Lasso and Ridge Regression

The Nappers
Group 5
Islam, Shervorn, Shirley

What is it?

- Lasso and Ridge are both Regression models but with a penalty (also called a regularization).
- They add a penalty to how big your coefficients can get, each in a different way.
- Lasso will shrink unimportant coefficients to zero and basically ‘drop’ features that don’t seem to have a high impact on the model
- Ridge will shrink coefficients to prevent any of them from being too big or extreme
- Regularization solves the problem of overfitting
- Lasso regularization is also called L1 penalty
- Ridge regularization is also called L2 penalty
- Both have a hyperparameter that controls how much you want to penalize the coefficients

Lasso Regularization

Advantages

- Automatic features selection
  - Decides which features are important
  - Sets the coefficients for features it does not consider interesting to zero
  - Reduces unwanted noise from useless features
  - Creates a simple model
- Reduced overfitting
  - Penalty helps prevent the model from overfitting
  - Decreases model complexity

Disadvantages

- Biased coefficients
  - Shrunken coefficients
  - Does not represent the true magnitude of the relationship between the features and the outcome
- Struggle with correlated features
  - A feature gets selected somewhat arbitrarily
  - All other features that are highly correlated with that feature are dropped from the model
  - May lead to the false conclusion that only the feature that was selected to remain in the model is important

Ridge Regularization

Advantages

- Handles correlated features
    - Too many correlated features may lead to overfitting
    - Reduces multicollinearity
- Reduced overfitting
    - Penalty shrinks some coefficients close to zero

Disadvantages

- Biased coefficients
- Not capable of performing feature selection

Some Similarities and Differences Between Ridge and Lasso Regressions:

Ridge Regression	**Lasso Regression**
Shrinks the coefficients toward zero	Shrinks and encourages some coefficients to be exactly zero
Does not eliminate any features	Can eliminate some features
Suitable when all features are important	Suitable when some features are irrelevant or redundant
More computationally efficient	Less computationally efficient
Requires setting a hyperparameter	Requires setting a hyperparameter
Performs better when there are many small to medium-sized coefficients	Performs better when there are a few large coefficients

Unique LASSO Regression Hyperparameters

precompute : ***‘auto’, bool or array-like of shape (n_features, n_features), default = ’auto’***

Allows you to use a precomputed Gram matrix to speed up calculations; The Gram matrix can also be passed as argument.

warm_start : ***bool, default = False***

When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.

selection : ***{‘cyclic’, ‘random’}, default = ’cyclic’***

If set to ‘random’, a random coefficient is updated every iteration rather than looping over features sequentially by default; this often leads to significantly faster convergence.

Unique Ridge Regression Hyperparameters

solver : *{‘auto’, ‘svd’, ‘cholesky’, ‘lsqr’, ‘sparse_cg’, ‘sag’, ‘saga’, ‘lbfgs’}, default = ’auto’*

Allows you to choose which solver to use during computational routines from the list above.

- auto: chooses the solver automatically based on the type of data
- svd: uses a Singular Value Decomposition of X to compute the Ridge coefficients
- cholesky: uses the standard scipy.linalg.solve function to obtain a closed-form solution
- lsqr: uses the dedicated regularized least-squares routine scipy.sparse.linalg.lsqr

Note: All solvers except ‘svd’ support both dense and sparse datasets where dense datasets are defined as datasets with filled with mostly non-zero values while sparse datasets are filled with zero values.

Unique Ridge Regression Hyperparameters Cont.

positive : *bool, default = False*

When set to True, forces the coefficients to be positive.

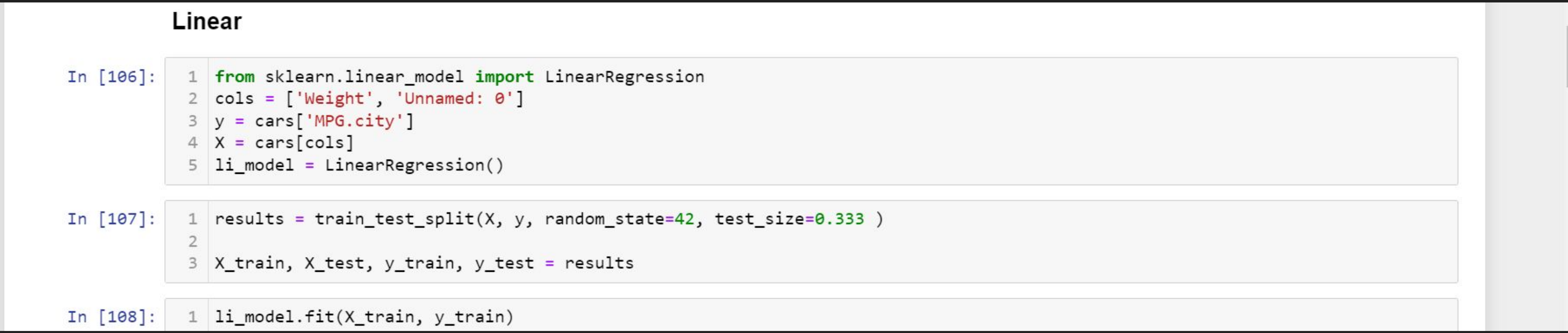
Note: The only solver that supports only positive coefficients is ‘lbfgs’.

fit_intercept : *bool, default = True*

Whether to fit the intercept for this model. If set to false, no intercept will be used in calculations (i.e. X and y are expected to be centered).

Note: Only ‘lsqr’, ‘sag’, ‘sparse_cg’, and ‘lbfgs’ support sparse input when fit_intercept is True.

Cars93 Dataset

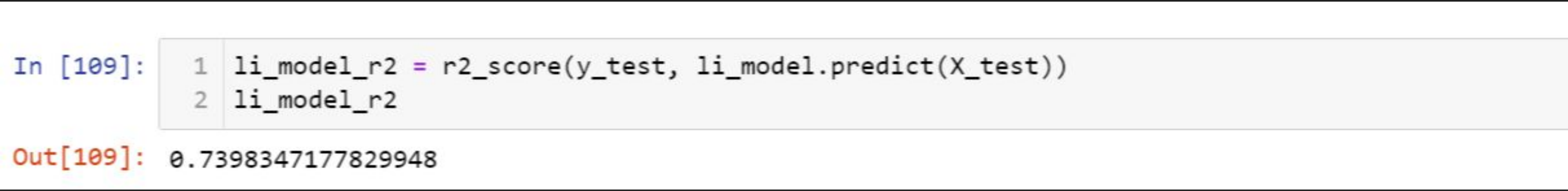


Linear

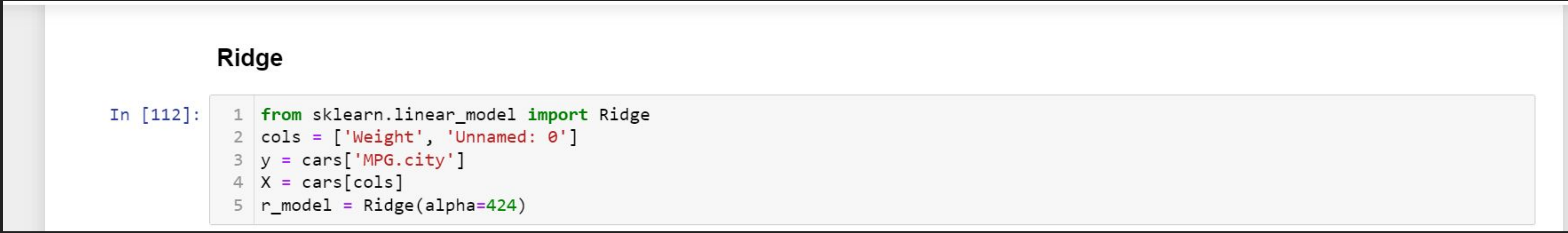
```
In [106]: from sklearn.linear_model import LinearRegression
          cols = ['Weight', 'Unnamed: 0']
          y = cars['MPG.city']
          X = cars[cols]
          li_model = LinearRegression()
```

```
In [107]: results = train_test_split(X, y, random_state=42, test_size=0.333 )

          X_train, X_test, y_train, y_test = results
```

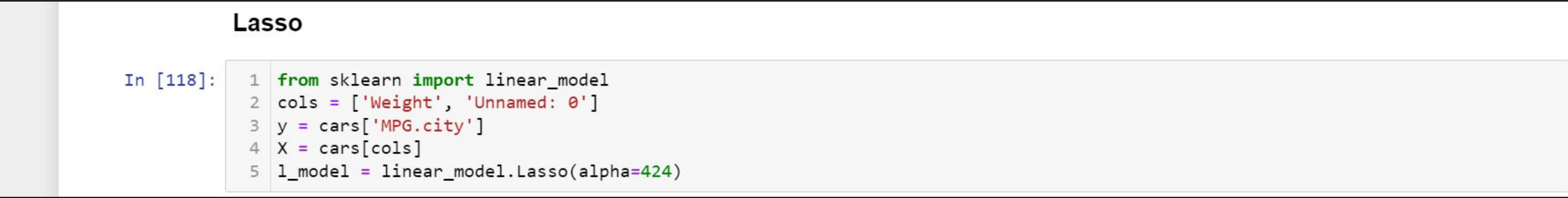
```
In [108]: li_model.fit(X_train, y_train)
```



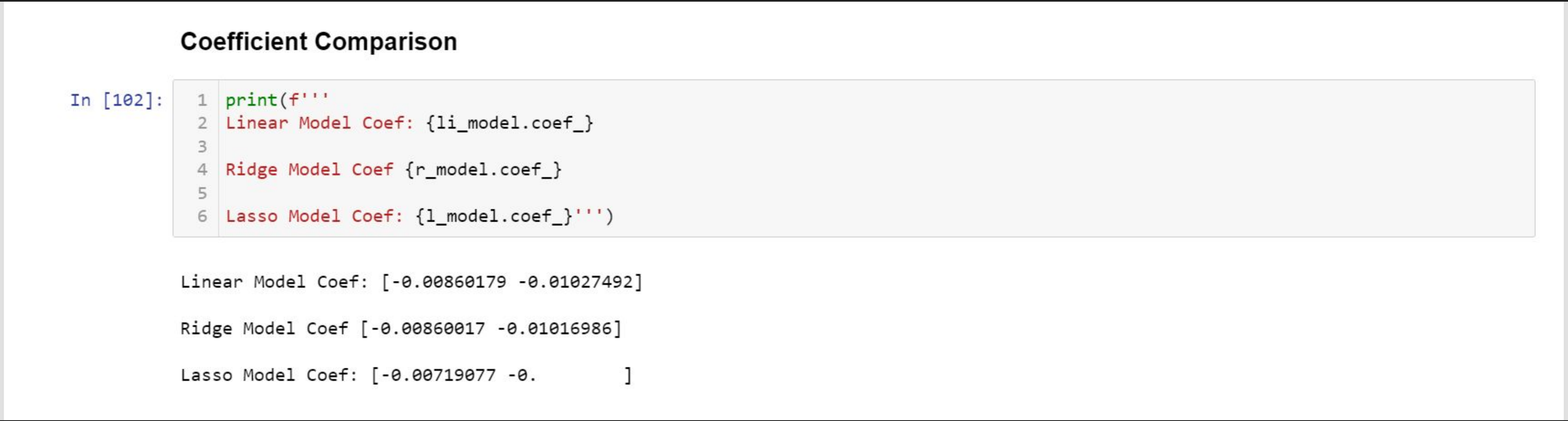
```
In [109]: li_model_r2 = r2_score(y_test, li_model.predict(X_test))
          li_model_r2

Out[109]: 0.739834717782948
```


Ridge

In [112]:
```python
from sklearn.linear_model import Ridge
cols = ['Weight', 'Unnamed: 0']
y = cars['MPG.city']
X = cars[cols]
r_model = Ridge(alpha=424)
```


Lasso

In [118]:
```python
from sklearn import linear_model
cols = ['Weight', 'Unnamed: 0']
y = cars['MPG.city']
X = cars[cols]
l_model = linear_model.Lasso(alpha=424)
```

