Lasso/Ridge Regression

Presented by: The Neural Nets

Presentation Layout

Intro

Lasso vs Ridge

Advantages / Disadvantages

Inner Workings

Dataset examples

Intro

Lasso and Ridge regression are both used to prevent overfitting (regularization)

Hyperparameter determines size of penalty applied to coefficients

Performance often measured by Mean Squared Error or r2

Similar to Linear Regression, useful to predict ***quantities*** (price, MPG, etc)

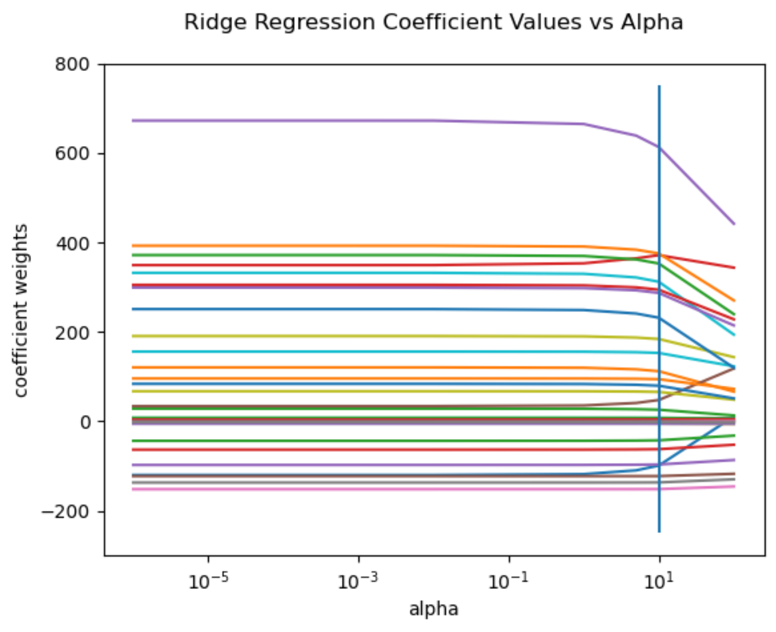
Advantages

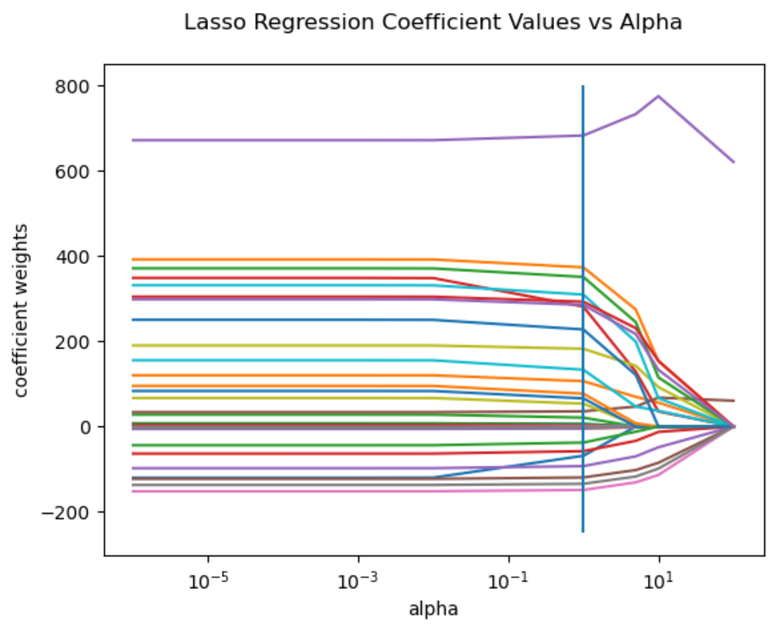
- Reduces the variance (high variance = overfitting)
- Lasso → Feature selection and dealing with outliers
- Ridge → better model performance since you're not losing any features

Disadvantages

- Certain alpha values can lead to bias (underfitting)
- Lasso → you have to be careful to not eliminate features (X-values)
- Ridge → Doesn't do well with outliers

