1 Executive Summary

There is a tradeoff between bias and variance. Reducing variance increases bias in our model and vice versa. We have to find an optimal point while fitting a model so that we make sure the variance is not too high on the training set and the error on the test set also does not increase. We can use Regularization techniques for reducing variance. We have some techniques of regularization:

1) Ridge regularization (L2),
2) Lasso regularization (L1),
3) Elastic Net.

Lasso, Ridge, and Elastic Net are modifications of ordinary least squares linear regression, which use additional penalty terms in the cost function to keep coefficient values small and simplify the model.
The results of our analysis indicate how regulirization techniques can be used to improve the precitive power of a model.

The need for regularization
Regularization can sometimes leads to better model performance. Regularization can be used to avoid overfitting, a more generic model may be preferred over a very specific one

The foundations of a regularizer
Regularizers are attached to the loss values of a machine learning model, and they are thus included in the optimization step. Combining the original loss value with the regularization component,the model will become simpler with likely losing not much of their predictive abilities.

2 Introduction

I plan to build a linear regression model in python to estimate the sales price of houses.Thr dataset contains 244 features which can lead to making an overfitted models. I will test 3 types pf regularization here to see the effect on the performance of the model on both training as well as test set.

3 Loading and Exploring Data

3.1 Loading libraries

In [2]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pandas import Series, DataFrame
from math import sqrt
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import (train_test_split, cross_val_score, RepeatedKFold)
from sklearn.ensemble import GradientBoostingRegressor
from sklearn import metrics
```

3.2 Loading Data

In [419]:
```python
data = pd.read_csv('AmesHousing_Processed.csv')
```