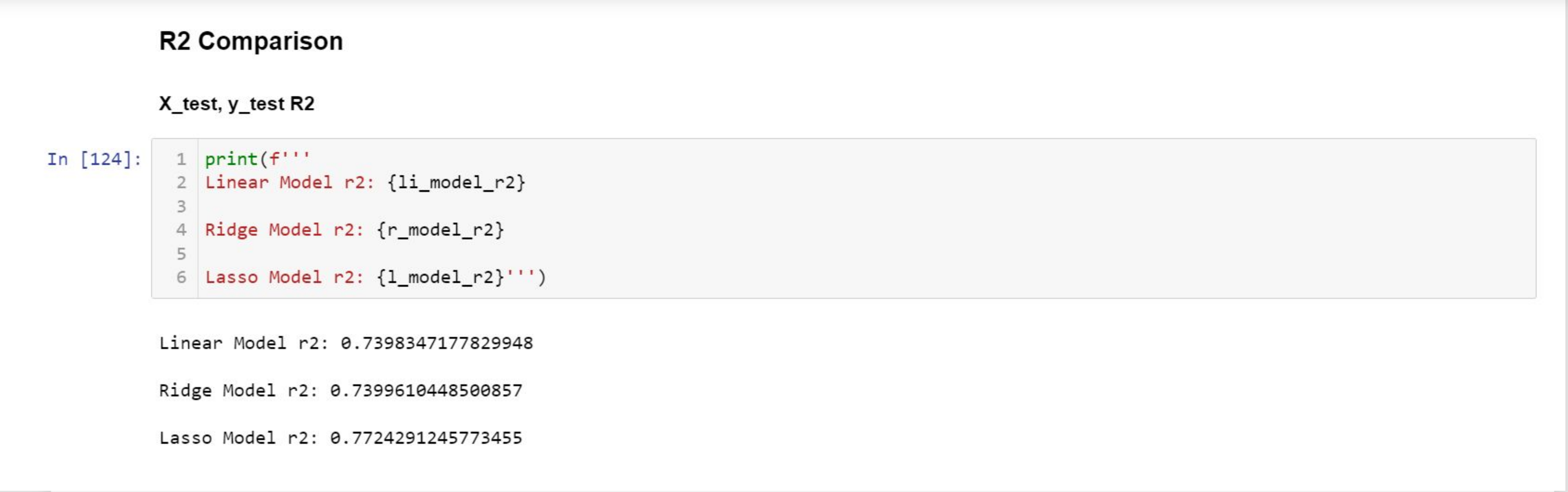


Coefficient Comparison

In [102]:

```python
print(f'''
Linear Model Coef: {li_model.coef_}

Ridge Model Coef {r_model.coef_}

Lasso Model Coef: {l_model.coef_}''')
```

```
Linear Model Coef: [-0.00860179 -0.01027492]

Ridge Model Coef [-0.00860017 -0.01016986]

Lasso Model Coef: [-0.00719077 -0.        ]
```

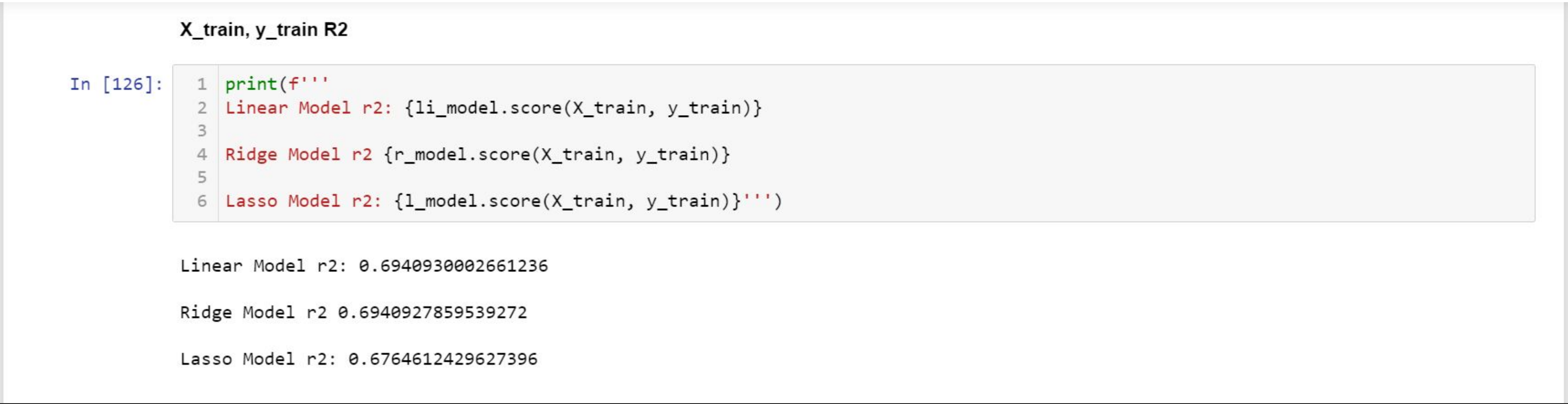


R2 Comparison

X_test, y_test R2

```
In [124]:
print(f'''
Linear Model r2: {li_model_r2}

Ridge Model r2: {r_model_r2}

Lasso Model r2: {l_model_r2}''')
```

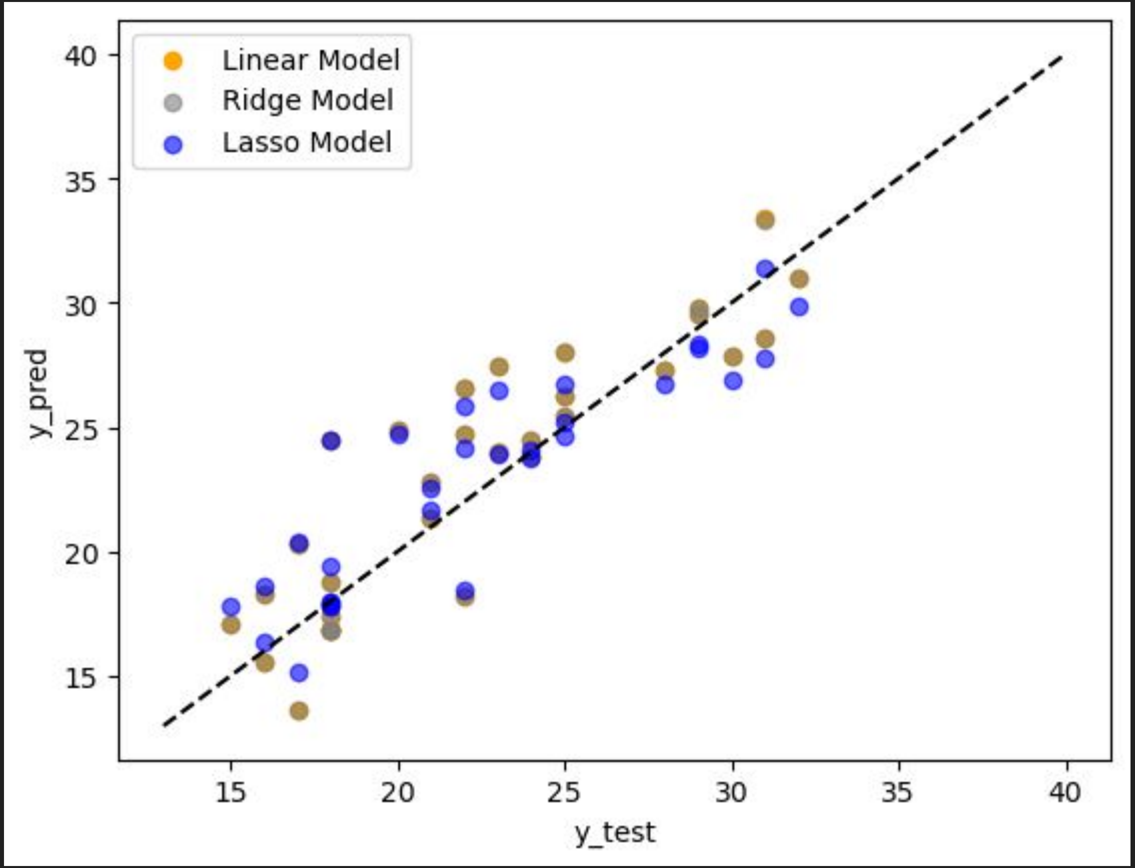
```
Linear Model r2: 0.7398347177829948

Ridge Model r2: 0.7399610448500857

Lasso Model r2: 0.7724291245773455
```



X_train, y_train R2

```
In [126]:
print(f'''
Linear Model r2: {li_model.score(X_train, y_train)}

Ridge Model r2 {r_model.score(X_train, y_train)}

Lasso Model r2: {l_model.score(X_train, y_train)}''')
```

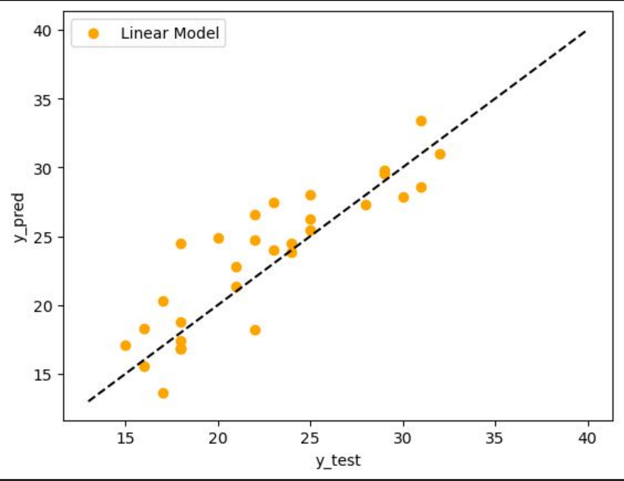
```
Linear Model r2: 0.6940930002661236

Ridge Model r2 0.6940927859539272

Lasso Model r2: 0.6764612429627396
```

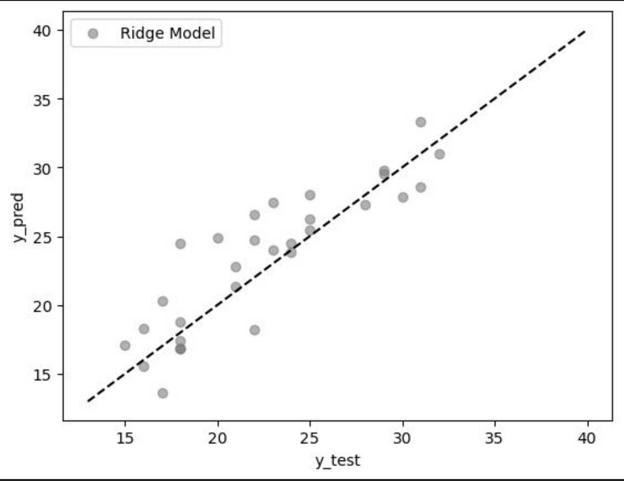

Linear Model
Ridge Model
Lasso Model
y_pred
y_test
40
35
30
25
20
15
15
20
25
30
35
40