Penalization of the Alpha Parameter





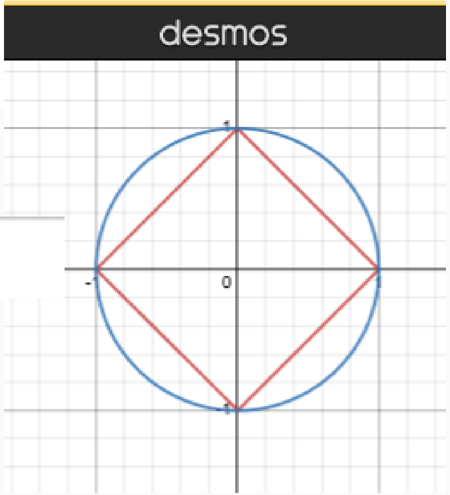
Lasso vs Ridge

Lasso = Linear regression with an L1 penalty

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

Loss function with L1 regularisation

“Diamond method” $|x| + |y| = 1$



Ridge = linear regression with an L2 penalty

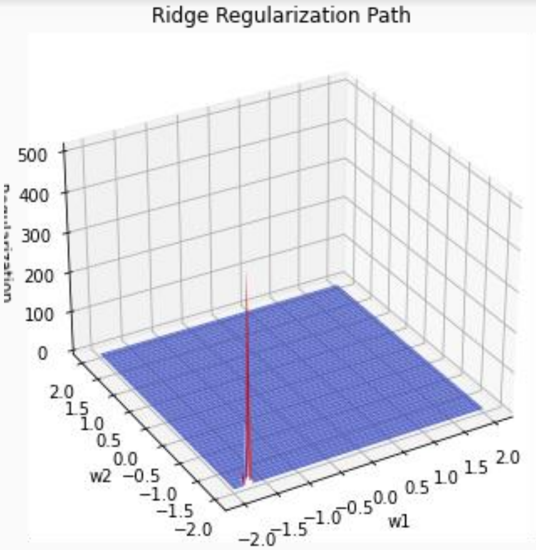
$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

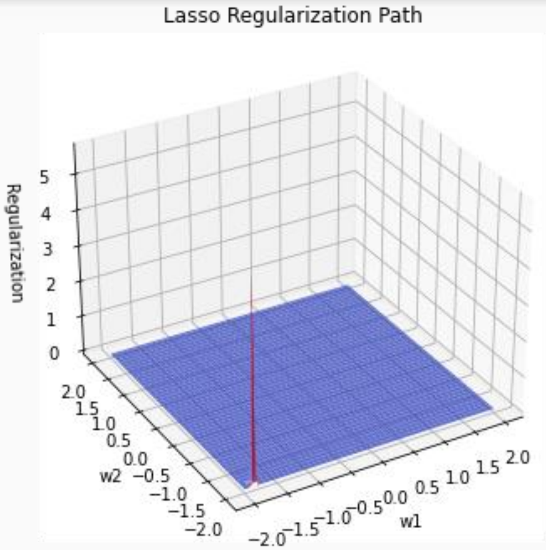
Loss function with L2 regularisation

“Circle Method” $x^2 + y^2 = 1$

L2 Regularization (Logistic Regression)

Lasso vs Ridge





https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261

$$Loss = Error(y, \hat{y})$$

Loss function with no regularisation

$$\hat{y} = w_1x_1 + w_2x_2 + ... + w_Nx_N + b$$

Inner Workings of Lasso/Ridge Regression

Data must be standardized

```
X = df_numeric.drop('Price', axis=1)
y = df_numeric['Price']

Xt = StandardScaler().fit_transform(X)
Xt = pd.DataFrame(Xt, columns=X.columns)

results = train_test_split(Xt, y, random_state=0, test_size=0.2)
X_train, X_test, y_train, y_test = results

Xt.describe().T.round(3)
```