```
The data has already been processed. It is checked for the existance of outliers and missing values. We only need to perform feature scaling.
```

3.3 Data size and structure

In [420]:
```python
data.shape
```

Out[420]: (2925, 244)

Our dataframe consists of 10 predictors and our response variable is "AverageTemperature".

In [505]:
```
# Let us look at some records of our data
data.head()
```

Out[505]:

	Unnamed: 0	MS SubClass	Lot Frontage	Lot Area	Land Slope	Overall Qual	Overall Cond	Year Built	Year Remod/Add	Mas Vnr Area	...	Sale Type_New	Sale Type_Oth	Sale Type_VWD	Sale Type_WD
0	0	20	141.0	31770	0	6	5	1960	1960	112.0	...	0	0	0	1
1	1	20	80.0	11622	0	5	6	1961	1961	0.0	...	0	0	0	1
2	2	20	81.0	14267	0	6	6	1958	1958	108.0	...	0	0	0	1
3	3	20	93.0	11160	0	7	5	1968	1968	0.0	...	0	0	0	1
4	4	60	74.0	13830	0	5	5	1997	1998	0.0	...	0	0	0	1

5 rows × 244 columns

4 EDA

In [422]:
```
data.describe()
```

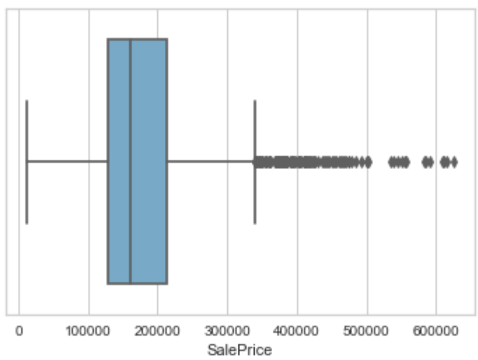
Out[422]:

	Unnamed: 0	MS SubClass	Lot Frontage	Lot Area	Land Slope	Overall Qual	Overall Cond	Year Built	Year Remod/Add	Mas Vn Are
count	2925.000000	2925.000000	2925.000000	2925.000000	2925.000000	2925.000000	2925.000000	2925.000000	2925.000000	2925.00000
mean	1462.000000	57.396581	57.460855	10103.583590	0.053675	6.088205	5.563761	1971.302906	1984.234188	99.91863
std	844.519094	42.668752	33.075613	7781.999124	0.248506	1.402953	1.112262	30.242474	20.861774	175.56615
min	0.000000	20.000000	0.000000	1300.000000	0.000000	1.000000	1.000000	1872.000000	1950.000000	0.00000
25%	731.000000	20.000000	43.000000	7438.000000	0.000000	5.000000	5.000000	1954.000000	1965.000000	0.00000
50%	1462.000000	50.000000	63.000000	9428.000000	0.000000	6.000000	5.000000	1973.000000	1993.000000	0.00000
75%	2193.000000	70.000000	78.000000	11515.000000	0.000000	7.000000	6.000000	2001.000000	2004.000000	162.00000
max	2924.000000	190.000000	313.000000	215245.000000	2.000000	10.000000	9.000000	2010.000000	2010.000000	1600.00000

8 rows × 244 columns

In [506]:
```
sns.boxplot( data["SalePrice"], palette="Blues")
```

Out[506]: <AxesSubplot:xlabel='SalePrice'>



As we saw in our previous analysis the average tempreture has been around 17 degrees

In [507]:
```
data.isna().sum().sum()
```

Out[507]: 0

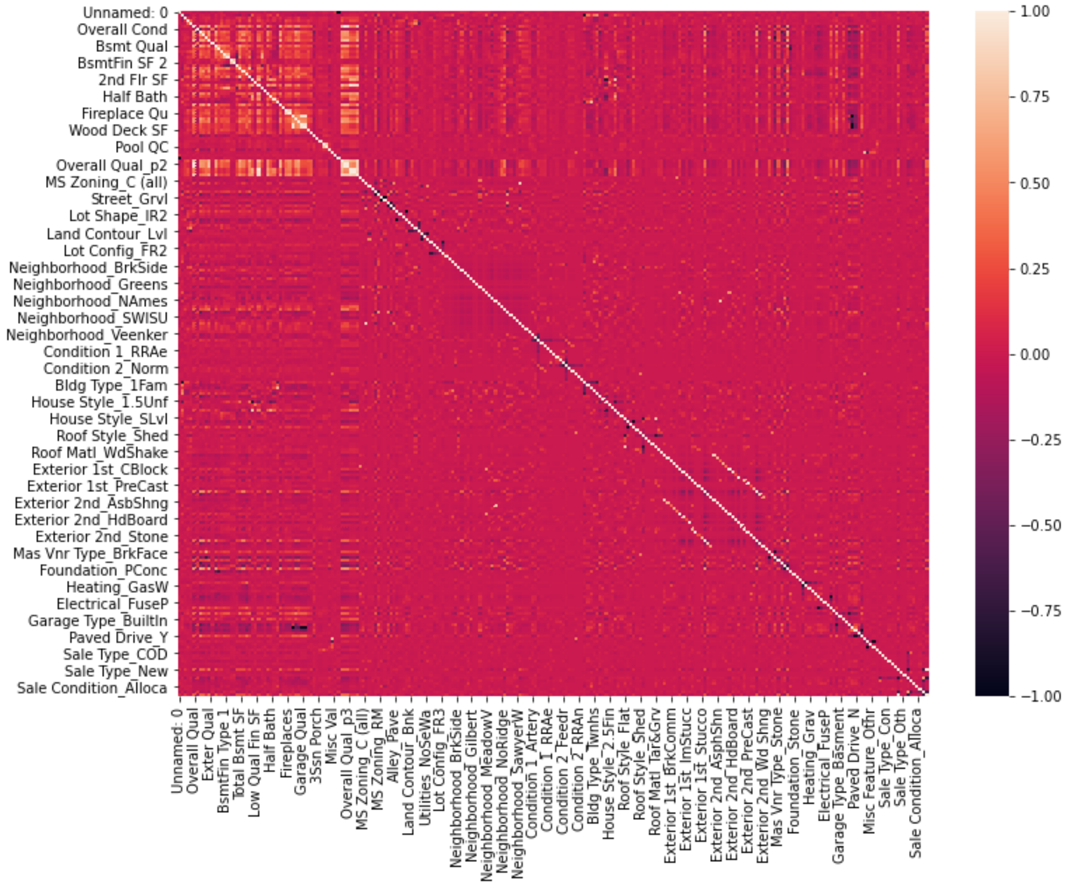
No missing values

In [508]:
```
# Let us keep a copy of our dataset
dataset = data.copy()
```

4.1 The Correlations Plot

In [426]:

```python
fig, ax = plt.subplots(figsize=(12,9))
sns.heatmap(dataset.corr(), ax=ax);
```



4.2 Feature Scaling

In [427]:

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# We need to fit the scaler to our data before transformation
dataset.loc[:, dataset.columns != 'SalePrice'] = scaler.fit_transform(
    dataset.loc[:, dataset.columns != 'SalePrice'])
```

In [428]:

```python
dataset
```

Out[428]:

	Unnamed: 0	MS SubClass	Lot Frontage	Lot Area	Land Slope	Overall Qual	Overall Cond	Year Built	Year Remod/Add	Mas Vnr Area	...	Sale Type_New	Sal Type_Ot
0	-1.731459	-0.876589	2.526134	2.784647	-0.216028	-0.062882	-0.506946	-0.373807	-1.161854	0.068826	...	-0.296252	-0.04897
1	-1.730274	-0.876589	0.681560	0.195152	-0.216028	-0.775786	0.392276	-0.340735	-1.113911	-0.569220	...	-0.296252	-0.04897
2	-1.729090	-0.876589	0.711798	0.535098	-0.216028	-0.062882	0.392276	-0.439950	-1.257739	0.046038	...	-0.296252	-0.04897
3	-1.727906	-0.876589	1.074666	0.135775	-0.216028	0.650022	-0.506946	-0.109233	-0.778312	-0.569220	...	-0.296252	-0.04897
4	-1.726722	0.061025	0.500126	0.478933	-0.216028	-0.775786	-0.506946	0.849847	0.659971	-0.569220	...	-0.296252	-0.04897
...	...	...	...	...	...	...	...	...	...	...	...	...	.
2920	1.726722	0.529832	-0.618714	-0.278457	-0.216028	-0.062882	0.392276	0.419915	-0.011228	-0.569220	...	-0.296252	-0.04897
2921	1.727906	-0.876589	-1.737554	-0.156617	3.808700	-0.775786	-0.506946	0.386843	-0.059170	-0.569220	...	-0.296252	-0.04897
2922	1.729090	0.647034	0.137259	0.043366	-0.216028	-0.775786	-0.506946	0.684489	0.372314	-0.569220	...	-0.296252	-0.04897
2923	1.730274	-0.876589	0.590843	-0.012028	3.808700	-0.775786	-0.506946	0.089198	-0.442712	-0.569220	...	-0.296252	-0.04897
2924	1.731459	0.061025	0.500126	-0.061252	3.808700	0.650022	-0.506946	0.717560	0.468200	-0.033717	...	-0.296252	-0.04897

2925 rows × 244 columns

5 Building the Model

In [429]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn import metrics
Model = LinearRegression()
```

In [430]:
```
x_train, x_test, y_train, y_test = train_test_split(
    dataset.drop('SalePrice', axis=1), dataset[['SalePrice']],
    test_size=0.3, random_state=101)
```

In [431]:
```
Model.fit(x_train, y_train)
```

Out[431]: LinearRegression()

5.1 Model Score on Training and Test Set

In [432]:
```
Model.score(x_train,y_train)
```

Out[432]: 0.9481149995701218

In [433]:
```
Model.score(x_test,y_test)
```

Out[433]: -1.4468301664039589e+20

The model score is not good on the test set.

5.2 Model Prediction

In [434]:
```
pred = Model.predict(x_test)
```

In [435]:
```
# Let us define a function for caculating RMSE
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
```

In [436]:
```
rmse_val = rmse(np.array(pred), np.array(y_test))
results = pd.DataFrame()
results["Method"] = ["Linear Regression-All Features"]
results["RMSE"] = rmse_val
results
```

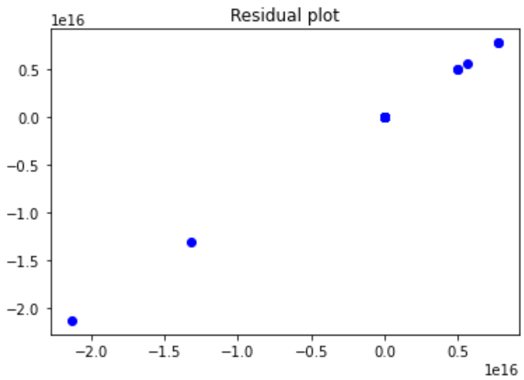
Out[436]:

	Method	RMSE
0	Linear Regression-All Features	9.717902e+14

When you overfit the training data, the test MSE will be very large because the supposed patterns that the method found in the training data simply don't exist in the test data

In [440]:
```
# Let us plot the residuls of our  model
plt.scatter(pred, (pred-y_test), c='b')
#plt.hlines(y=0,xmin= 0, xmax=100)
plt.title('Residual plot')
```

Out[440]: Text(0.5, 1.0, 'Residual plot')



If we can detect a clear pattern or trend in our residuals, then our model has room for improvement.

5.3 Intercept and Coeficcients

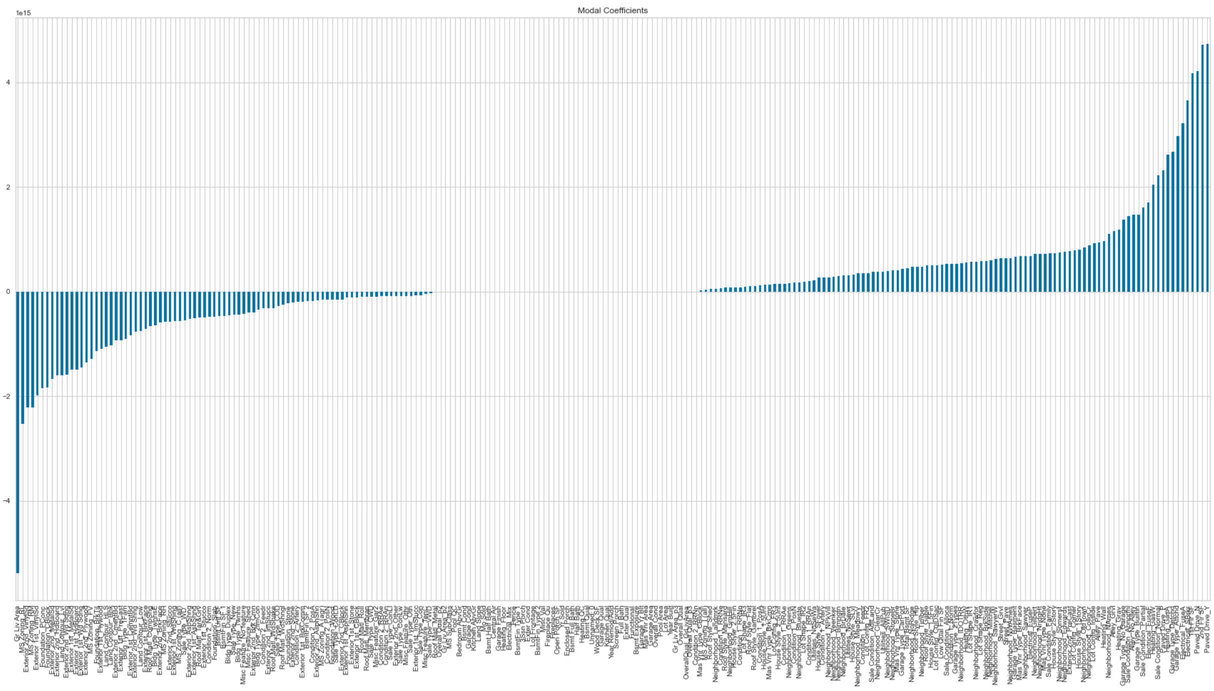
In [297]:
```
#print (Model.coef_)
```

In [448]:
```
print (Model.intercept_)
```