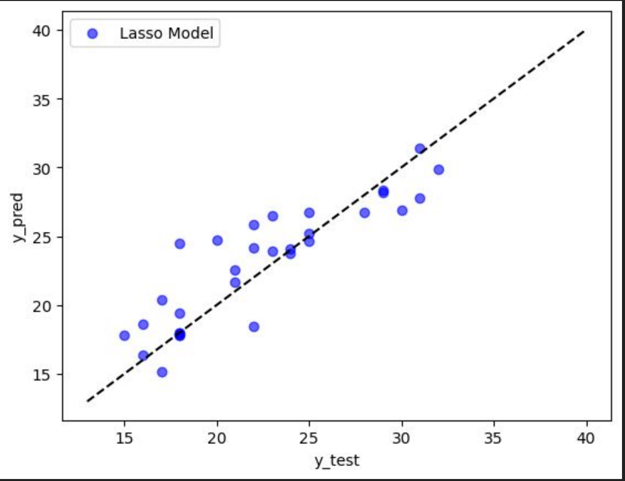
Linear = 5 data points

Ridge = 5 data points

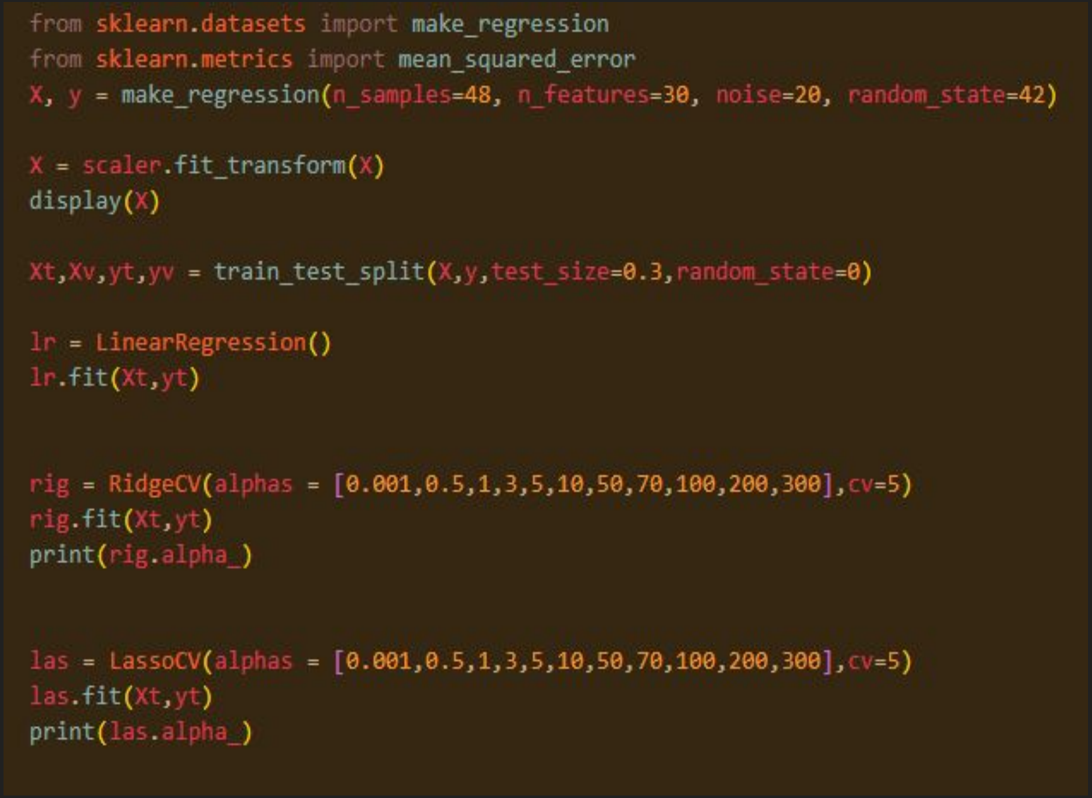
Lasso = 8 data points







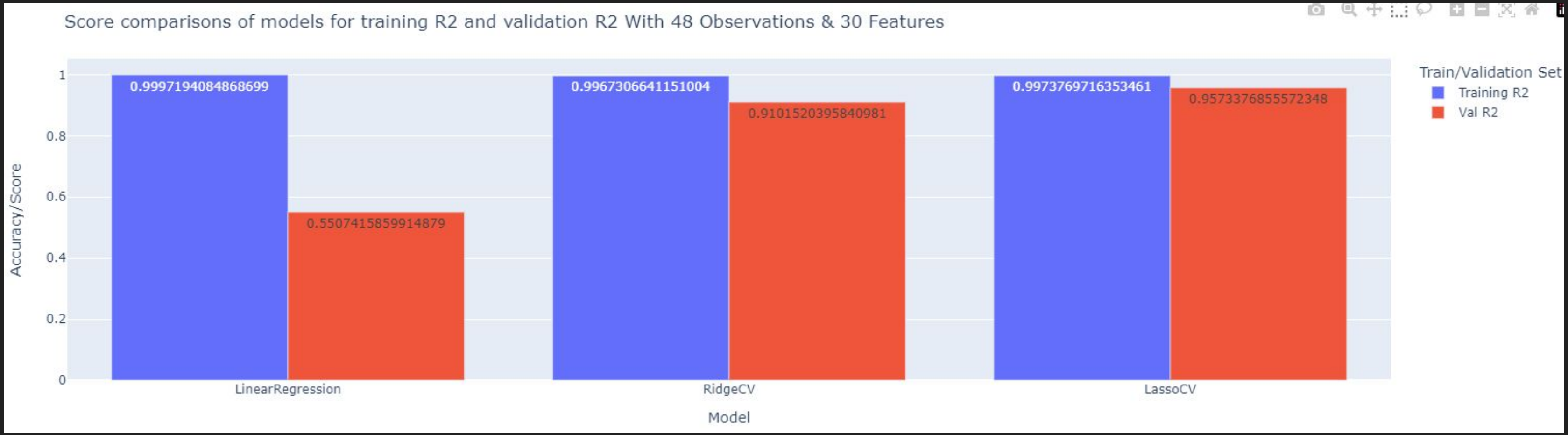
Make_regression() - Linear Regression Code

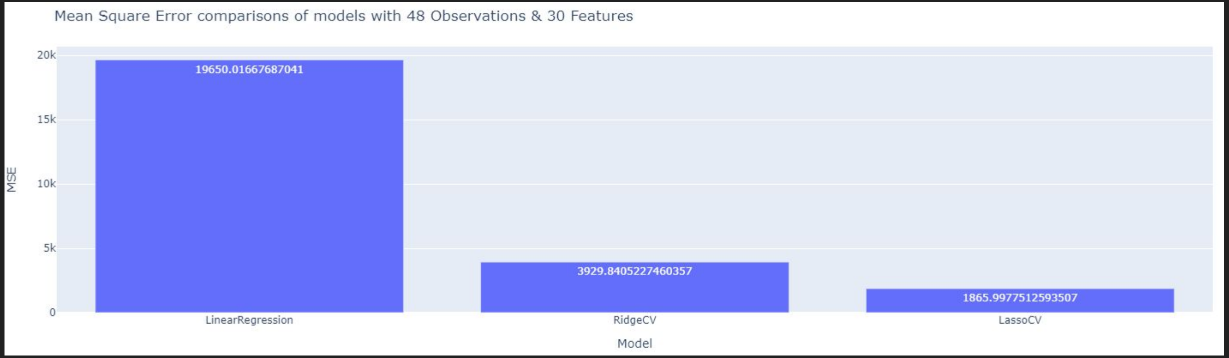


```
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error
X, y = make_regression(n_samples=48, n_features=30, noise=20, random_state=42)

X = scaler.fit_transform(X)
display(X)

Xt,Xv,yt,yv = train_test_split(X,y,test_size=0.3,random_state=0)

lr = LinearRegression()
lr.fit(Xt,yt)


rig = RidgeCV(alphas = [0.001,0.5,1,3,5,10,50,70,100,200,300],cv=5)
rig.fit(Xt,yt)
print(rig.alpha_)


las = LassoCV(alphas = [0.001,0.5,1,3,5,10,50,70,100,200,300],cv=5)
las.fit(Xt,yt)
print(las.alpha_)
```

- Make_regression - Generates a random regression problem
- Noise - The standard deviation of the gaussian noise applied to the output. - Meaningless data to try to make model worse
- N_samples - The number of samples.
- N_features - The number of features.
- RidgeCV and LassoCV are used in linear regression where you can pass in many alpha values and it will run the fit on all of them and give you back/ use the best one it found from the list.
- Alpha hyperparameter in linear regression penalizes the weights/coefficients more when it's larger and less when it's smaller
- Cv = cross validation strategy & pass in number of folds to use
- Use 48 observations and 30 features with 20 noise

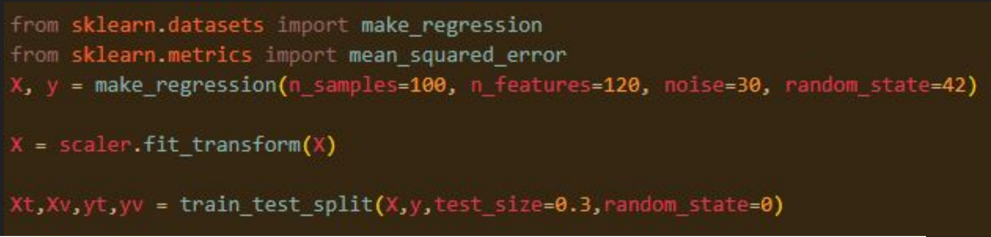
Make_regression() - Training and Validation Scores between models on 30 features, 48 observations



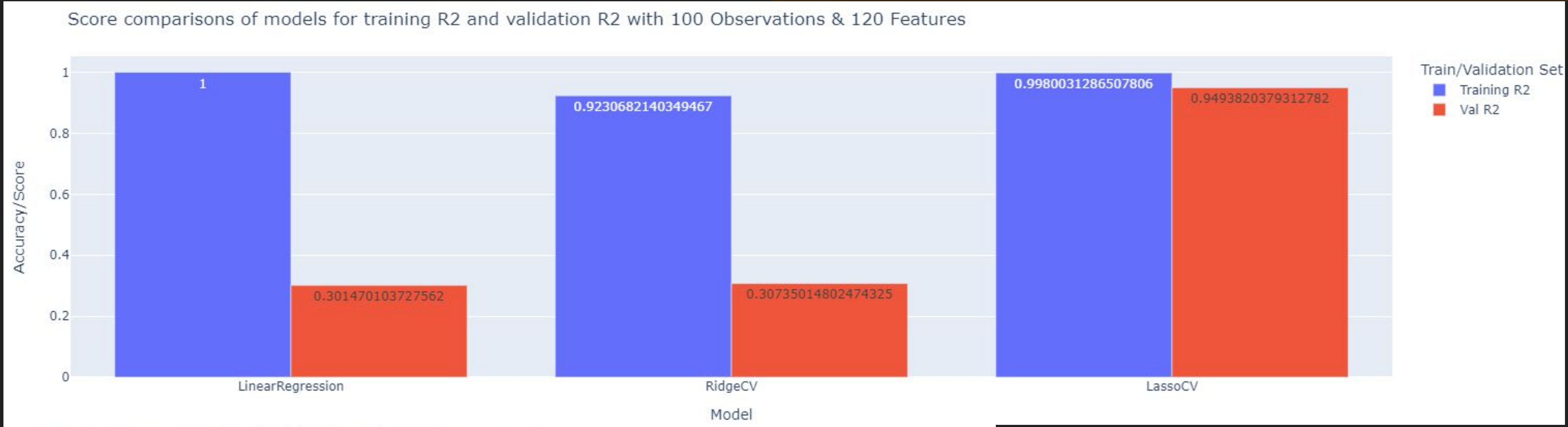


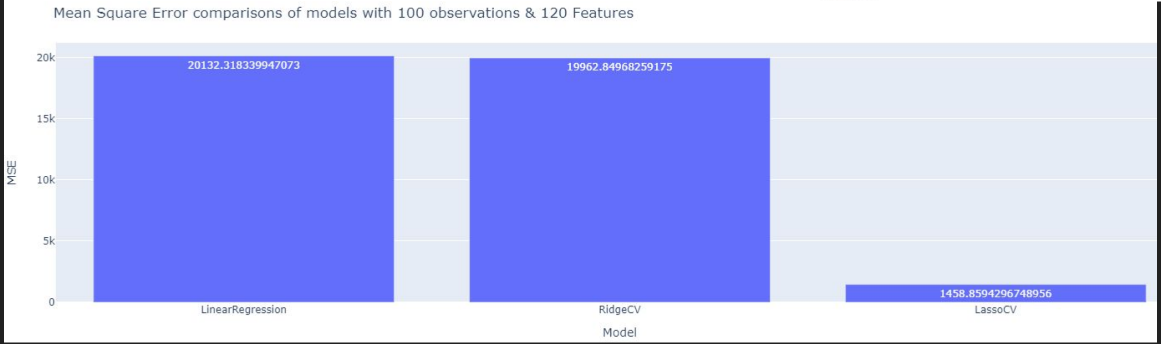
- Ridge and Lasso did better on both training and validation
- Linear Regression without regularization was clearly overfitted

Make_regression() - Training and Validation between models on 120 features, 100 observations



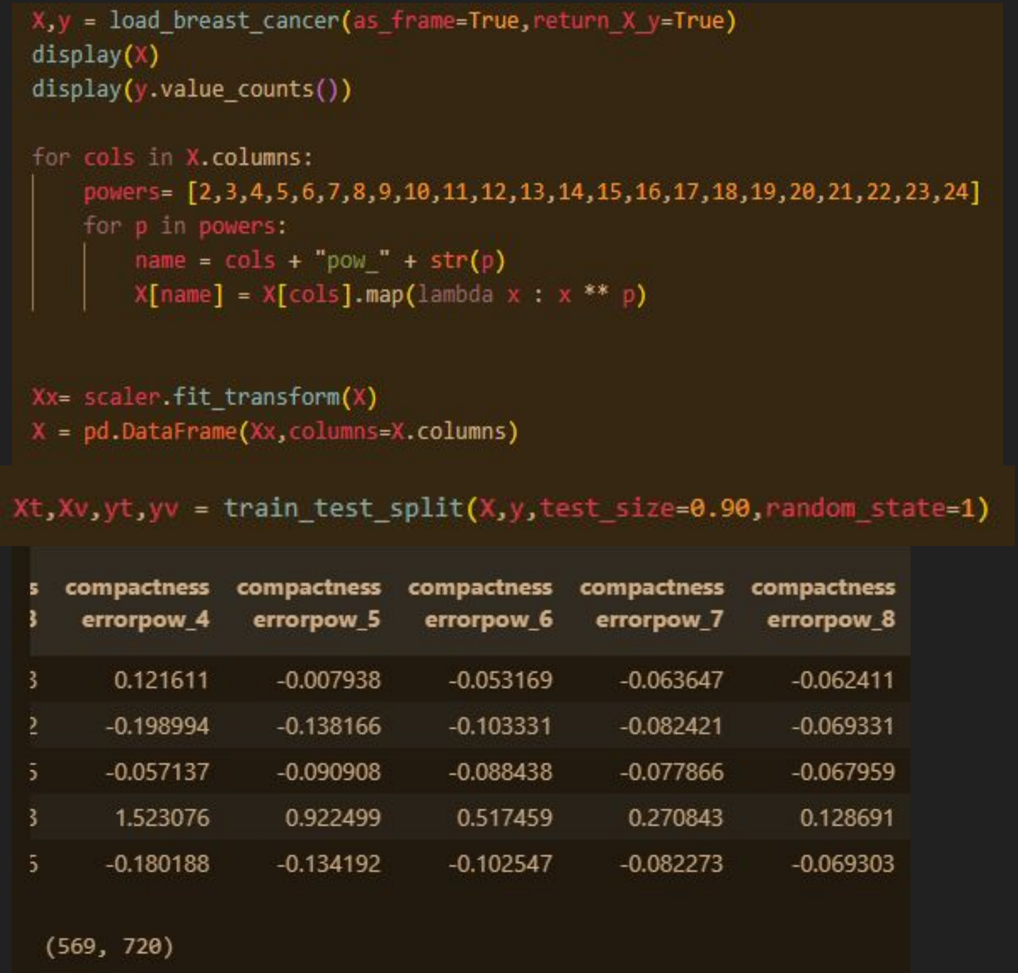
```
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error
X, y = make_regression(n_samples=100, n_features=120, noise=30, random_state=42)

X = scaler.fit_transform(X)

Xt,Xv,yt,yv = train_test_split(X,y,test_size=0.3,random_state=0)
```





- Ridge does poorly this time because there are too many features
- Lasso does better because it drops most of these features