✓ 0.1s

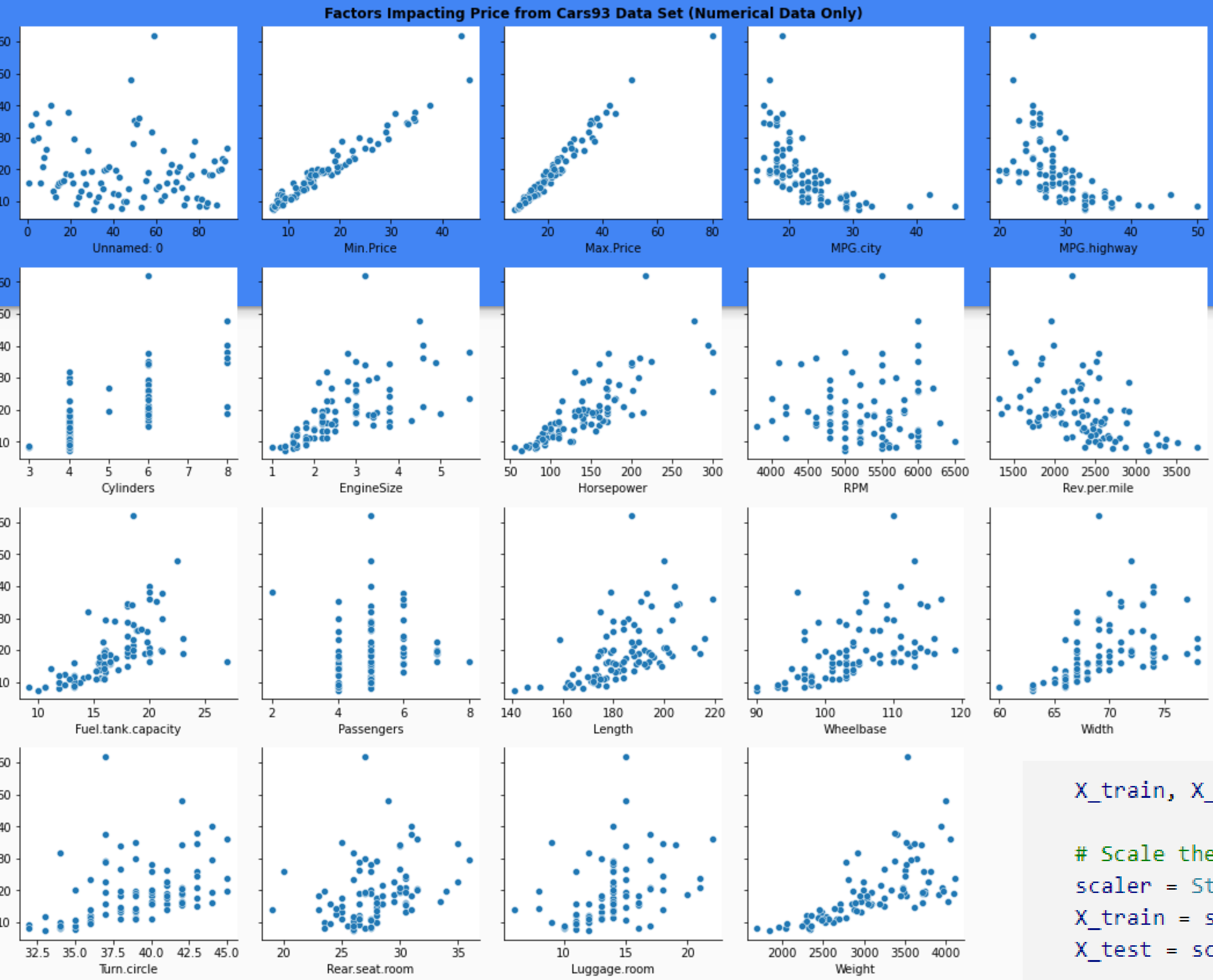
	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	93.0	0.0	1.005	-1.714	-0.857	0.000	0.857	1.714
Min.Price	93.0	0.0	1.005	-1.199	-0.727	-0.279	0.365	3.250
Max.Price	93.0	-0.0	1.005	-1.276	-0.656	-0.210	0.310	5.296
MPG.city	93.0	-0.0	1.005	-1.318	-0.781	-0.244	0.471	4.228
MPG.highway	93.0	0.0	1.005	-1.713	-0.582	-0.205	0.361	3.944
EngineSize	93.0	-0.0	1.005	-1.616	-0.841	-0.259	0.613	2.939
Horsepower	93.0	0.0	1.005	-1.705	-0.784	-0.073	0.502	2.998
RPM	93.0	-0.0	1.005	-2.495	-0.810	-0.136	0.791	2.054
Rev.per.mile	93.0	0.0	1.005	-2.050	-0.703	0.016	0.471	2.881
uel.tank.capacity	93.0	0.0	1.005	-2.289	-0.664	-0.081	0.655	3.169
Passengers	93.0	0.0	1.005	-2.986	-1.051	-0.083	0.884	2.820
Length	93.0	-0.0	1.005	-2.906	-0.634	-0.014	0.606	2.465
Wheelbase	93.0	-0.0	1.005	-2.056	-0.877	-0.140	0.893	2.219