```
[-1.15373485e+12]
```

In [449]:
```
plt.figure(figsize=(30,15))
predictors = x_train.columns
coef = Series(Model.coef_[0],predictors).sort_values()
coef.plot(kind='bar', title='Modal Coefficients')
```

Out[449]:
```
<AxesSubplot:title={'center':'Modal Coefficients'}>
```



5.4 Ridge regularization (L2)

In [450]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
ridge = Ridge()
```

5.4.1 Hyper parameter tuning

In [451]:
```
parameters = {'alpha': [1e-15, 1e-10, 1e-8, 1e-4, 1e-3,1e-2, 1, 5, 10, 20, 25 , 50 , 60, 100, 200]}
ridgeReg = GridSearchCV(ridge, parameters,scoring='neg_mean_squared_error', cv=5)
```

In [452]:
```
ridgeReg.fit(x_train,y_train)
```

Out[452]:
```
GridSearchCV(cv=5, estimator=Ridge(),
             param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.0001, 0.001, 0.01, 1,
                                   5, 10, 20, 25, 50, 60, 100, 200]},
             scoring='neg_mean_squared_error')
```

In [453]:
```
ridgeReg.best_params_
```

Out[453]:
```
{'alpha': 20}
```

In [454]:
```
ridgeReg.best_score_
```

Out[454]:
```
-453794162.24202394
```

In [455]:
```
ridgeReg = Ridge(alpha=20, normalize=True)
ridgeReg.fit(x_train,y_train)
```

Out[455]:
```
Ridge(alpha=20, normalize=True)
```

5.5 Model Score on Training and Test Set

In [456]:
```
ridgeReg.score(x_train, y_train)
```
Out[456]:
```
0.6079077829943409
```

In [457]:
```
ridgeReg.score(x_test, y_test)
```
Out[457]:
```
-0.7235885835714739
```

