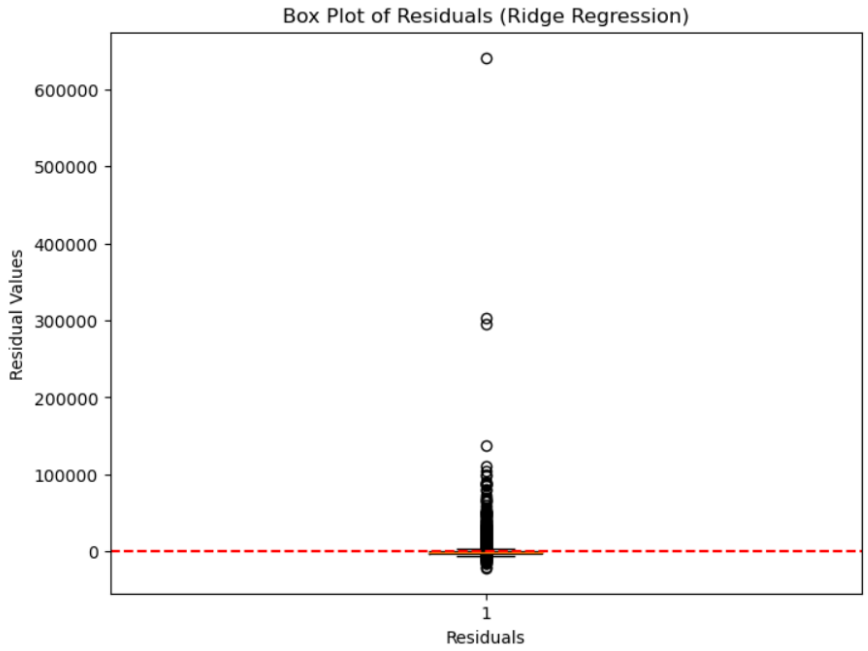

Residuals Plot
Residuals
600000
500000
400000
300000
200000
100000
0
0
1000
2000
3000
4000
5000
6000
7000
8000

در قسمت بعدی سعی میکنیم الگوریتم ریج رگرژن را پیاده سازی کنیم. آلفا قدرت رگیولاریزیشن است که مقادیر بالاتر منجر به منظم شدن بیشتر می شود. در نهایت، پیش‌بینی‌هایی روی داده‌های آزمایش انجام دادیم، میانگین مربعات خطا را محاسبه کردیم و برای ارزیابی عملکرد مدل، مربع ار را محاسبه کردیم. اگر باقیمانده ها به طور مساوی توزیع شده و دارای واریانس های مشابه در محدوده مقادیر پیش بینی شده باشند، نشان می دهد که پیش بینی های مدل در کل محدوده مقادیر پیش بینی شده سازگار است. اگر باقیمانده ها به طور تصادفی در اطراف خط y=0 پراکنده شوند، نشان می دهد که باقیمانده ها به طور سیستماتیک سوگیری ندارند و هیچ الگو یا روند خاصی را نشان نمی دهند. اگر باقیمانده‌ها یک الگو یا روند قابل تشخیص را نشان دهند، مانند فنینگ یا خوشه‌بندی به سمت یک انتهای نمودار، ممکن است نشان‌دهنده ناهمسانی باشد، به این معنی که واریانس باقیمانده‌ها در محدوده مقادیر پیش‌بینی‌شده ثابت نیست. این ممکن است نشان دهد که پیش‌بینی‌های مدل در مناطق خاصی از مقادیر پیش‌بینی‌شده دقیق یا کمتر قابل اعتماد هستند.



بعد از این هم الگوریتم لسو را عینا مانند ریج پیاده سازی میکنیم.

```
In [34]: from sklearn.linear_model import Lasso

In [35]: X = df.iloc[:, 2:-1]
         y = df.iloc[:, -1]

In [36]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In [37]: lasso = Lasso(alpha=1.0)

In [38]: lasso.fit(X_train, y_train)

C:\Users\katti\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: C
ot converge. You might want to increase the number of iterations, check the scale of the fea
risation. Duality gap: 1.981e+12, tolerance: 4.402e+08
  model = cd_fast.enet_coordinate_descent(

Out[38]: Lasso()

In [39]: # Predict on the testing data
         y_pred = lasso.predict(X_test)

In [41]: plt.figure(figsize=(8, 6))
         plt.boxplot(y_residuals)
         plt.axhline(y=0, color='red', linestyle='--')
         plt.xlabel('Residuals')
         plt.ylabel('Residual Values')
         plt.title('Box Plot of Residuals (Ridge Regression)')
```



در این مرحله برای گرفتن نتیجه بهتر به بررسی فیچرها میپردازیم که شامل اسکیل کردن فیچر ها و پلینومیال کردن مدل میشود.