Cross Validation is important in finding best alpha

```
parameters = {'alpha': [1e-12, 1e-9, 1e-6, 1e-3, 1, 10, 100, 1000, 10000, 1e6, 1e9]}


linear_regressor = LinearRegression()
linear_regressor.fit(X_train, y_train)
mse = cross_val_score(linear_regressor, X_train, y_train, scoring='neg_mean_squared_error', cv=5)
mean_mse = np.mean(mse)

lasso_regressor = GridSearchCV(lasso_model, parameters, scoring='neg_mean_squared_error', cv=5)
ridge_regressor = GridSearchCV(ridge_model, parameters, scoring='neg_mean_squared_error', cv=5)

lasso_regressor.fit(X_train, y_train);
ridge_regressor.fit(X_train, y_train);

print(f' Ridge best alpha: {ridge_regressor.best_params_}\n Ridge best score: {ridge_regressor.best_score_}\n')
print(f' Lasso best alpha: {lasso_regressor.best_params_}\n Lasso best score: {lasso_regressor.best_score_}\n')
print(f' Linear best score: {mean_mse}')
```

✓ 0.4s

```
Ridge best alpha: {'alpha': 1e-12}
Ridge best score: -0.0016084298156191334

Lasso best alpha: {'alpha': 0.001}
Lasso best score: -0.0011734736611818818

Linear best score: -0.0016084298156126369
```

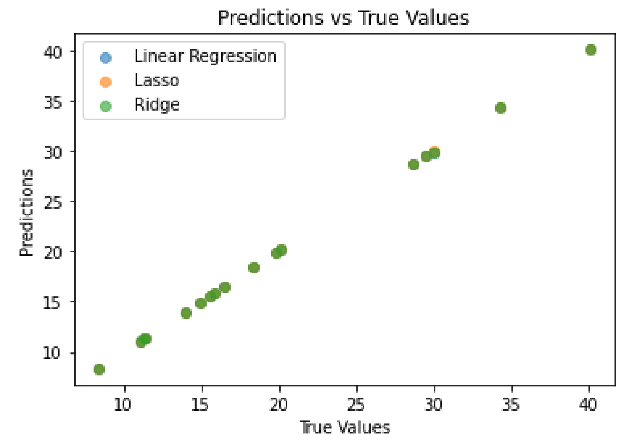


Cars Dataset

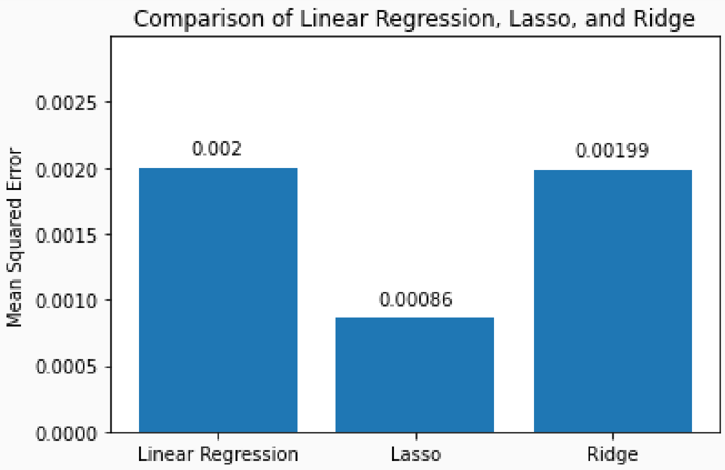
```
X_train, X_test, y_train, y_test = train_test_split

# Scale the input features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Cars Dataset

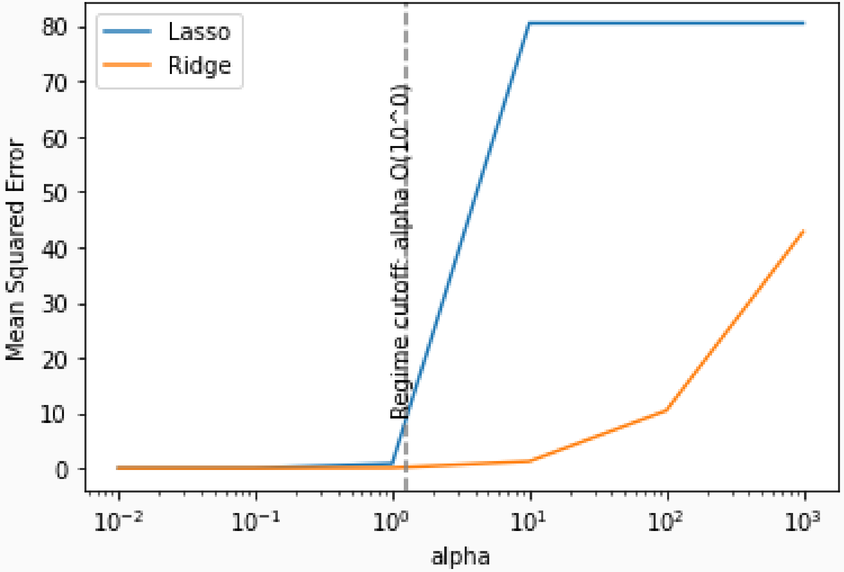
```
alphas = [0.01, 0.1, 1, 10, 100, 1000] # range of alpha values to try
lasso_scores = []
ridge_scores = []