5.6 Model Prediction

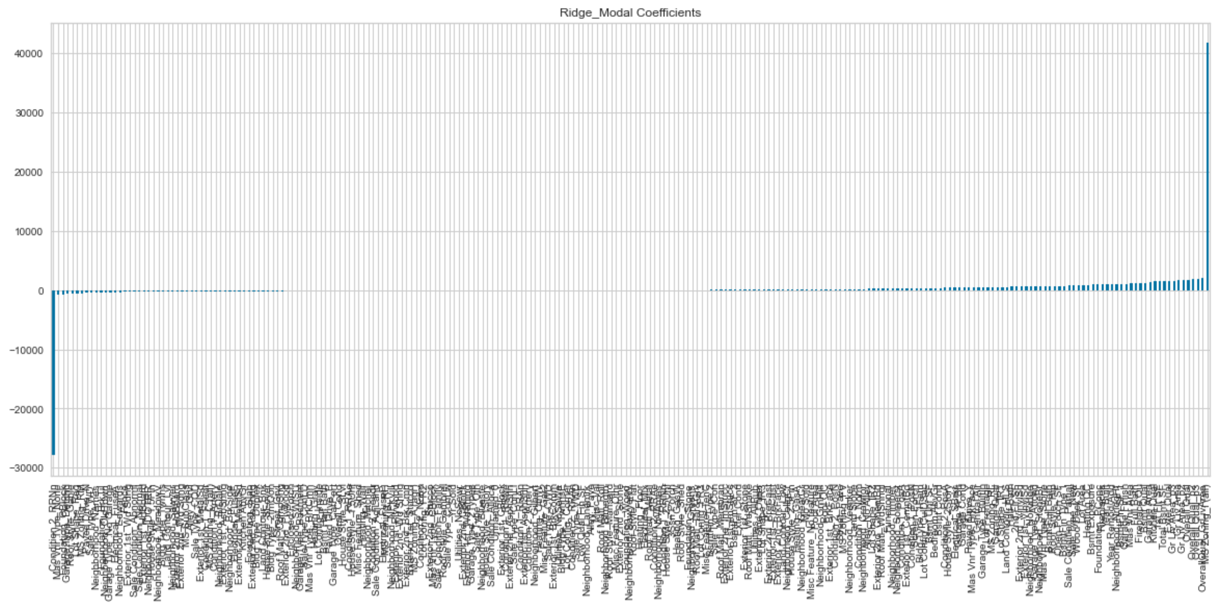
In [458]:
```
pred = ridgeReg.predict(x_test)
```

In [459]:
```
rmse_val = rmse(np.array(pred), np.array(y_test))
results.loc[1] = ["RidgeReg", rmse_val]
results
```
Out[459]:

	Method	RMSE
0	Linear Regression-All Features	9.717902e+14
1	RidgeReg	1.060671e+05

5.6.1 Ridge_Modal Coefficients

In [460]:
```
%matplotlib inline
plt.figure(figsize=(20,8))
predictors = x_train.columns
coef = Series(ridgeReg.coef_[0],predictors).sort_values()
coef.plot(kind='bar', title='Ridge_Modal Coefficients')
```
Out[460]:
```
<AxesSubplot:title={'center':'Ridge_Modal Coefficients'}>
```



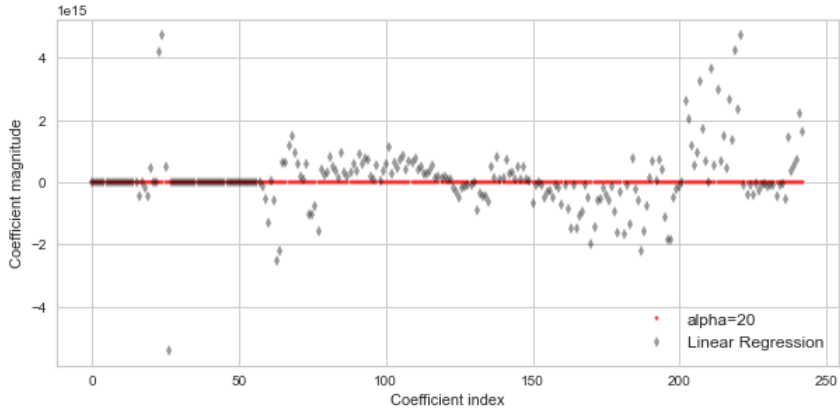
5.6.2 Coefficient Magnitude & Coefficient Index

In [461]:
```
ridgeReg2 = Ridge(alpha=60, normalize=True)
ridgeReg2.fit(x_train,y_train)
```
Out[461]:
```
Ridge(alpha=60, normalize=True)
```

In [462]:
```
ridgeReg3 = Ridge(alpha=1e-15, normalize=True)
ridgeReg3.fit(x_train,y_train)
```
Out[462]:
```
Ridge(alpha=1e-15, normalize=True)
```

5.6.3 Comparison with Linear Regression

In [465]:
```
# Let us plot the Coefficient magnetitude of oue modes with respect to the alpha values ↔
```



This is an example of shrinking coefficient magnitude using Ridge regression. For alpha = 60, we can see the coefficient is nearly zero, which is not the case for alpha=1e-15. For small values of α (1e-15) in which the coefficient is less restricted, the magnitudes of the coefficient is almost the same as of linear regression.