ابتدا نیاز است متد های مختلف اسکیل کردن فیچرها را امتحان کنیم و هر یک را بررسی کنیم. اسکیل کردن ویژگی های ورودی اغلب می تواند عملکرد مدل های رگرسیون را بهبود ببخشد. در اینجا ما 3 متد اسکیل کردن فیچرها را انجام میدهیم. اولین متد، متد معروف Min-Max Scaling میباشد. این روش ویژگی ها را در یک محدوده خاص، معمولاً بین 0 و 1 مقیاس می کند. تأثیر این روش این است که مقادیر ویژگی‌ها را به یک محدوده مشترک تبدیل می‌کند، که می‌تواند به جلوگیری از تسلط ویژگی‌های با مقادیر بزرگتر بر مدل در طول آموزش کمک کند.

Method1 : Min-Max scailing

In [97]:
```
scaler = MinMaxScaler()
X_train_minmax = scaler.fit_transform(X_train)
X_test_minmax = scaler.transform(X_test)
```

In [101]:
```
ridge.fit(X_train_minmax, y_train)
y_pred_minmax = ridge.predict(X_test_minmax)
mse_minmax = mean_squared_error(y_test, y_pred_minmax)
r2_minmax = r2_score(y_test, y_pred_minmax)
```

متد بعدی، متد Standard Scaling میباشد که ویژگی ها را برای داشتن میانگین صفر و واریانس واحد مقیاس می کند. اثر مقیاس‌بندی استاندارد این است که مقادیر ویژگی‌ها را حول صفر متمرکز می‌کند و آنها را برای داشتن واریانس‌های مشابه مقیاس می‌دهد، که می‌تواند به جلوگیری از بی‌ثباتی عددی کمک کند و فرآیند بهینه‌سازی را روان‌تر کند.

Method2 : Standard Scaling

In [102]:
```
scaler = StandardScaler()
X_train_standard = scaler.fit_transform(X_train)
X_test_standard = scaler.transform(X_test)
```

In [103]:
```
ridge.fit(X_train_standard, y_train)
y_pred_standard = ridge.predict(X_test_standard)
mse_standard = mean_squared_error(y_test, y_pred_standard)
r2_standard = r2_score(y_test, y_pred_standard)
```