for alpha in alphas:
    # fit Lasso model
    lasso = Lasso(alpha=alpha,)
    lasso.fit(X_train, y_train)
    # lasso_regressor.fit(X_train, y_train)
    lasso_scores.append(mean_squared_error(y_test, lasso.predict(X_test)))

    # fit Ridge model
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    # ridge_regressor.fit(X_train, y_train)
    ridge_scores.append(mean_squared_error(y_test, ridge.predict(X_test)))
```



Cars Dataset




```
Ridge best alpha: {'alpha': 0.001}
Ridge best score: -0.0010378010955886137
Lasso best alpha: {'alpha': 0.001}
Lasso best score: -0.0009668405665075328
Linear best score: -0.0010422280127113358, or score: 0.9999748590252451
```



Cars Dataset

Cars Dataset

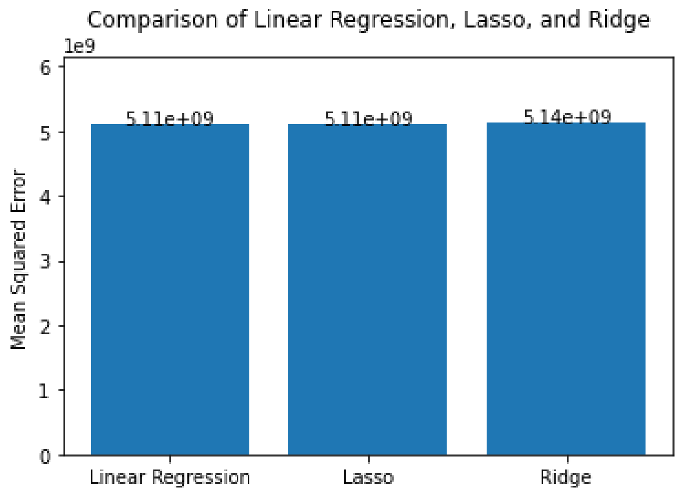
```python
# prepare for experiment, only numeric X's
num_cols1 = cars_new.columns[cars_new.dtypes != 'object'].dropna()
cars_new_num=cars_new[num_cols1].dropna()

# drop y
num_cols1 = num_cols1.drop('MSRP')

cars_new_num.head()
```

✓ 0.0s Python

	Year	Engine HP	Engine Cylinders	Number of Doors	highway MPG	city mpg	Popularity	MSRP
0	2011	335.0	6.0	2.0	26	19	3916	46135
1	2011	300.0	6.0	2.0	28	19	3916	40650
2	2011	300.0	6.0	2.0	28	20	3916	36350
3	2011	230.0	6.0	2.0	28	18	3916	29450
4	2011	230.0	6.0	2.0	28	18	3916	34500

Cars Dataset

```
Ridge best alpha: {'alpha': 10}
Ridge best score: -41.9
Lasso best alpha: {'alpha': 0.001}
Lasso best score: -43
Linear best score: -43, or score: -0.405
```