Sklearn Breast Cancer Dataset - Logistic Regression - Multicollinearity



```
X,y = load_breast_cancer(as_frame=True,return_X_y=True)
display(X)
display(y.value_counts())

for cols in X.columns:
    powers= [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24]
    for p in powers:
        name = cols + "pow_" + str(p)
        X[name] = X[cols].map(lambda x : x ** p)


Xx= scaler.fit_transform(X)
X = pd.DataFrame(Xx,columns=X.columns)
```

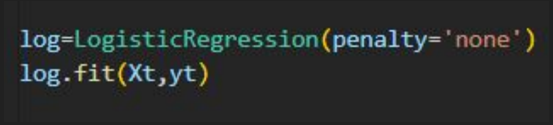
```
Xt,Xv,yt,yv = train_test_split(X,y,test_size=0.90,random_state=1)
```

s 3	compactness errorpow_4	compactness errorpow_5	compactness errorpow_6	compactness errorpow_7	compactness errorpow_8
3	0.121611	-0.007938	-0.053169	-0.063647	-0.062411
2	-0.198994	-0.138166	-0.103331	-0.082421	-0.069331
5	-0.057137	-0.090908	-0.088438	-0.077866	-0.067959
3	1.523076	0.922499	0.517459	0.270843	0.128691
5	-0.180188	-0.134192	-0.102547	-0.082273	-0.069303

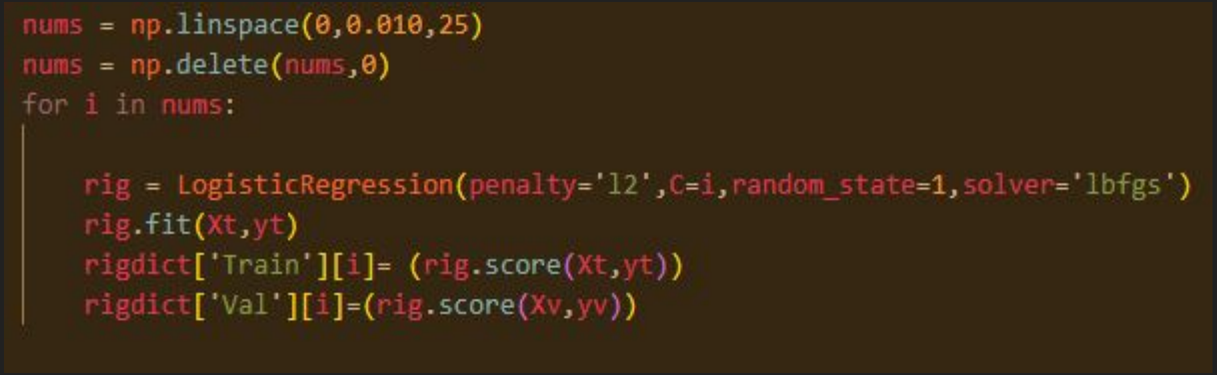
```
(569, 720)
```

- All numerical data
- Added power transformations to add the effect of multicollinearity
- Multicollinearity occurs when two or more independent variables have a high correlation with one another in a regression model
- With transforms there are 720 features and 569 observations
- Scaled the X data
- Split with a training data as 0.10 and test data as 0.90 to try to overfit the data to the model

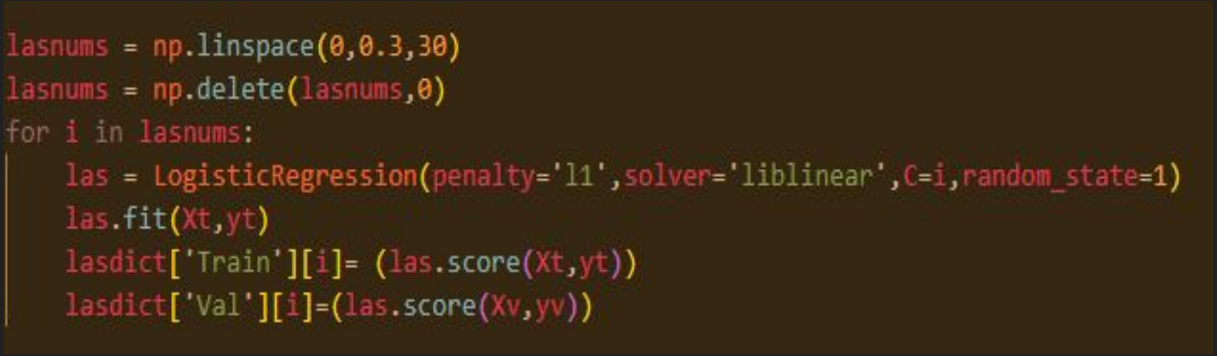
Sklearn Breast Cancer - Logistic Regression Code



```
log=LogisticRegression(penalty='none')
log.fit(Xt,yt)
```

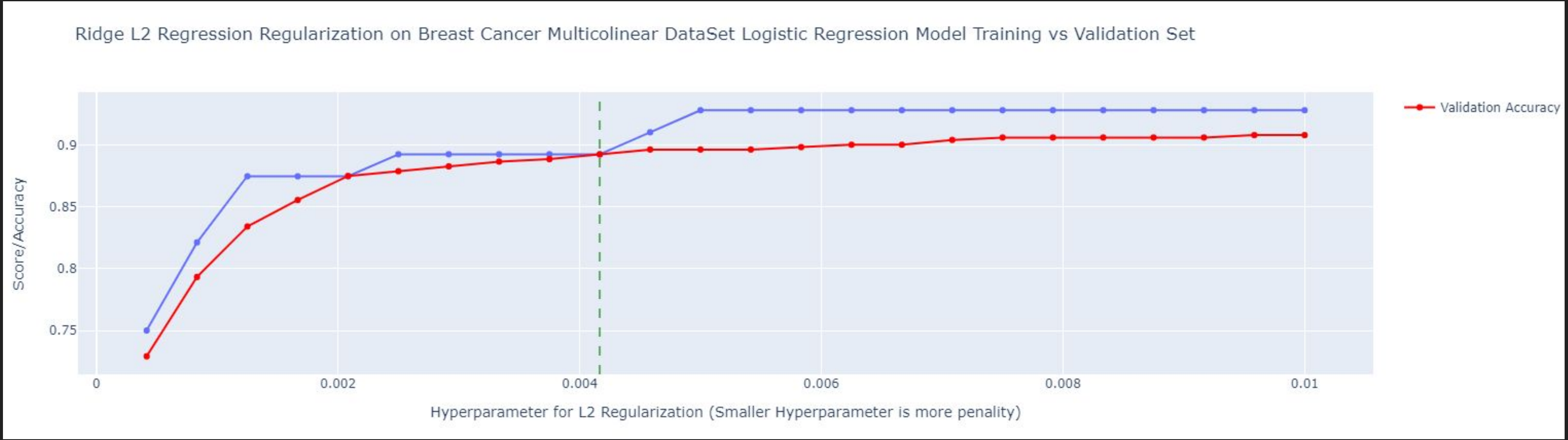


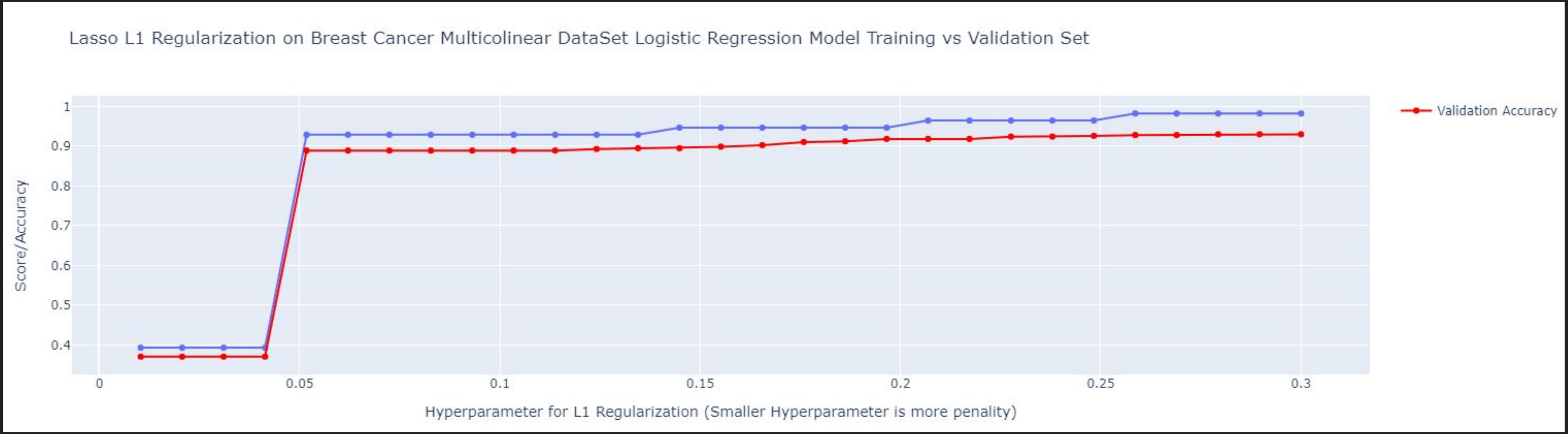
```
nums = np.linspace(0,0.010,25)
nums = np.delete(nums,0)
for i in nums:

    rig = LogisticRegression(penalty='l2',C=i,random_state=1,solver='lbfgs')
    rig.fit(Xt,yt)
    rigdict['Train'][i]= (rig.score(Xt,yt))
    rigdict['Val'][i]=(rig.score(Xv,yv))
```



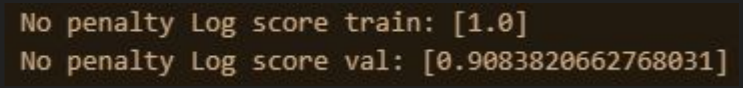
```
lasnums = np.linspace(0,0.3,30)
lasnums = np.delete(lasnums,0)
for i in lasnums:
    las = LogisticRegression(penalty='l1',solver='liblinear',C=i,random_state=1)
    las.fit(Xt,yt)
    lasdict['Train'][i]= (las.score(Xt,yt))
    lasdict['Val'][i]=(las.score(Xv,yv))
```

- Hyperparameters: C with penalty
- L2 is ridge penalty
- L1 is lasso penalty
- In logistic regression, the hyperparameter ‘C’ penalizes more with a smaller value and less with a larger value

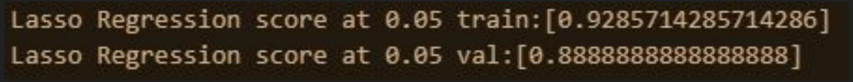

Ridge L2 Regression Regularization on Breast Cancer Multicolinear DataSet Logistic Regression Model Training vs Validation Set
Validation Accuracy
Score/Accuracy
0.9
0.85
0.8
0.75
0
0.002
0.004
0.006
0.008
0.01
Hyperparameter for L2 Regularization (Smaller Hyperparameter is more penality)


Lasso L1 Regularization on Breast Cancer Multicolinear DataSet Logistic Regression Model Training vs Validation Set
Validation Accuracy
Score/Accuracy
1
0.9
0.8
0.7
0.6
0.5
0.4
0
0.05
0.1
0.15
0.2
0.25
0.3
Hyperparameter for L1 Regularization (Smaller Hyperparameter is more penality)

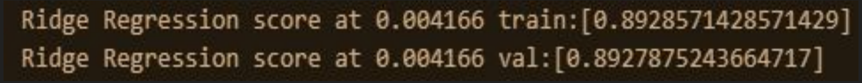
Sklearn Breast Cancer - Confusion Matrices and scores



```
No penalty Log score train: [1.0]
No penalty Log score val: [0.9083820662768031]
```

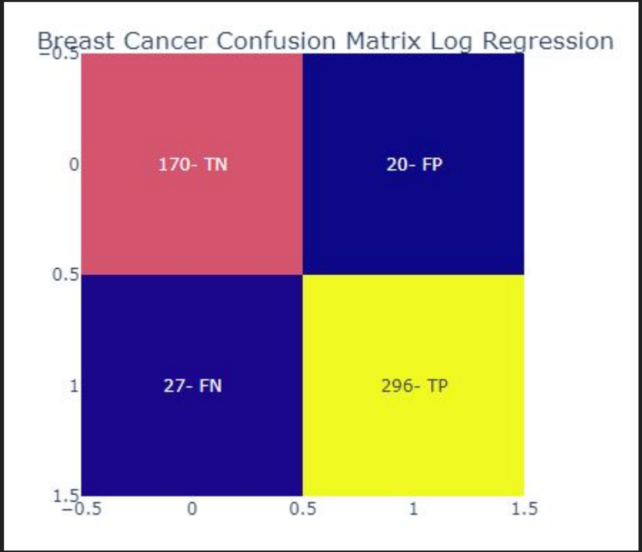


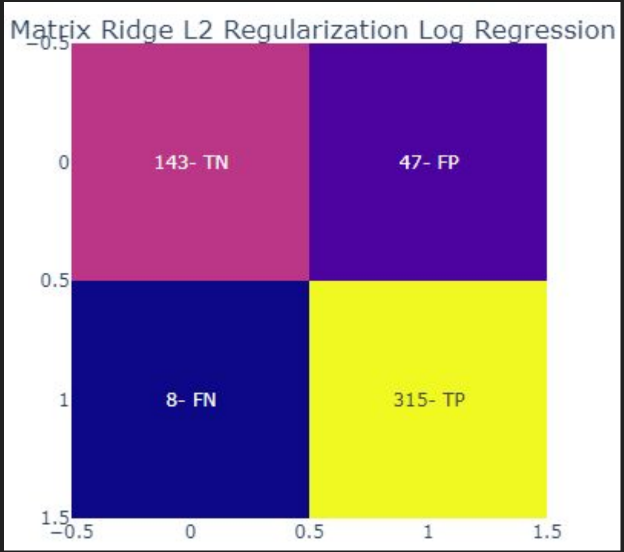
```
Lasso Regression score at 0.05 train:[0.9285714285714286]
Lasso Regression score at 0.05 val:[0.8888888888888888]
```