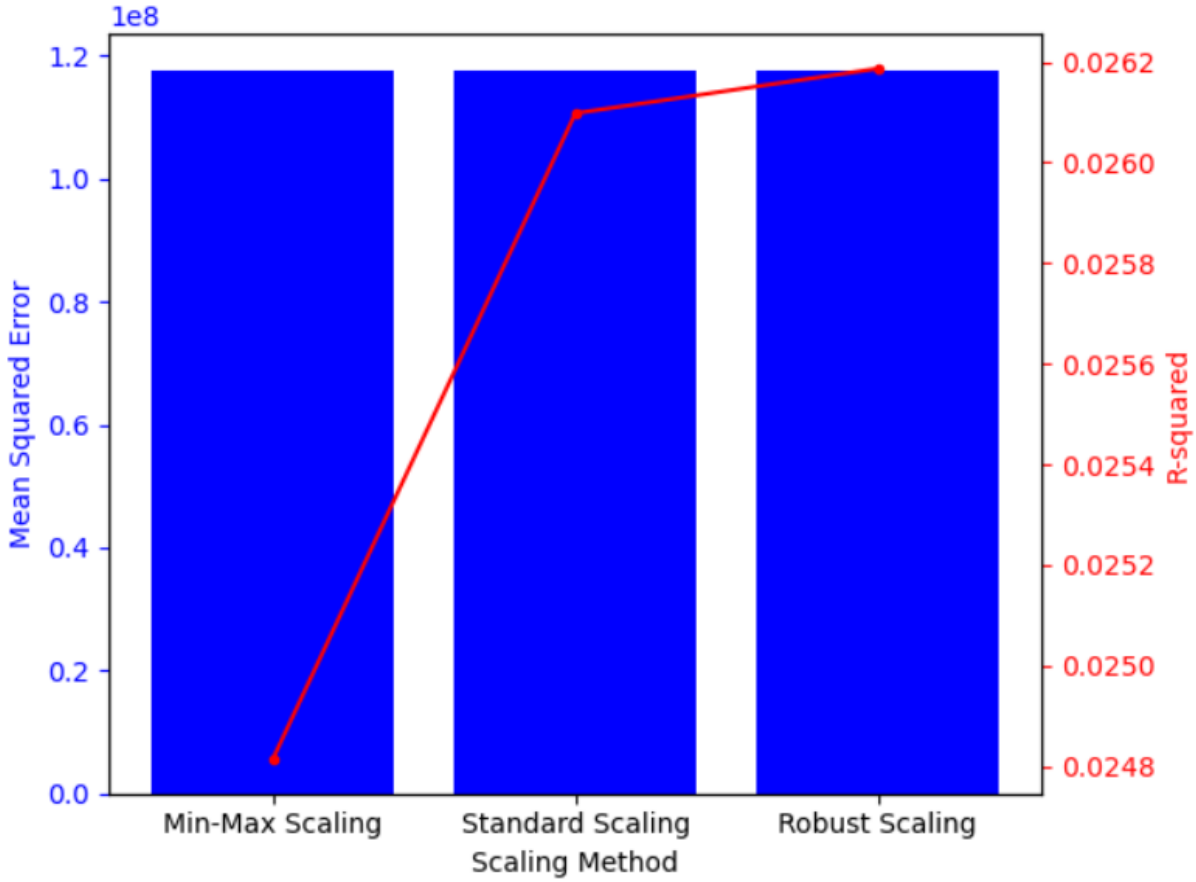
متد اخر متد Robust Scaling می باشد که با استفاده از آمارهای قوی که حساسیت کمتری نسبت به وجود نقاط پرت دارند، ویژگی ها را مقیاس می کند. از فرمول (x - mean) / IQR استفاده می کند که در آن ایکس مقدار ویژگی اصلی است، میانه میانگین مشخصه است، و محدوده بین ربعی ویژگی است (صدک 75 - صدک 25). تأثیر مقیاس‌گذاری قوی این است که تأثیر نقاط پرت را بر فرآیند مقیاس‌گذاری کاهش می‌دهد و آن را برای مجموعه‌های داده با مقادیر پرت بالقوه مناسب‌تر می‌کند.

Method3 : Robust Scaling

In [89]:
```
scaler = RobustScaler()
X_train_robust = scaler.fit_transform(X_train)
X_test_robust = scaler.transform(X_test)
```

In [90]:
```
ridge.fit(X_train_robust, y_train)
y_pred_robust = ridge.predict(X_test_robust)
mse_robust = mean_squared_error(y_test, y_pred_robust)
r2_robust = r2_score(y_test, y_pred_robust)
```

در نمودار، مشاهده می‌کنید که میانگین مربعات خطا برای مقیاس‌بندی Min-Max Scaling در مقایسه با روش‌های دیگر کمتر است، که نشان‌دهنده عملکرد بهتر مدل از نظر خطاهای پیش‌بینی کمتر است. مقدار R-squared نیز برای مقیاس حداقل حداکثری بالاتر است، که نشان دهنده تناسب بهتر مدل با داده ها است.