```python
# cars93 columns to compare
features = ['Cylinders', 'Horsepower', 'MPG.city', 'MPG.highway']

#new cars columns to compare
features1 = ['Engine Cylinders','Engine HP','city mpg','highway MPG'

# Select the features to train on from cars93 data
X_train1 = cars_num[features]
y_train1 = cars_num['Price']

#select test set from new cars dataset
X_test1 = cars_new_num[features1]
y_test1 = cars_new_num['MSRP']
```

```python
X_train1.shape, X_test1.shape, y_train1.shape,y_test1.shape
```

✓ 0.0s

```
((82, 4), (11815, 4), (82,), (11815,))
```

Diabetes Dataset

1. Preprocess → Done Already
2. Training the models:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.20)
```

```
### Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
### Ridge Regression
rr = Ridge(alpha=0.001)
rr.fit(X_train, y_train)
```

```
### Lasso Regression
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
```

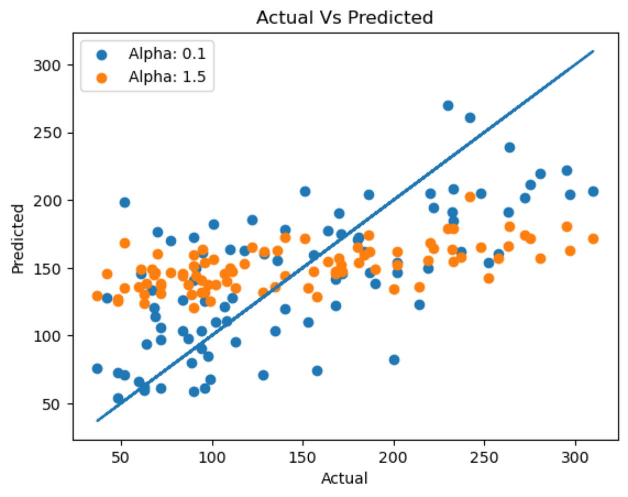
Diabetes Dataset Continued

3. Evaluation of the 3 models:

Model	$R^2 \rightarrow$ Training	$R^2 \rightarrow$ Testing	Mean Squared Error
Linear	0.5279198995709651	0.4526066021617378 7	2900.1732878832318
Ridge	0.5278462338370481	0.4534315003732732	2895.802851981438
Lasso	0.5169420144043178	0.4718552616908693 3	2798.190968742363

Diabetes Dataset Wrap Up

4. Visualizations



Diamonds Dataset - Intro Code

```
diamonds = pd.read_csv('../data/diamonds.csv')
diamonds.info()
```

✓ 0.1s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  53940 non-null  int64
 1   carat       53940 non-null  float64
 2   cut         53940 non-null  object
 3   color       53940 non-null  object
 4   clarity     53940 non-null  object
 5   depth       53940 non-null  float64
 6   table       53940 non-null  float64
 7   price       53940 non-null  int64
 8   x           53940 non-null  float64
 9   y           53940 non-null  float64
 10  z           53940 non-null  float64
dtypes: float64(6), int64(2), object(3)
```

```
# diamonds_expanded.info()

def append_random_data(dataframe):
    rows = dataframe.shape[0]
    cols = dataframe.shape[1] * 3
    col_list = ['s' + str(x) for x in range(1, cols + 1)]
    random_dataframe = pd.DataFrame(np.random.randint(0, 1000, size=(rows, cols)), columns=col_list)
    expanded_df = pd.concat([dataframe, random_dataframe], axis=1)
    return expanded_df
```

✓ 0.0s

```
expanded_df = append_random_data(diamonds)

X = expanded_df.drop(['Unnamed: 0', 'price'], axis=1)
y = expanded_df[['price']]
X = pd.get_dummies(X, drop_first=True)