5.7 Lasso regularization (L1)

In [466]:
```
from sklearn.linear_model import Lasso
lasso = Lasso()
```

5.7.1 Hyper parameter tuning

In [467]:
```
parameters = {'alpha': [1e-15, 1, 5, 10, 20, 25, 50, 60, 100, 500, 750, 1000]}
lassoReg = GridSearchCV(lasso, parameters, scoring='neg_mean_squared_error', cv = 5)
lassoReg.fit(x_train,y_train)
```

Out[467]:
```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': [1e-15, 1, 5, 10, 20, 25, 50, 60, 100, 500,
                                   750, 1000]},
             scoring='neg_mean_squared_error')
```

In [468]:
```
lassoReg.best_params_
```

Out[468]:
```
{'alpha': 100}
```

In [469]:
```
lassoReg.best_score_
```

Out[469]:
```
-435972196.32547045
```

In [470]:
```
lassoReg = Lasso(alpha=100, normalize=True)
lassoReg.fit(x_train,y_train)
```

Out[470]:
```
Lasso(alpha=100, normalize=True)
```

5.8 Model Score on Training and Test Set

In [471]:
```
lassoReg.score(x_train,y_train)
```

Out[471]:
```
0.8962518805250732
```

In [472]:
```
lassoReg.score(x_test,y_test)
```

Out[472]:
```
0.8799222725945898
```

5.9 Model Prediction

In [473]:
```
predic = lassoReg.predict(x_test)
```

In [474]:
```
pred = pred.reshape(-1,1)
```

In [475]:
```
lassoReg.coef_[0]
```

Out[475]: 0.0

In [476]:
```
rmse_val = rmse(np.array(pred), np.array(y_test))
results.loc[2] = ["lassoReg", rmse_val]
results
```

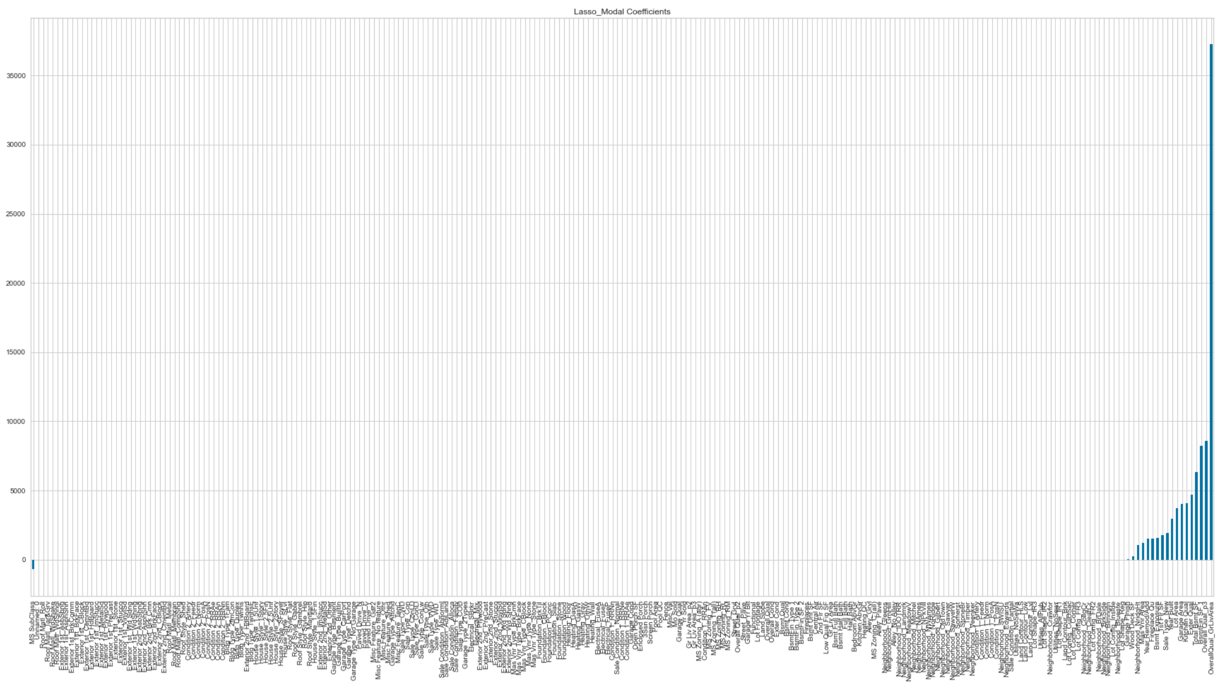
Out[476]:

	Method	RMSE
0	Linear Regression-All Features	9.717902e+14
1	RidgeReg	1.060671e+05
2	lassoReg	1.060671e+05

5.9.1 Lasso_Modal Coefficients

In [477]:
```
plt.figure(figsize=(30,15))
predictors = x_train.columns
coef = Series(lassoReg.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Lasso_Modal Coefficients')
```

Out[477]: <AxesSubplot:title={'center':'Lasso_Modal Coefficients'}>



5.9.2 Coefficient Magnitude & Coefficient Index

In [478]:
```
lassoReg2 = Lasso(alpha=500, normalize=True)
lassoReg2.fit(x_train,y_train)
```

Out[478]: Lasso(alpha=500, normalize=True)

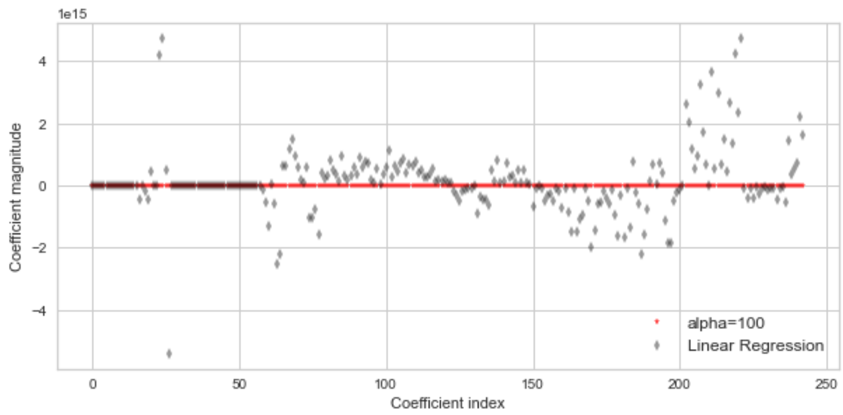
In [479]:
```
lassoReg3 = Lasso(alpha=1000, normalize=True)
lassoReg3.fit(x_train,y_train)
```

Out[479]: Lasso(alpha=1000, normalize=True)

5.9.3 Comparison with Linear Regression

In [503]:
```
# Let us plot the Coefficient magnetitude of oue modes with respect to the alpha values ↔
```

No handles with labels found to put in legend.



For larger values of alpha [20, 60], we can see most of the coefficients are zero or nearly zero.

5.10 Elastic Net

In [486]:
```
from sklearn.linear_model import ElasticNet
```

5.10.1 Hyper parameter tuning

In [489]:
```
import numpy as np
from sklearn.model_selection import GridSearchCV
parametersGrid = {"max_iter": [1, 5, 10],
                      "alpha": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
                      "l1_ratio": np.arange(0.0, 1.0, 0.1)}
eNet = ElasticNet()
grid = GridSearchCV(eNet, parametersGrid, scoring='r2', cv=10)
```

In [490]:
```
grid.fit(x_train, y_train)
```

Out[490]:
```
GridSearchCV(cv=10, estimator=ElasticNet(),
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
                         'l1_ratio': array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]),
                         'max_iter': [1, 5, 10]},
             scoring='r2')
```

In [491]:
```
grid.best_params_
```

Out[491]:
```
{'alpha': 1, 'l1_ratio': 0.4, 'max_iter': 10}
```

In [492]:
```
grid.best_score_
```

Out[492]:
```
0.9179132390772669
```

In [495]:
```
ENreg = ElasticNet(alpha=1, l1_ratio=0.4, max_iter =10, normalize=False)
ENreg.fit(x_train, y_train)
```

Out[495]:
```
ElasticNet(alpha=1, l1_ratio=0.4, max_iter=10)
```

5.11 Model Score on Training and Test Set

In [496]:
```
ENreg.score(x_train,y_train)
```

Out[496]:
```
0.9300815903071482
```

In [497]:
```
ENreg.score(x_test,y_test)
```

Out[497]: 0.9080795970022025

5.12 Model Prediction

In [498]:
```
pred = ENreg.predict(x_test)
```

In [499]:
```
pred=pred.reshape(-1,1)
```

In [500]:
```
rmse_val = rmse(np.array(pred), np.array(y_test))
results.loc[3] = ["ElasticNet", rmse_val]
results
```