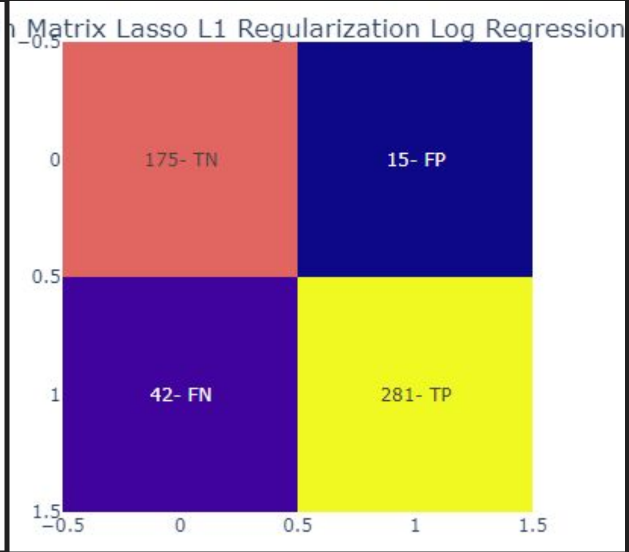


```
Ridge Regression score at 0.004166 train:[0.8928571428571429]
Ridge Regression score at 0.004166 val:[0.8927875243664717]
```

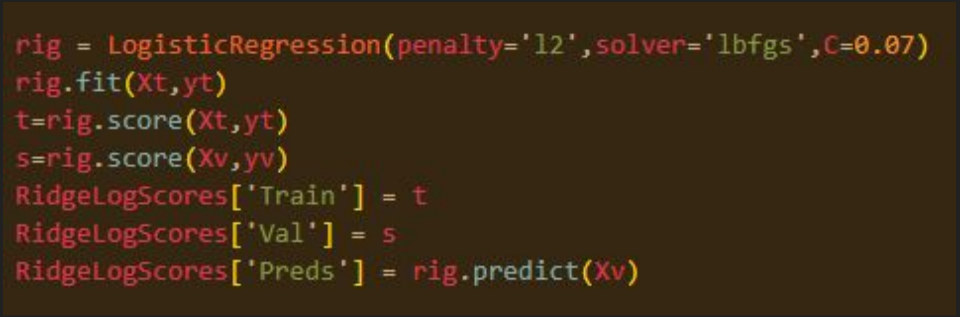
- Lasso and Ridge scores show less overfitting compared to no penalty logistic regression



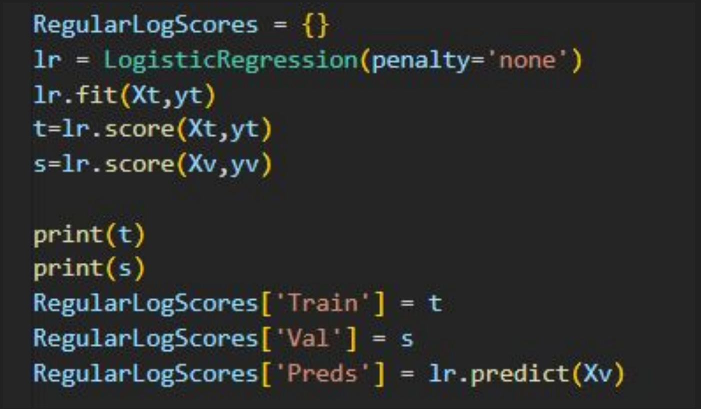




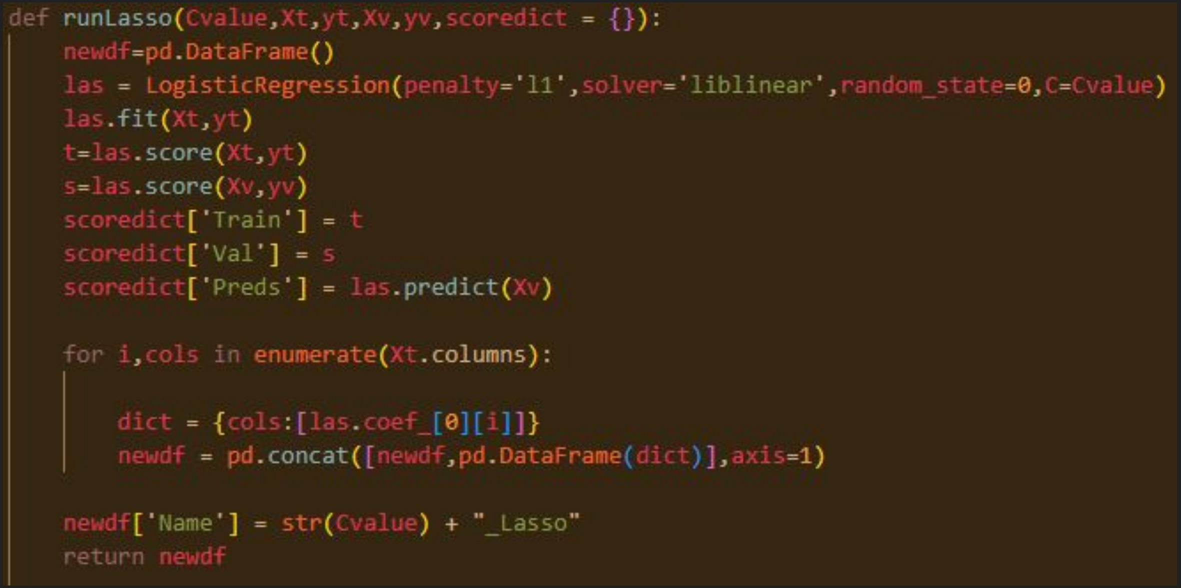
Titanic Dataset - Logistic Regression Code



```
rig = LogisticRegression(penalty='l2',solver='lbfgs',C=0.07)
rig.fit(Xt,yt)
t=rig.score(Xt,yt)
s=rig.score(Xv,yv)
RidgeLogScores['Train'] = t
RidgeLogScores['Val'] = s
RidgeLogScores['Preds'] = rig.predict(Xv)
```



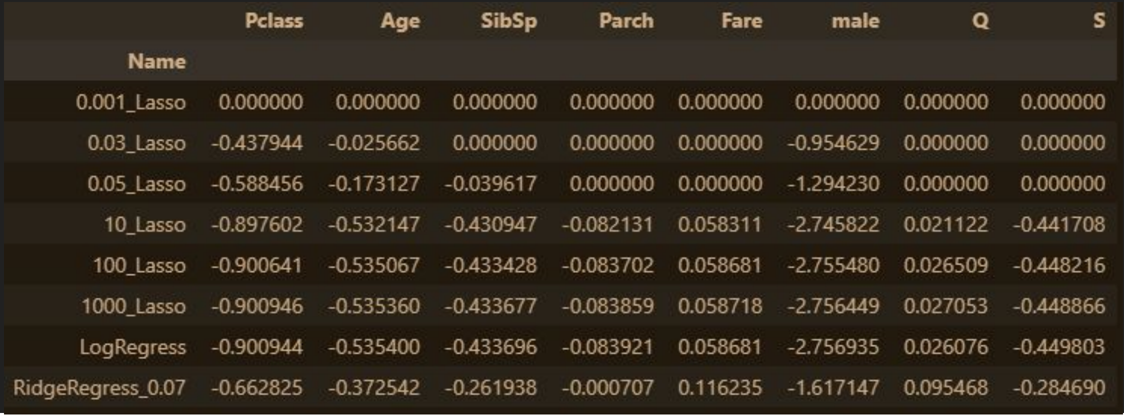
```
RegularLogScores = {}
lr = LogisticRegression(penalty='none')
lr.fit(Xt,yt)
t=lr.score(Xt,yt)
s=lr.score(Xv,yv)

print(t)
print(s)
RegularLogScores['Train'] = t
RegularLogScores['Val'] = s
RegularLogScores['Preds'] = lr.predict(Xv)
```



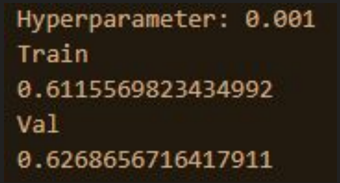
```
def runLasso(Cvalue,Xt,yt,Xv,yv,scoredict = {}):
    newdf=pd.DataFrame()
    las = LogisticRegression(penalty='l1',solver='liblinear',random_state=0,C=Cvalue)
    las.fit(Xt,yt)
    t=las.score(Xt,yt)
    s=las.score(Xv,yv)
    scoredict['Train'] = t
    scoredict['Val'] = s
    scoredict['Preds'] = las.predict(Xv)

    for i,cols in enumerate(Xt.columns):

        dict = {cols:[las.coef_[0][i]]}
        newdf = pd.concat([newdf,pd.DataFrame(dict)],axis=1)

    newdf['Name'] = str(Cvalue) + "_Lasso"
    return newdf
```

- 11 Features
- 891 Observations
- Imputed Numerical Nulls with mean, and Categorical Nulls with most_frequent
- Dummy vars for sex:male
- Dummy vars for Embarked: Q,S

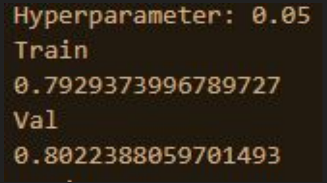
Titanic - Lasso vs Ridge for feature selection



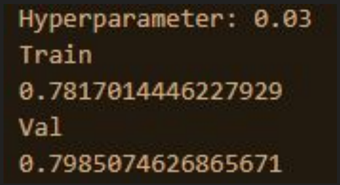
Name	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0.001_Lasso	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.03_Lasso	-0.437944	-0.025662	0.000000	0.000000	0.000000	-0.954629	0.000000	0.000000
0.05_Lasso	-0.588456	-0.173127	-0.039617	0.000000	0.000000	-1.294230	0.000000	0.000000
10_Lasso	-0.897602	-0.532147	-0.430947	-0.082131	0.058311	-2.745822	0.021122	-0.441708
100_Lasso	-0.900641	-0.535067	-0.433428	-0.083702	0.058681	-2.755480	0.026509	-0.448216
1000_Lasso	-0.900946	-0.535360	-0.433677	-0.083859	0.058718	-2.756449	0.027053	-0.448866
LogRegress	-0.900944	-0.535400	-0.433696	-0.083921	0.058681	-2.756935	0.026076	-0.449803
RidgeRegress_0.07	-0.662825	-0.372542	-0.261938	-0.000707	0.116235	-1.617147	0.095468	-0.284690



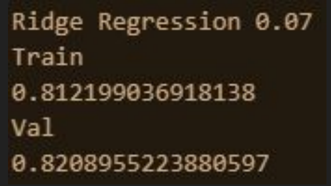
```
Hyperparameter: 0.001
Train
0.6115569823434992
Val
0.6268656716417911
```



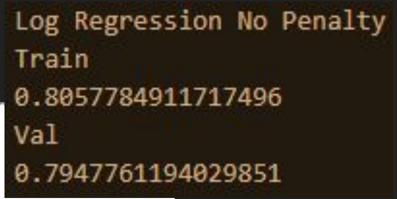
```
Hyperparameter: 0.05
Train
0.7929373996789727
Val
0.8022388059701493
```



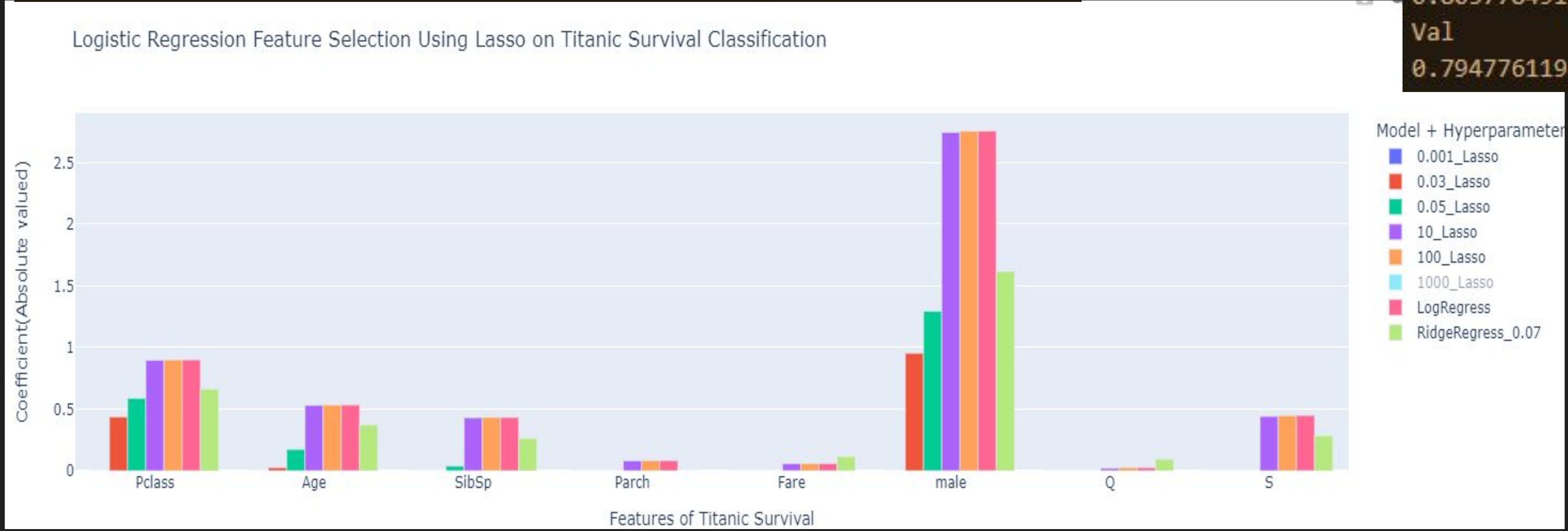
```
Hyperparameter: 0.03
Train
0.7817014446227929
Val
0.7985074626865671
```