بعد از این سراغ اضافه کردن فیچر های پولینومیال میرویم. اثراین کار این است که به مدل اجازه می دهد تا روابط غیر خطی بین ویژگی ها و متغیر هدف را ثبت کند. درجات بالاتر از ویژگی‌های چند جمله‌ای ممکن است منجر به مدل‌های پیچیده‌تر با افزایش توانایی در برازش داده‌های آموزشی شود، اما ممکن است خطر اورفیتینگ را نیز افزایش دهد. به طور کلی، درجات پایین‌تری از ویژگی‌های چند جمله‌ای ممکن است برای جلوگیری از اورفیتینگ ترجیح داده شوند، در حالی که درجات بالاتر ممکن است برای ثبت الگوهای پیچیده‌تر در داده‌ها مفید باشند. مهم است که با درجات مختلف ویژگی های چند جمله ای آزمایش شود و یکی را انتخاب شود که پیچیدگی و عملکرد مدل را در مجموعه داده خاص به بهترین شکل متعادل کند. در مثال ارائه شده در بالا با درجه 2 برای ویژگی های چند جمله ای، تأثیر بر عملکرد مدل را می توان با بررسی میانگین مربعات خطا ارزیابی کرد، که معیاری است برای اینکه چقدر پیش بینی های مدل با مقادیر هدف واقعی مطابقت دارند.

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler, PolynomialFeatures
```

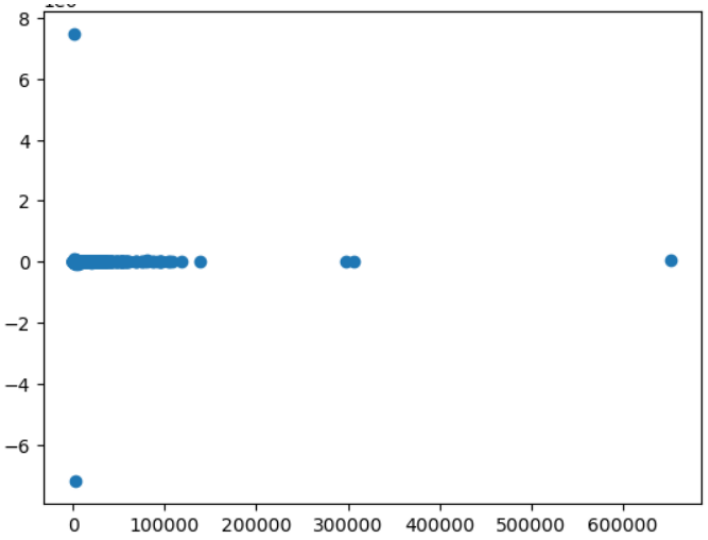
```
scaling_methods = {
    'Min-Max Scaling': MinMaxScaler(),
    'Standard Scaling': StandardScaler(),
    'Robust Scaling': RobustScaler()
}
```

```
polynomial_degrees = [1, 2, 3]
```

```
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
```

```
model = LinearRegression()
model.fit(X_train_poly, y_train)
y_pred = model.predict(X_test_poly)
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error (MSE):', mse)
```

در نمودار مقادیر کمترنشان‌دهنده عملکرد بهتر است، زیرا به این معنی است که مقادیر پیش‌بینی‌شده به مقادیر واقعی نزدیک‌تر هستند.



در این قسمت از تمرین از GridSearchCV همراه RandomizedSearchCV را پیاده سازی میکنیم. ما ابتدا از ان برای انجام جستجوی شبکه ای روی یک شبکه هایپرپارامتر از پیش تعریف شده برای مدل اصلی استفاده می کنیم. سپس از بهترین هایپرپارامترهای بدست آمده از گریدسرچ تی وی برای آموزش یک مدل رگرسیون خطی با آن فراپارامترها استفاده می کنیم. به طور مشابه، ما از رندومایزد سرچ برای اجرا استفاده می کنیم.

In [121]:
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

In [122]:
```
model = LinearRegression()
```

Define hyperparameter grid for GridSearchCV

In [123]:
```
param_grid = {
    'fit_intercept': [True, False],
    'normalize': [True, False]
}
```

Define hyperparameter grid for GridSearchCV

In [124]:
```
param_grid = {
    'fit_intercept': [True, False],
    'normalize': [True, False]
}
```

Initialize the random forest regressor model

```
from sklearn.ensemble import RandomForestRegressor
```

```
rf_model = RandomForestRegressor()
```

Define hyperparameter distribution for RandomizedSearchCV