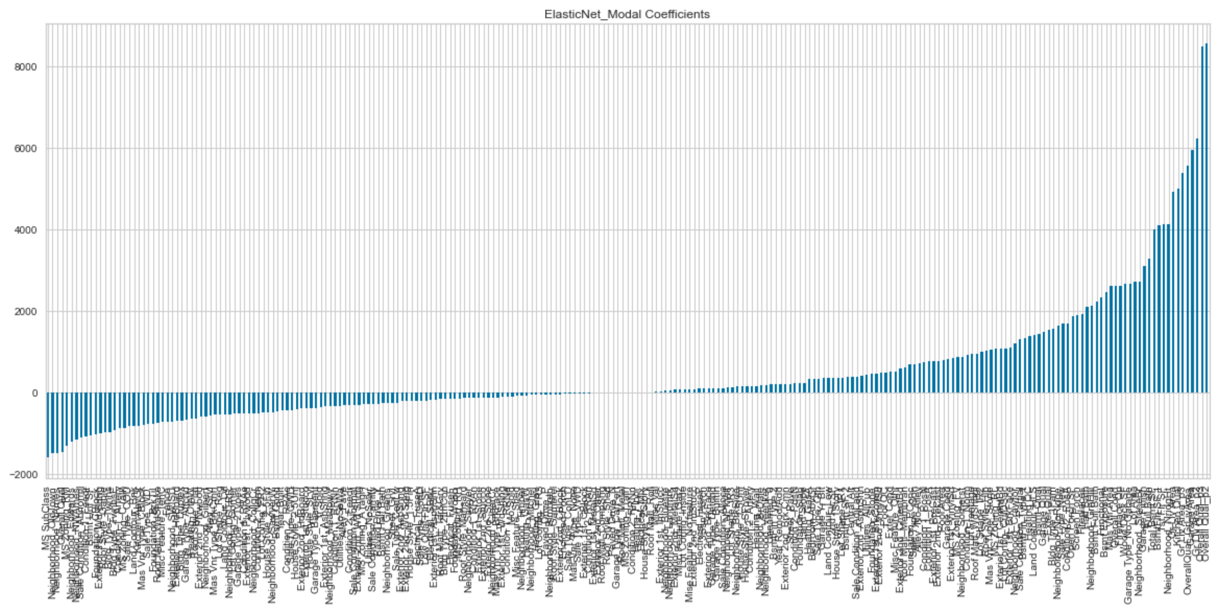
Out[500]:

	Method	RMSE
0	Linear Regression-All Features	9.717902e+14
1	RidgeReg	1.060671e+05
2	lassoReg	1.060671e+05
3	ElasticNet	2.449456e+04

5.12.1 ElasticNet_Modal Coefficients

In [501]:
```
%matplotlib inline
plt.figure(figsize=(20,8))
predictors = x_train.columns
coef = Series(ENreg.coef_,predictors).sort_values()
coef.plot(kind='bar', title='ElasticNet_Modal Coefficients')
```

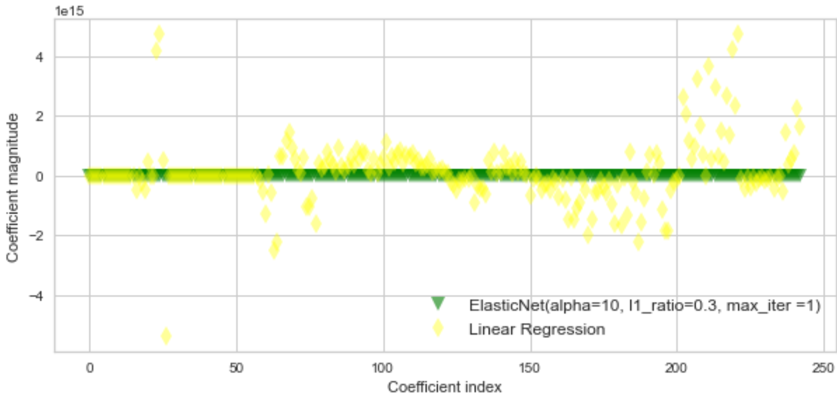
Out[501]: <AxesSubplot:title={'center':'ElasticNet_Modal Coefficients'}>



5.12.2 Comparison with Linear Regression

In [502]:

```
1 # let us plot the Coefficient magnetitude of oue modes with respect to the alpha values ↔
```



6 Conclusion

The results of our analysis indicate how regulirization techniques can be used to improve the precitive power of a model. In this case study we observed that lots of the predictors are not associated with our responce varaiable therefore, their coeffcients better be ommited from the model. We also showed the importance of parameter tuning for having an optimized version of the model.