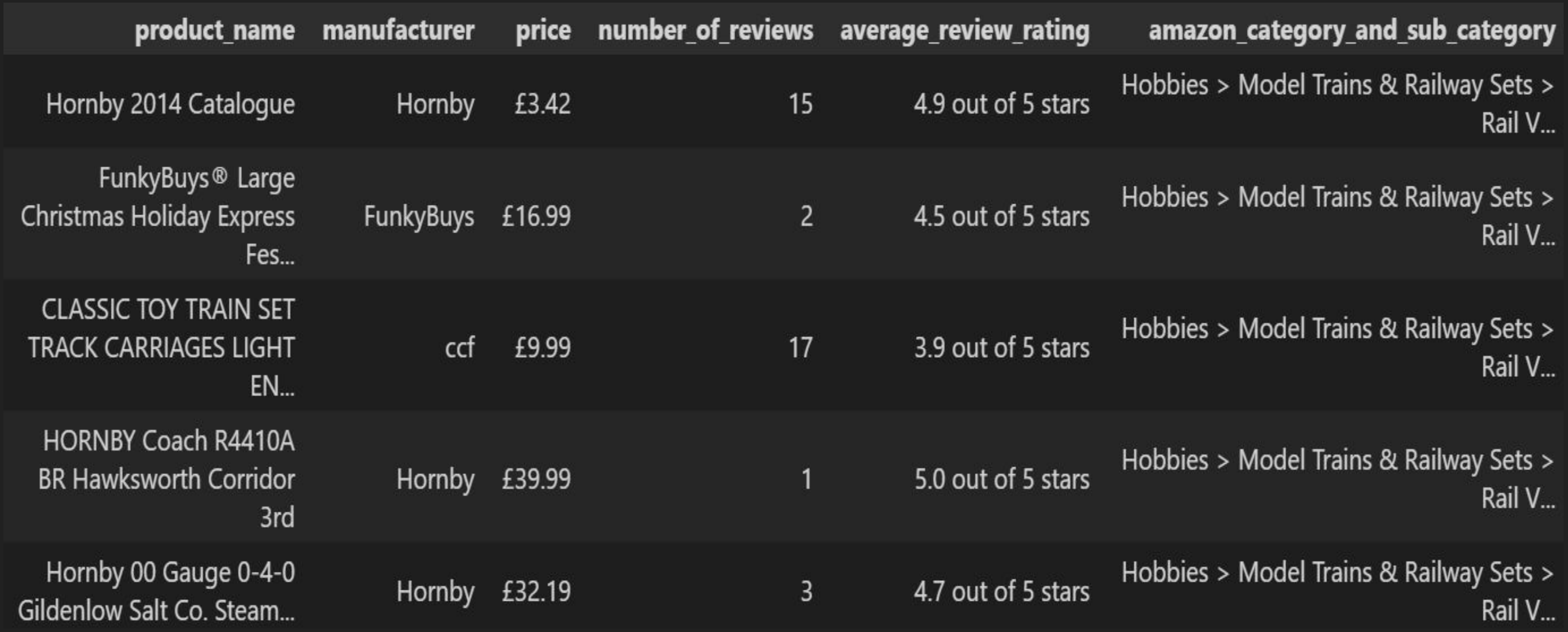
```
Ridge Regression 0.07
Train
0.812199036918138
Val
0.8208955223880597
```



```
Log Regression No Penalty
Train
0.8057784911717496
Val
0.7947761194029851
```

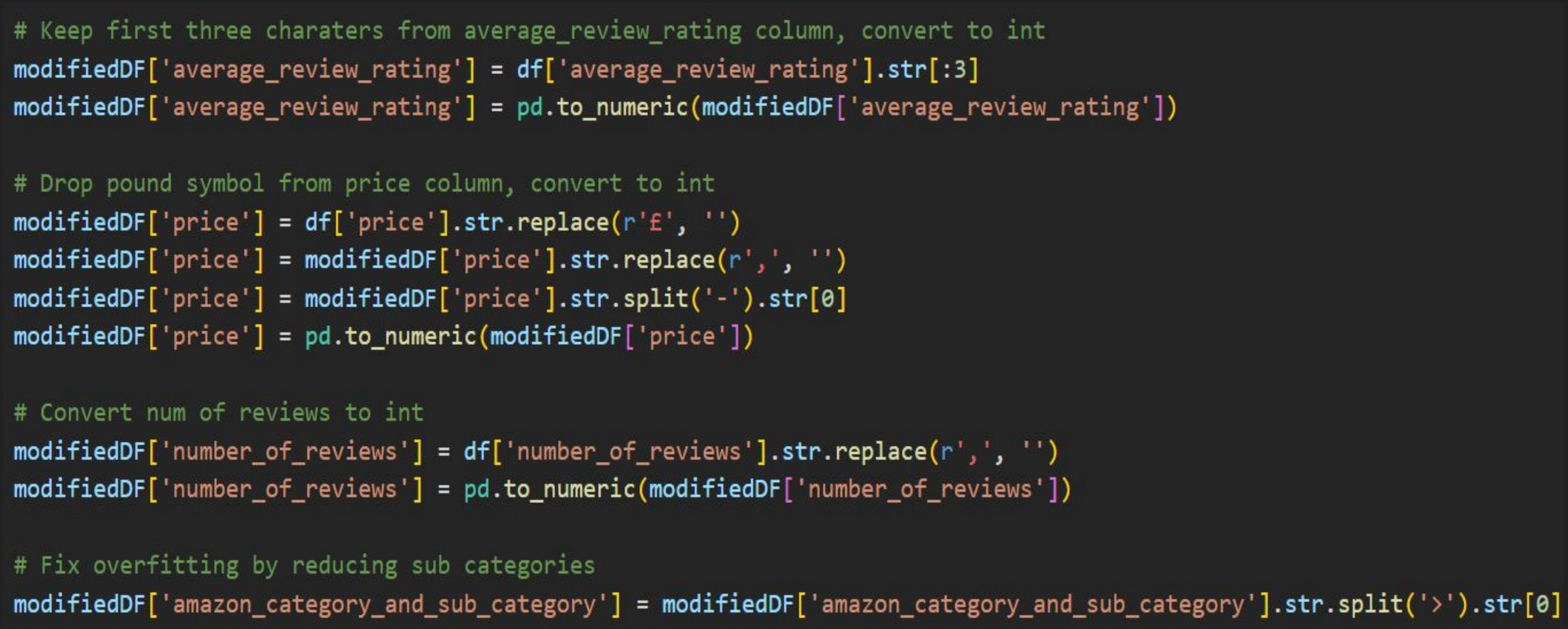


Toy Review Dataset



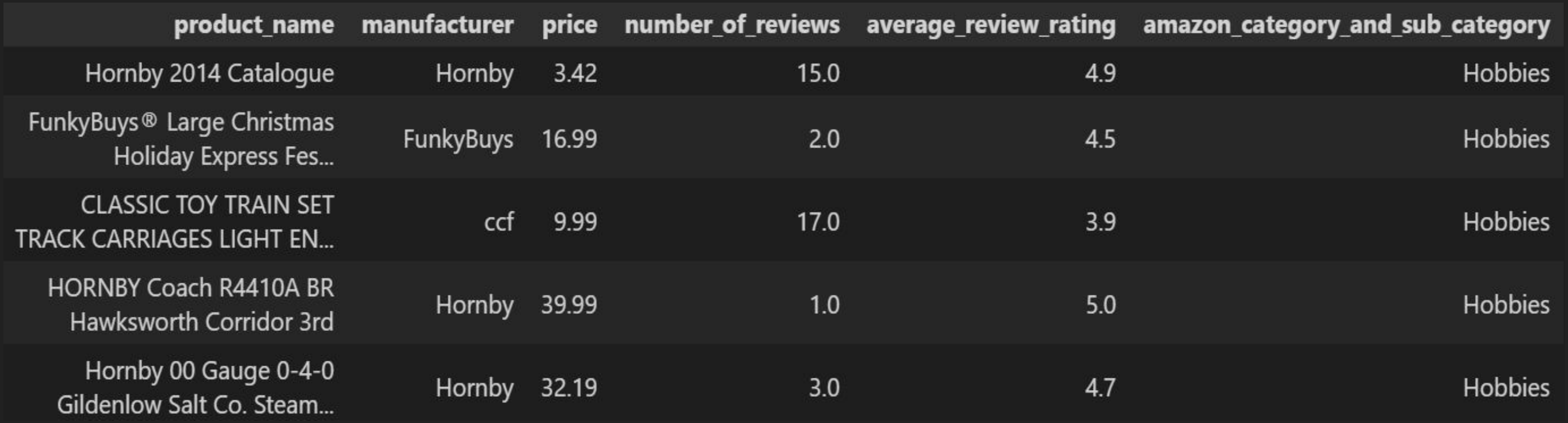
product_name	manufacturer	price	number_of_reviews	average_review_rating	amazon_category_and_sub_category
Hornby 2014 Catalogue	Hornby	£3.42	15	4.9 out of 5 stars	Hobbies > Model Trains & Railway Sets > Rail V...
FunkyBuys® Large Christmas Holiday Express Fes...	FunkyBuys	£16.99	2	4.5 out of 5 stars	Hobbies > Model Trains & Railway Sets > Rail V...
CLASSIC TOY TRAIN SET TRACK CARRIAGES LIGHT EN...	ccf	£9.99	17	3.9 out of 5 stars	Hobbies > Model Trains & Railway Sets > Rail V...
HORNBY Coach R4410A BR Hawksworth Corridor 3rd	Hornby	£39.99	1	5.0 out of 5 stars	Hobbies > Model Trains & Railway Sets > Rail V...
Hornby 00 Gauge 0-4-0 Gildenlow Salt Co. Steam...	Hornby	£32.19	3	4.7 out of 5 stars	Hobbies > Model Trains & Railway Sets > Rail V...

Toy Review Dataset Transformation



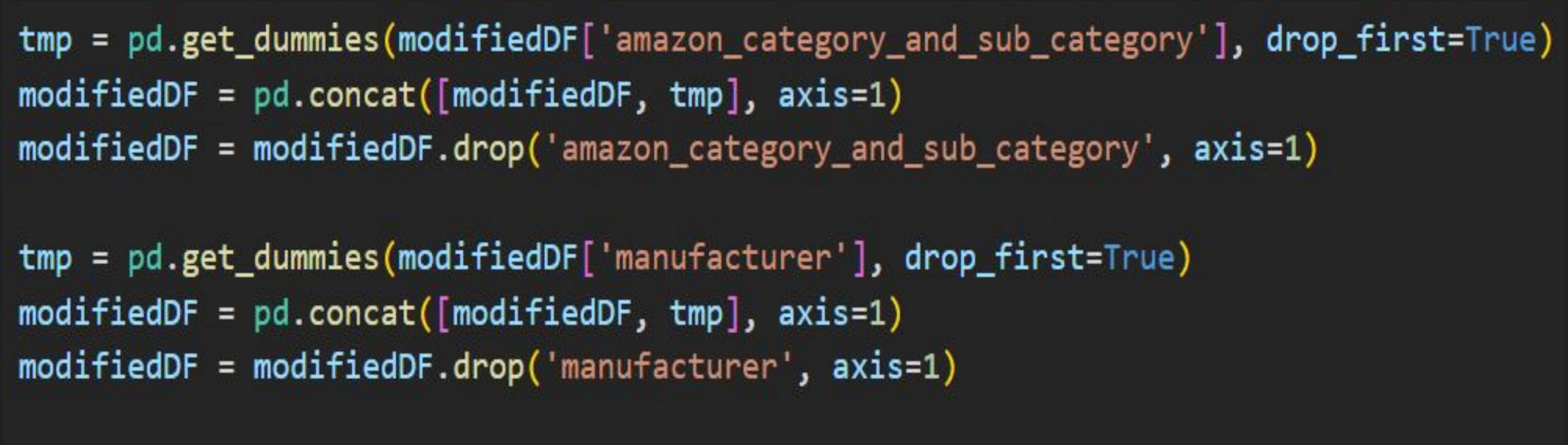
```
# Keep first three charaters from average_review_rating column, convert to int
modifiedDF['average_review_rating'] = df['average_review_rating'].str[:3]
modifiedDF['average_review_rating'] = pd.to_numeric(modifiedDF['average_review_rating'])

# Drop pound symbol from price column, convert to int
modifiedDF['price'] = df['price'].str.replace(r'£', '')
modifiedDF['price'] = modifiedDF['price'].str.replace(r',', '')
modifiedDF['price'] = modifiedDF['price'].str.split('-').str[0]
modifiedDF['price'] = pd.to_numeric(modifiedDF['price'])

# Convert num of reviews to int
modifiedDF['number_of_reviews'] = df['number_of_reviews'].str.replace(r',', '')
modifiedDF['number_of_reviews'] = pd.to_numeric(modifiedDF['number_of_reviews'])

# Fix overfitting by reducing sub categories
modifiedDF['amazon_category_and_sub_category'] = modifiedDF['amazon_category_and_sub_category'].str.split('>').str[0]
```