```
param_dist = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

random_search = RandomizedSearchCV(rf_model, param_distributions=param_dist, n_iter=10, cv=5, scoring='neg_mean_squared_error')
random_search.fit(X_train, y_train)

best_params = random_search.best_params_
best_rf_model = RandomForestRegressor(**best_params)
best_rf_model.fit(X_train, y_train)
y_pred = best_rf_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("R-squared (R2) Score:", r2)
```

در این قسمت میخواهیم متد های رفت و برگشت فیچرسلکشن را اجرا کنیم. متد فوروارد روش رایجی است که در یادگیری ماشین برای انتخاب زیرمجموعه ای از ویژگی ها از مجموعه بزرگتر ویژگی ها بر اساس عملکرد پیش بینی آنها استفاده می شود. ایده اصلی این است که با مجموعه ای خالی از ویژگی های انتخاب شده شروع کنید و به طور مکرر یک ویژگی را در یک زمان اضافه کنید و ویژگی را انتخاب کنید که بهترین بهبود را در عملکرد مدل ارائه می دهد.

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

selected_features = []

while len(selected_features) < X.shape[1]:
    best_feature = None
    best_mse = float('inf')

    # Iterate over features
    for feature in X.columns:
        if feature not in selected_features:
            features = list(selected_features) + [feature]
            X_train_selected = X_train[features]
            X_val_selected = X_val[features]
            model = LinearRegression()
            model.fit(X_train_selected, y_train)

            y_val_pred = model.predict(X_val_selected)
            mse = mean_squared_error(y_val, y_val_pred)

            # Update best feature if necessary
            if mse < best_mse:
                best_feature = feature
                best_mse = mse

    # Add the best feature to the set of selected features
    selected_features.append(best_feature)
    print(f'Selected Feature: {best_feature}, MSE: {best_mse}')
```

```
# Choose final feature subset
final_feature_subset = selected_features[:-1]  # Remove last feature added as it resulted in the worst performance

X_train_final = X_train[final_feature_subset]
X_val_final = X_val[final_feature_subset]
model_final = LinearRegression()
model_final.fit(X_train_final, y_train)
y_val_pred_final = model_final.predict(X_val_final)
mse_final = mean_squared_error(y_val, y_val_pred_final)

X_test = ...
y_test = ...
X_test_final = X_test[final_feature_subset]
y_test_pred_final = model_final.predict(X_test_final)
mse_test_final = mean_squared_error(y_test, y_test_pred_final)

print(f'Final Feature Subset: {final_feature_subset}')
print(f'Validation MSE with Final Model: {mse_final}')
print(f'Test MSE with Final Model: {mse_test_final}')
```

به صورتی که برای فیچر های انتخاب شده خواهیم داشت:

```
Selected Feature:  LDA_00, MSE: 117132801.34588273
Selected Feature:  num_videos, MSE: 117155814.46287051
Selected Feature:  self_reference_max_shares, MSE: 117180149.96496108
Selected Feature:  kw_max_max, MSE: 117207370.29463711
Selected Feature:  n_tokens_content, MSE: 117267470.26899552
Selected Feature:  LDA_03, MSE: 117340282.3776753
Selected Feature:  LDA_02, MSE: 117341110.86502957
Selected Feature:  data_channel_is_bus, MSE: 117421313.41471866
Selected Feature:  n_tokens_title, MSE: 117512637.27801216
Final Feature Subset: [' kw_avg_avg', ' self_reference_min_shares', ' kw_max_avg', ' kw_min_avg', ' avg_negative_polarity', ' k
w_min_min', ' data_channel_is_entertainment', ' num_imgs', ' average_token_length', ' weekday_is_monday', ' num_hrefs', ' num_s
elf_hrefs', ' data_channel_is_lifestyle', ' is_weekend', ' min_positive_polarity', ' kw_min_max', ' title_sentiment_polarity',
' self_reference_avg_sharess', ' data_channel_is_socmed', ' data_channel_is_world', ' global_rate_positive_words', ' data_chann
el_is_tech', ' kw_avg_max', ' LDA_04', ' weekday_is_tuesday', ' n_non_stop_unique_tokens', ' n_non_stop_words', ' max_positive_
polarity', ' num_keywords', ' kw_avg_min', ' LDA_01', ' global_rate_negative_words', ' rate_positive_words', ' rate_negative_wo
rds', ' n_unique_tokens', ' weekday_is_friday', ' title_subjectivity', ' abs_title_sentiment_polarity', ' avg_positive_polarit
y', ' weekday_is_saturday', ' weekday_is_sunday', ' kw_max_min', ' abs_title_subjectivity', ' min_negative_polarity', ' max_neg
ative_polarity', ' weekday_is_thursday', ' weekday_is_wednesday', ' global_subjectivity', ' global_sentiment_polarity', ' LDA_0
0', ' num_videos', ' self_reference_max_shares', ' kw_max_max', ' n_tokens_content', ' LDA_03', ' LDA_02', ' data_channel_is_bu
s']
```