Xt = StandardScaler().fit_transform(X)
Xt = pd.DataFrame(Xt, columns=X.columns)
```

Diamonds Dataset - Findings

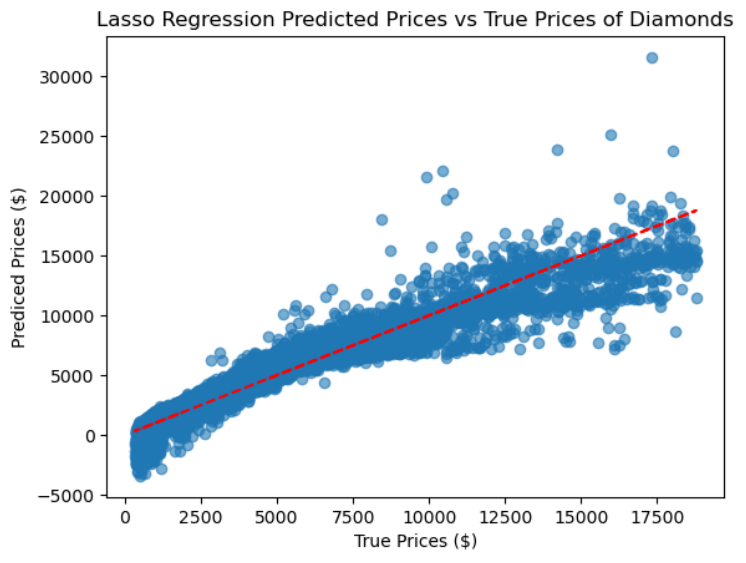
```
Lasso best alpha: {'alpha': 1}
Lasso best score (-mse): -1287387.434664847
Lasso mean error: 1134.6309684936539
------
Ridge best alpha: {'alpha': 10}
Ridge best score (-mse): -1288223.8251853979
Ridge mean error: 1134.9994824604098
------
Linear best score (-mse): -1288371.935763155
Linear mean error: 1135.0647275654173
```

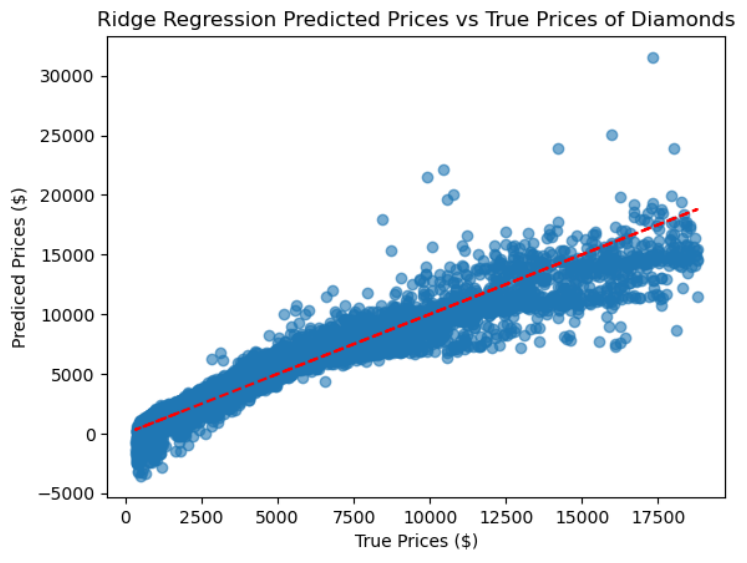
```
Lasso best alpha: {'alpha': 1}
Lasso best score (r2): 0.9191967785023828
 ------
Ridge best alpha: {'alpha': 10}
Ridge best score (r2): 0.9191438355439777
 ------
Linear r2: 0.9211350189097484
```

Decrease in # of coefficients due to regularization process

```
Coefficients > 0 in ridge regression: 56
Coefficients > 0 in lasso regression: 50
```

Diamonds Dataset - Prediction Plots





Resources

Articles

[L1 and L2 Regularization Methods]

[Regularization in Machine Learning]

[GridSearchCV documentation]

[Mean Squared Error or R-Squared?]

[Ridge Coeffs vs Regularization plot]

YouTube Videos

https://www.youtube.com/watch?v=uu2X47cSLmM

https://www.youtube.com/watch?v=0yI0-r3Ly40

Thanks!

Questions?

GitHub Link:
https://github.com/jhoffmanDEV10/LassoRidge

Citations

- https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229
- https://regenerativetoday.com/understanding-regularization-in-plain-language-l1-and-l2-regularization/
- https://www.analyticsvidhya.com/blog/2016/01/ridge-lasso-regression-python-complete-tutorial/
- https://www.section.io/engineering-education/regularization-to-prevent-overfitting/
- https://www.datacamp.com/tutorial/tutorial-lasso-ridge-regression
- https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261