Toy Review Dataset Transformed



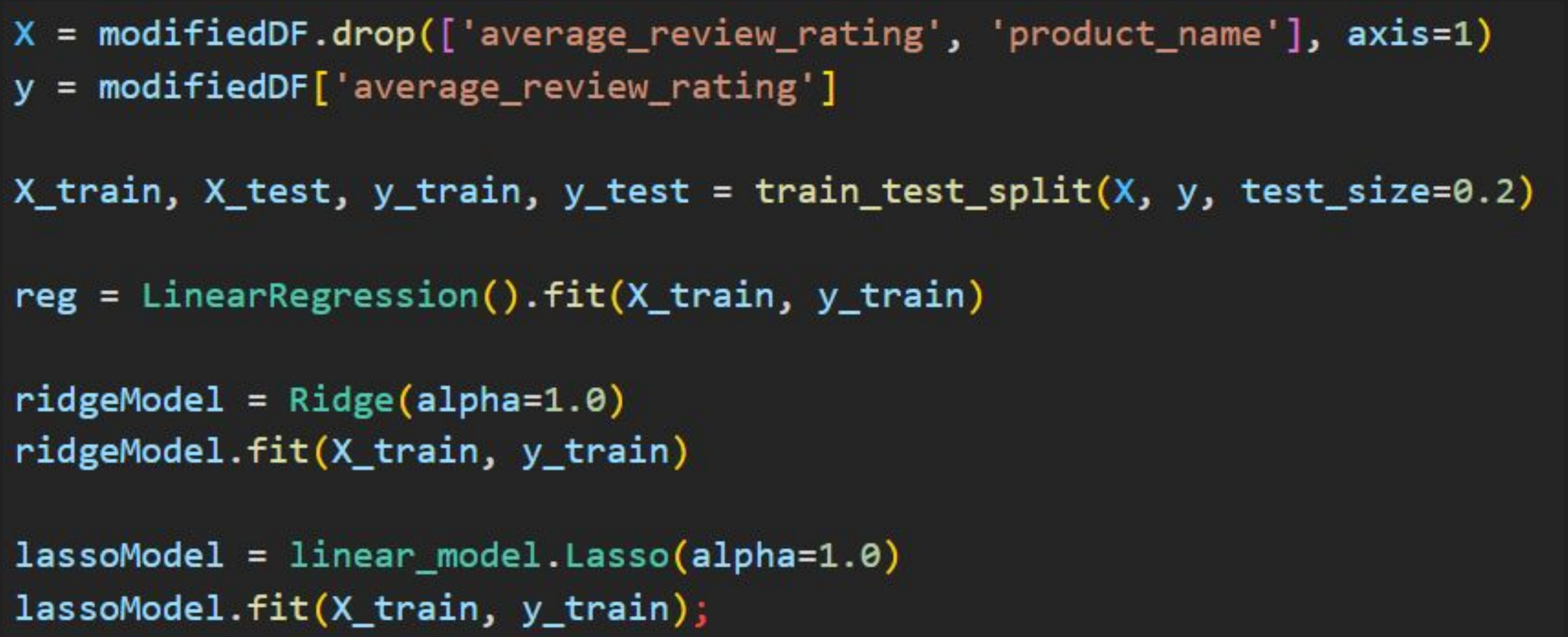
product_name	manufacturer	price	number_of_reviews	average_review_rating	amazon_category_and_sub_category
Hornby 2014 Catalogue	Hornby	3.42	15.0	4.9	Hobbies
FunkyBuys® Large Christmas Holiday Express Fes...	FunkyBuys	16.99	2.0	4.5	Hobbies
CLASSIC TOY TRAIN SET TRACK CARRIAGES LIGHT EN...	ccf	9.99	17.0	3.9	Hobbies
HORNBY Coach R4410A BR Hawksworth Corridor 3rd	Hornby	39.99	1.0	5.0	Hobbies
Hornby 00 Gauge 0-4-0 Gildenlow Salt Co. Steam...	Hornby	32.19	3.0	4.7	Hobbies

Toy Review Dataset - Dummy Variables

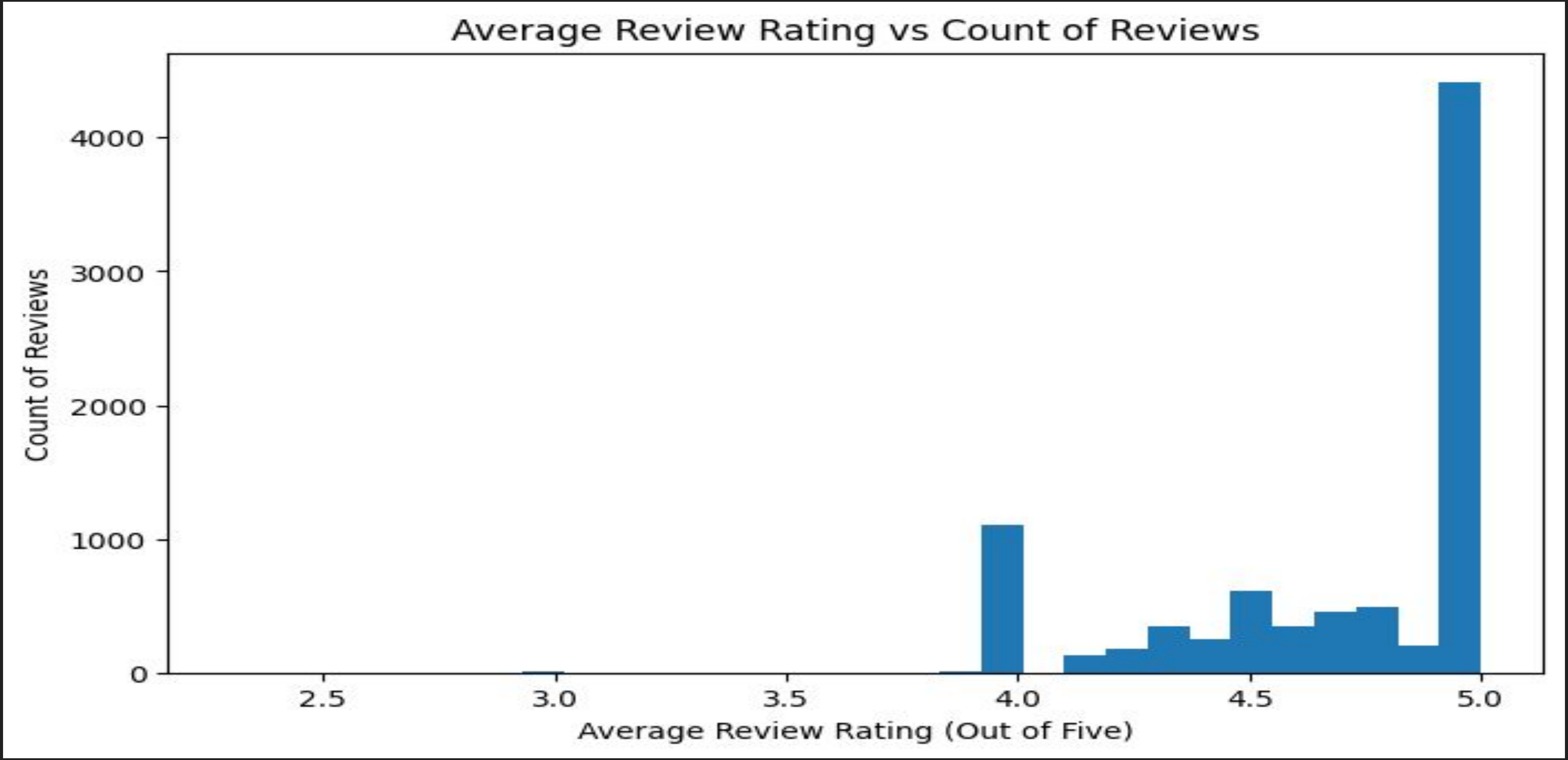


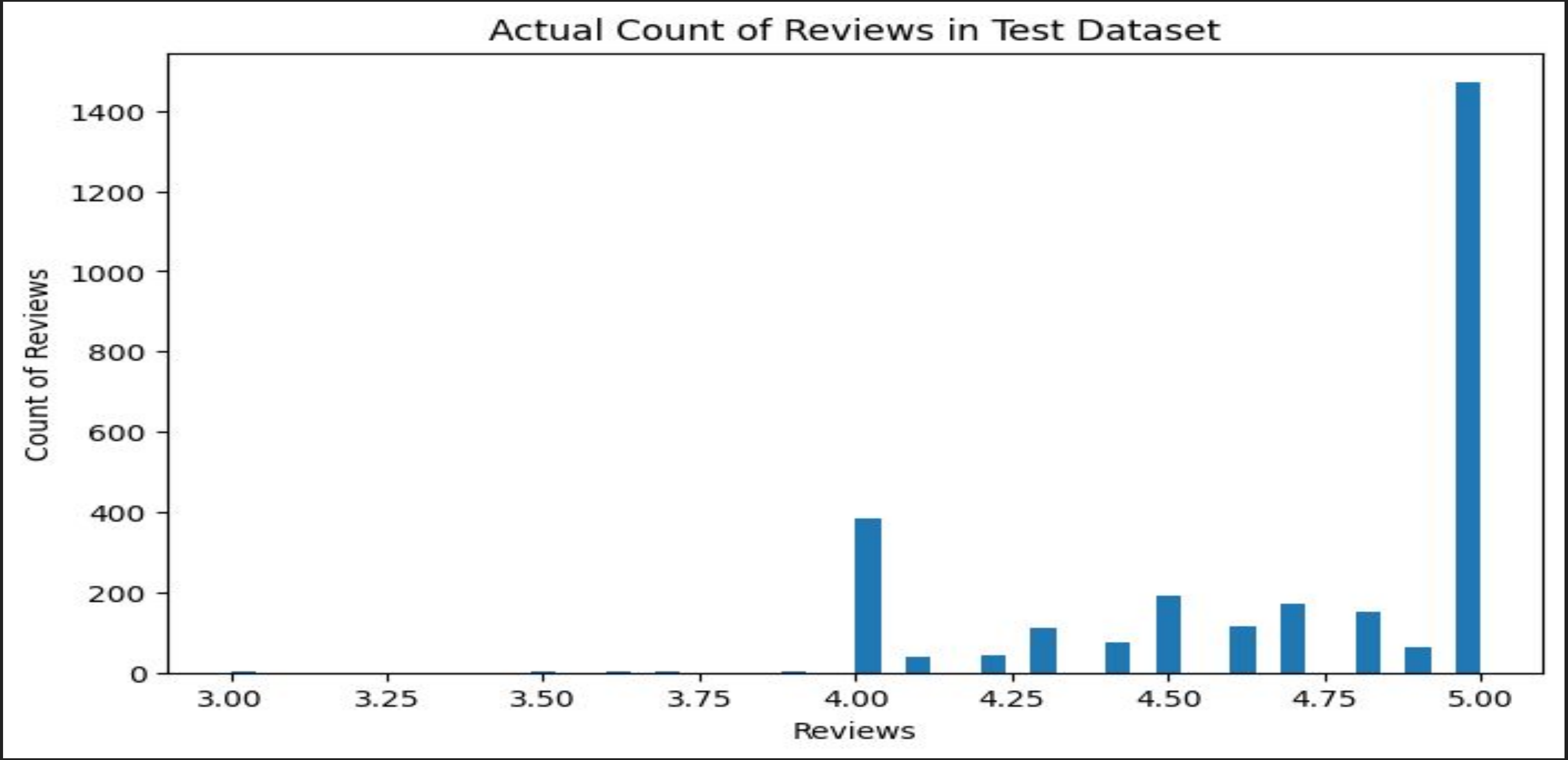
```
tmp = pd.get_dummies(modifiedDF['amazon_category_and_sub_category'], drop_first=True)
modifiedDF = pd.concat([modifiedDF, tmp], axis=1)
modifiedDF = modifiedDF.drop('amazon_category_and_sub_category', axis=1)

tmp = pd.get_dummies(modifiedDF['manufacturer'], drop_first=True)
modifiedDF = pd.concat([modifiedDF, tmp], axis=1)
modifiedDF = modifiedDF.drop('manufacturer', axis=1)
```