بعد از ان به سراغ پیاده سازی متد بکوارد همانند متد قبلی میرویم:

```python
selected_features = set(X.columns)

# Backward feature selection
mse_vals = []
while len(selected_features) > 1:
    best_feature = None
    best_mse = float('inf')

    # Iterate over features
    for feature in selected_features:
        # Remove the feature from the set of selected features
        features = list(selected_features - set([feature]))
        X_train_selected = X_train[features]
        X_val_selected = X_val[features]
        model = LinearRegression()
        model.fit(X_train_selected, y_train)

        y_val_pred = model.predict(X_val_selected)
        mse = mean_squared_error(y_val, y_val_pred)

        if mse < best_mse:
            best_feature = feature
            best_mse = mse

    # Remove the best feature from the set of selected features
    selected_features.remove(best_feature)
    mse_vals.append(best_mse)
```

```
final_feature_subset = list(selected_features)
X_train_final = X_train[final_feature_subset]
X_val_final = X_val[final_feature_subset]
model_final = LinearRegression()
model_final.fit(X_train_final, y_train)
y_val_pred_final = model_final.predict(X_val_final)
mse_final = mean_squared_error(y_val, y_val_pred_final)

# Evaluate final model performance on the test set
X_test = ...  # Load and preprocess the test set
y_test = ...  # Load the target variable for the test set

print(f'Final Feature Subset: {final_feature_subset}')
print(f'Validation MSE with Final Model: {mse_final}')
```

در قسمت اخر این تمرین متدهای مختلف برای لاس فانکشن را پیاده سازی میکنیم. اولین متد ، متدAbsolute Error میباشد. در این مثال، ما تابعی تعریف می‌کنیم که میانگین خطای مطلق را با گرفتن تفاوت مطلق بین مقادیر هدف واقعی و مقادیر پیش‌بینی‌شده محاسبه می‌کند و سپس میانگین تفاوت‌های مطلق را می‌گیرد. سپس از این تابع برای محاسبه"ام آ ای" پیش‌بینی‌های مدل رگرسیون خطی در مجموعه آزمایشی استفاده می‌کنیم.

```
def absolute_error(y_true, y_pred):
    return np.mean(np.abs(y_true - y_pred))

model = LinearRegression()
model.fit(X_train, y_train)
y_val_pred = model.predict(X_val)

mae = absolute_error(y_val, y_val_pred)
print(f'Mean Absolute Error: {mae}')

mae_sklearn = mean_absolute_error(y_val, y_val_pred)
```

```
Mean Absolute Error: 2999.186075775858
```

از آنجایی که اجرای تابع از دست دادن خطای مطلق با مدل رگرسیون خطی مشابه با استفاده از تابع ضرر میانگین مربعات خطا است، عملکرد مدل را می توان با استفاده از معیارهای مشابه، مانند امتیاز مربع ار و ریشه میانگین مربعات خطا ارزیابی کرد.

متد بعدی که پیاده سازی میشود، متد Epsilon-sensetive میباشد.

در این مثال هم، یک تابع تعریف می‌کنیم که خطای حساس به اپسیلون را با گرفتن تفاوت مطلق بین مقادیر هدف واقعی و مقادیر پیش‌بینی‌شده محاسبه می‌کند و سپس مقدار اپسیلون را از تفاوت‌های مطلق کم می‌کند. اگر اختلاف مطلق کوچکتر از اپسیلون باشد، خطا روی صفر تنظیم می شود. در نهایت، میانگین خطاهای حاصل را برای محاسبه خطای حساس به اپسیلون می گیریم.

```python
def epsilon_sensitive_error(y_true, y_pred, epsilon=0.1):
    errors = np.maximum(0, np.abs(y_true - y_pred) - epsilon)
    return np.mean(errors)

model = LinearRegression()
model.fit(X_train, y_train)
y_val_pred = model.predict(X_val)

epsilon = 0.1 # Set the value of epsilon
ese = epsilon_sensitive_error(y_val, y_val_pred, epsilon)
print(f'Epsilon-Sensitive Error (epsilon={epsilon}): {ese}')

mae_sklearn = mean_absolute_error(y_val, y_val_pred)
print(f'Mean Absolute Error (from sklearn): {mae_sklearn}')
```

```
Epsilon-Sensitive Error (epsilon=0.1): 2999.08607577583
Mean Absolute Error (from sklearn): 2999.186075775861
```