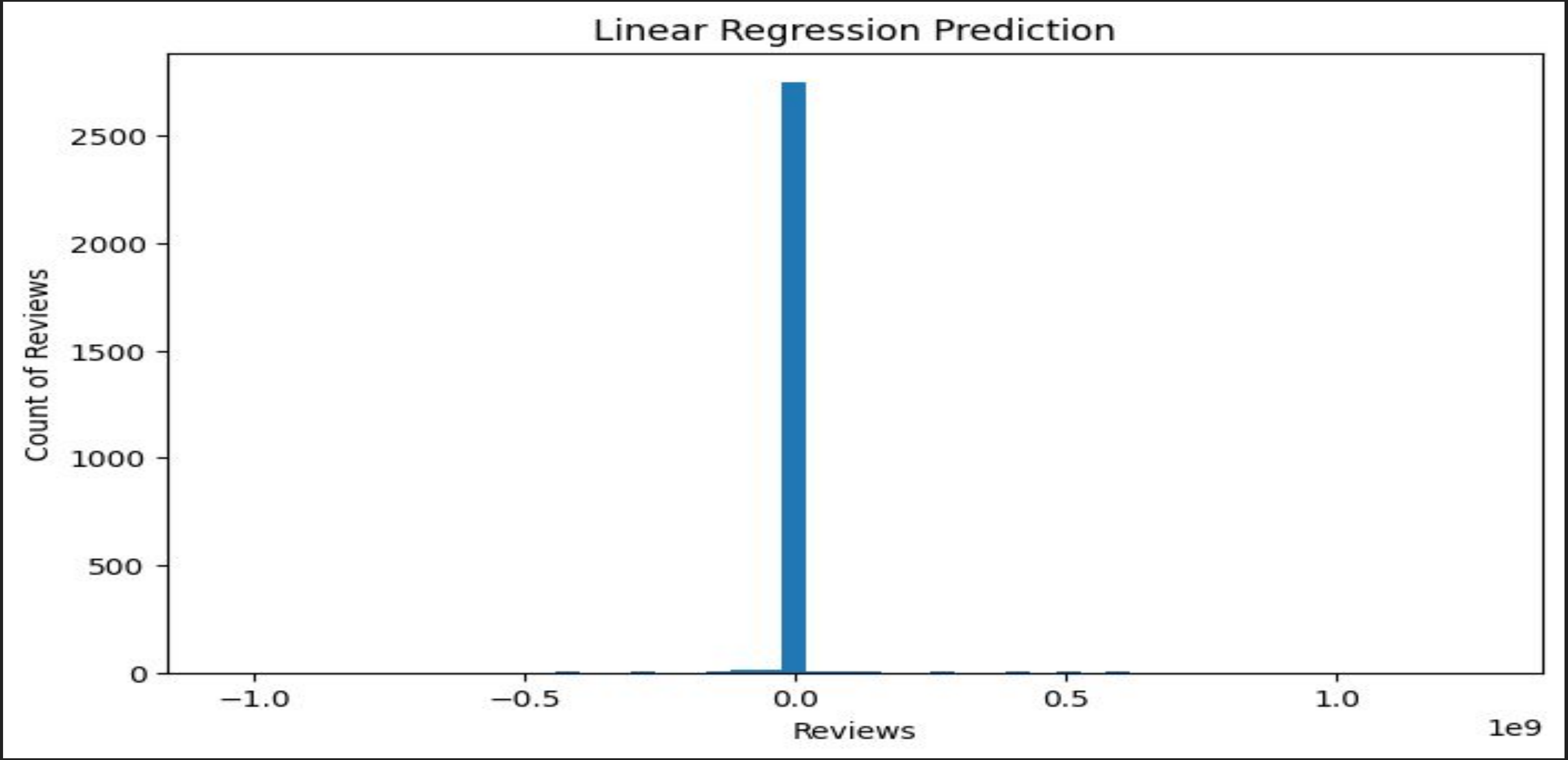
Toy Review Dataset - Fitting the Models

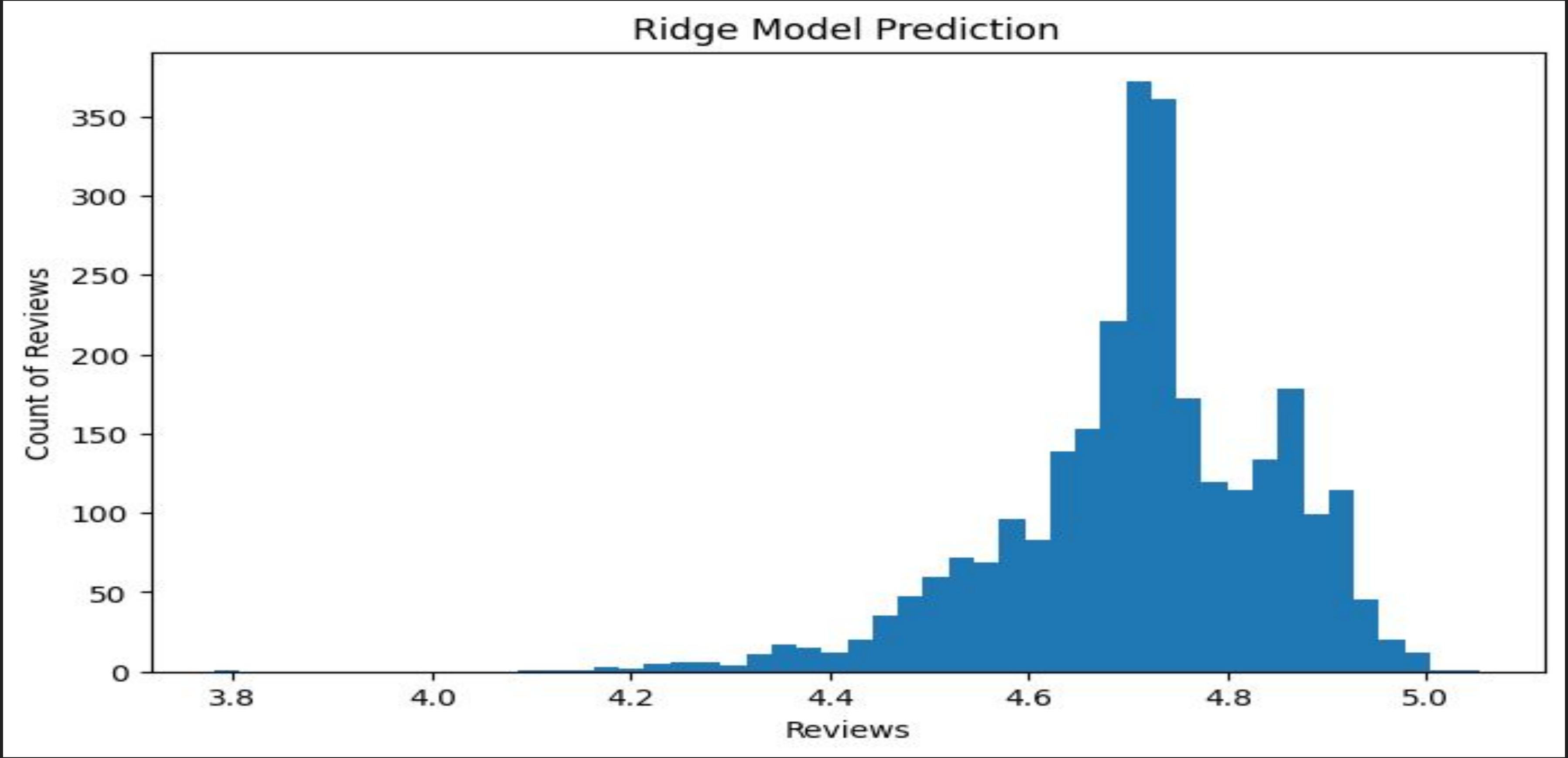


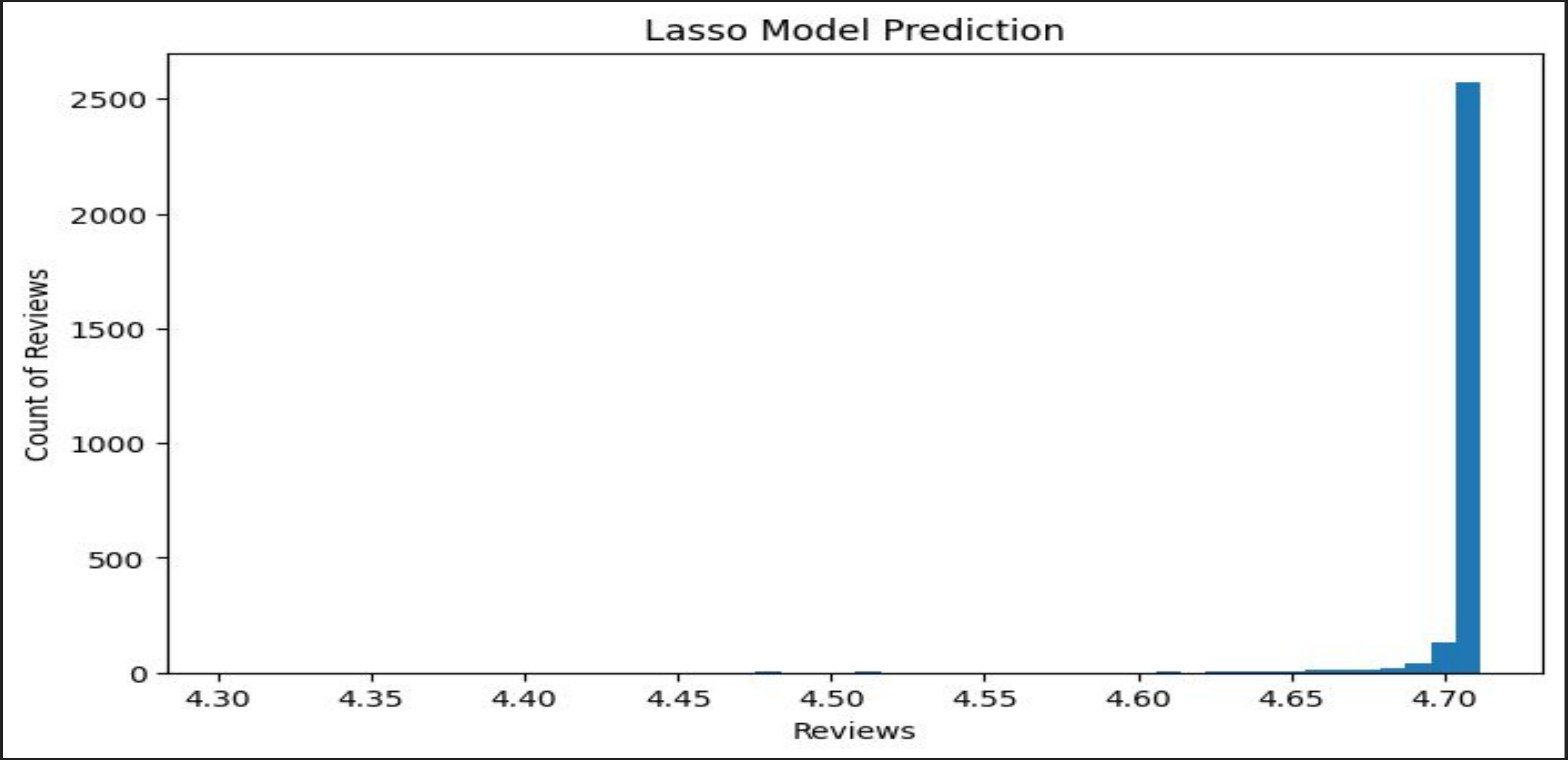
```
X = modifiedDF.drop(['average_review_rating', 'product_name'], axis=1)
y = modifiedDF['average_review_rating']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

reg = LinearRegression().fit(X_train, y_train)

ridgeModel = Ridge(alpha=1.0)
ridgeModel.fit(X_train, y_train)

lassoModel = linear_model.Lasso(alpha=1.0)
lassoModel.fit(X_train, y_train);
```


Average Review Rating vs Count of Reviews
Count of Reviews
0
1000
2000
3000
4000
2.5
3.0
3.5
4.0
4.5
5.0
Average Review Rating (Out of Five)


Actual Count of Reviews in Test Dataset
Count of Reviews
Reviews
0
200
400
600
800
1000
1200
1400
3.00
3.25
3.50
3.75
4.00
4.25
4.50
4.75
5.00


Linear Regression Prediction
Count of Reviews
2500
2000
1500
1000
500
0
−1.0
−0.5
0.0
0.5
1.0
Reviews
1e9


Ridge Model Prediction
Count of Reviews
Reviews
0
50
100
150
200
250
300
350
3.8
4.0
4.2
4.4
4.6
4.8
5.0


Lasso Model Prediction
Count of Reviews
2500
2000
1500
1000
500
0
4.30
4.35
4.40
4.45
4.50
4.55
4.60
4.65
4.70
Reviews

Toy Review Dataset - Correlation

Algorithm	**Correlation Value**
Linear Regression	-0.0054
Ridge	0.139
LASSO	0.111

Links to Resources

When to use LASSO - Crunching the Data

When to use ridge regression - Crunching the Data

Ridge and Lasso Regression Explained (tutorialspoint.com)

Pros and Cons of Common Machine Learning Algorithms

Ridge Documentation 1

Ridge Documentation 2

Lasso Documentation

Ridge vs Lasso Regression, Visualized Video