مقدار خطای حساس به اپسیلون به مقدار اپسیلون بستگی دارد که یک آستانه است. با تنظیم مقدار اپسیلون، می توانید حساسیت خطا به تفاوت بین مقادیر پیش بینی شده و واقعی را کنترل کنید. مقدار بزرگتر اپسیلون باعث می شود که خطا نسبت به تفاوت های کوچک تحمل بیشتری داشته باشد، در حالی که مقدار کمتر اپسیلون باعث می شود خطا کمتر تحمل شود و تفاوت های کوچک را بیشتر جریمه می کند. مهم است که مقدار مناسبی از اپسیلون را بر اساس دامنه مشکل و الزامات خاص کار پیش بینی خود انتخاب کنید. در این مثال، ما از مقدار اپسیلون = 0.1 استفاده کردیم، اما می‌توانید مقادیر مختلف را آزمایش کرد تا بهترین را برای مجموعه داده پیدا شود.

در نهایت اخرین تابعی که پیاده سازی میکنیم، متد Huber میباشد. در این قسمت نیز متدی برای این فانکشن طراحی میکنیم. پارامتر اپسیلون آستانه انتقال بین تلفات مجذور و تلفات خطی را در تابع ضرر هوبر مشخص می کند. تابع ما برای محاسبه ضرر هوبر از ابتدا با در نظر گرفتن مقدار دلتا که معادل پارامتر اپسیلون است، تعریف شده است. مقدار ار دو محاسبه شده و ریشه میانگین مربعات خطا را می توان برای ارزیابی عملکرد مدل آموزش دیده با تابع ضرر هوبر استفاده کرد.

```python
def huber_error(y_true, y_pred, delta=1.0):
    errors = np.abs(y_true - y_pred)
    mask = errors <= delta
    squared_errors = np.where(mask, 0.5 * np.square(errors), delta * (errors - 0.5 * delta))
    return np.mean(squared_errors)

model = LinearRegression()
model.fit(X_train, y_train)
y_val_pred = model.predict(X_val)

delta = 1.0 # Set the value of delta
huber = huber_error(y_val, y_val_pred, delta)
print(f'Huber Error (delta={delta}): {huber}')

mae_sklearn = mean_absolute_error(y_val, y_val_pred)
print(f'Mean Absolute Error (from sklearn): {mae_sklearn}')
```

```
Huber Error (delta=1.0): 2998.6861374500622
Mean Absolute Error (from sklearn): 2999.186075775861
```

تابع هوبر یک پارامتر آستانه به نام «دلتا» را معرفی می‌کند که نقطه‌ای را تعیین می‌کند که در آن تابع ضرر از رفتاری مانند "ام اس ای" برای خطاهای کوچک به رفتاری مانند "ام آ ای" برای خطاهای بزرگ تغییر می‌کند. این امر باعث می‌شود که ضرر هوبر در مقایسه با "ام اس ای" نسبت به نقاط پرت شدید حساسیت کمتری داشته باشد، زیرا عبارت مربع در "ام اس ای" می‌تواند به طور قابل‌توجهی تأثیر اقلام پرت را بر ضرر کلی تقویت کند. به طور کلی تابع خطای هوبر نسبت به "اس ای" حساسیت کمتری به مقادیر پرت دارد. نقاط پرت، که نقاط داده ای هستند که به طور قابل توجهی از روند عمومی منحرف می شوند، می توانند تأثیر نامتناسبی بر عملکرد مدل در هنگام استفاده از "ام اس ای" به عنوان تابع ضرر داشته باشند. هوبر می‌تواند با اختصاص وزن‌های کمتر به خطاهای بزرگ، این مشکل را کاهش دهد و مدل را در برابر موارد پرت قوی‌تر کند..