Machine Learning I

Assignment I

Institute of Business Administration - IBA

Regression Kaggle Competition

Submitted By:

Bilal Naseem - ERP ID: 13216

Date: March 11, 2023

Abstract

This report details the steps taken to prepare and analyze a dataset for a Kaggle regression competition. Various algorithms, including linear regression, random forest, and gradient boosting regressor, were applied, along with feature selection and transformation techniques. XGBoost yielded the best results, with an RMSE of 1911.87712 on the Kaggle leaderboard. The report provides insights into the performance of each algorithm and feature selection/transformation approach.

Instructions:

The first competition (Regression problem) has been posted on Kaggle. You can access it via the following link:
`https://www.kaggle.com/t/a32b7844fe2644a790e26ca9b48d17c7`
You are expected to participate in the competition individually and use your proper full name. This is important because many people have similar first names and it may create problems at the time of grading. Please submit a report in MS Word describing the data preparation steps you took and the algorithms you applied. Describe in sufficient details what works for you against which algorithm. There should be a section on your feature selection/transformation effort (be it using one-hot or label encoding, feature selection function in sklearn, selection using p-values, variance/correlation filters, lasso/ridge based feature selection, and/or anything else). Also write in detail the impact of doing interaction and polynomial (or even higher order) on different models' performance.
You may eventually submit 100s of submissions in an attempt to become the leader on the leaderboard but I would still like you to specify what worked for you against these algorithms:

- Linear Regression
- Lasso Regression
- Ridge Regression
- Elastic Net Regression
- Regression Tree
- Random Forest Regressor
- Gradient Boosting Regressor
- k-NN Regressor

Contents

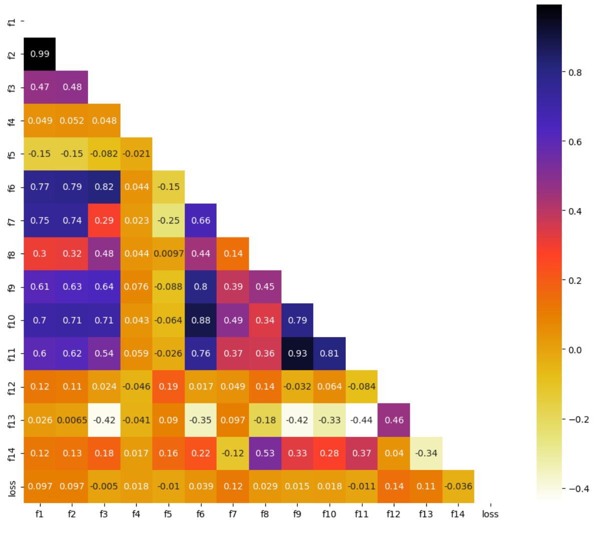
1 Introduction - Exploring Dataset

The training data set consisted of 150,000 rows with 132 columns, of which 14 were numerical (excluding row id and loss) and rest were categorical.

1.1 Exploring Numerical Features

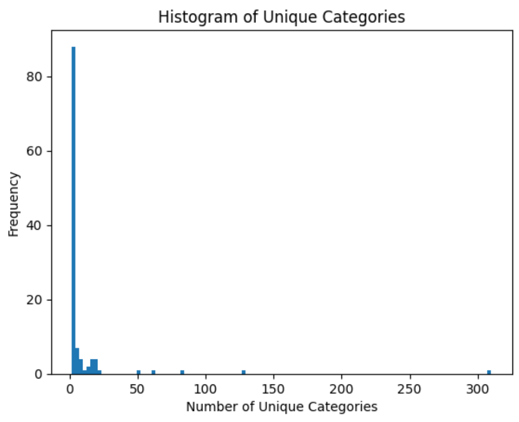
Variance and correlation of numerical columns was found, and the following plots were obtained

Column	Variance
f1	0.0440139
f2	0.0438951
f3	0.0452965
f4	0.0494812
f5	0.0437151
f6	0.0421973
f7	0.0317821
f8	0.0398503
f9	0.0330563
f10	0.0345538
f11	0.0352433
f12	0.0428711
f13	0.0408529
f14	0.0447291



1.2 Exploring Categorical Features

Number of unique categories in each categorical column was found, and the following plot was obtained:



It can be seen from the above image that there are 5 data sets with greater than 50 categories, one of which as greater than 300 categories. Also some of have more than 10 categories, but most of them have 2 categories.
Distribution of columns having ≥ 50 categories was found, and the following results were obtained:

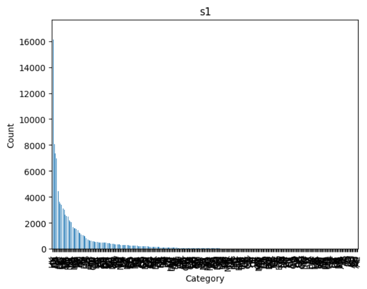


Figure 1: s1

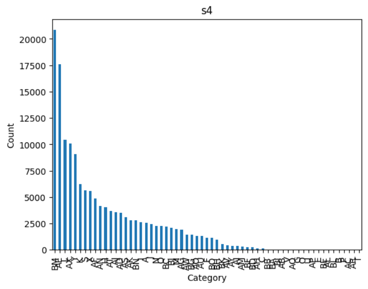


Figure 2: s4

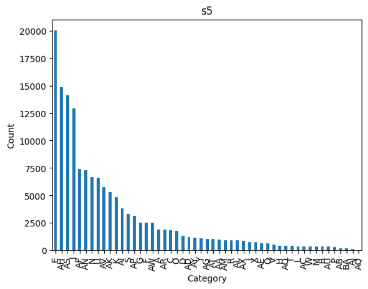


Figure 3: s5

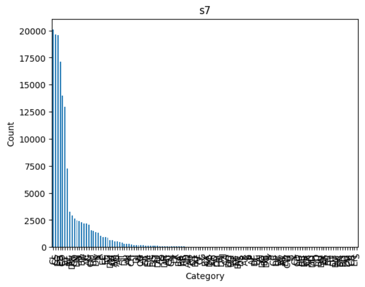


Figure 4: s7

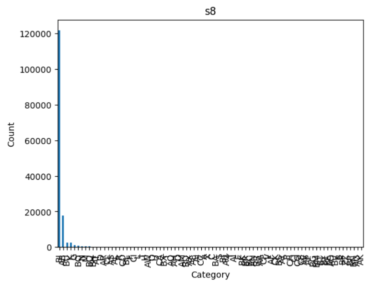


Figure 5: s8

It can be seen that in these columns are also highly dominated by specific type of categories, especially s1, s7 and s8.

2 Data Processing & Feature Engineering

Since most of the features are categorical, there are only a few features which have very high number of categories, I made several data sets out of the original by dropping columns ≥ 5, 10, 20, 50 and 100 unique categories. This would prevent the number of columns to increase exponentially when applying One-Hot Encoding. Code for this is present in Appendix A

2.1 One-Hot Encoding

Based on results of section 1.2, I only encoded top features of each categorical column (not encoding those categories which occur ≤ 1000 times in each categorical column).

This was done through scikit-learn's OneHotEncoder, and the code is present in Appendix B. After this the dataframe obtained consisted of 333 columns, as opposed to 1036 features which I got through regular One-Hot Encoding.

2.2 Feature Reduction through p values and Lasso

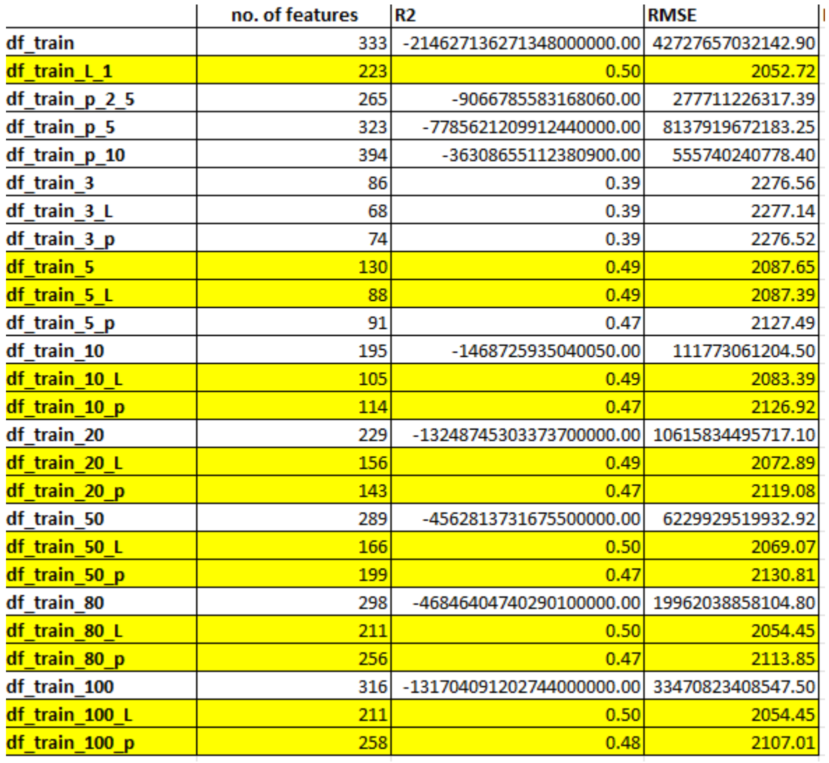
In order to further filter out relevant features, on each of these data sets which I had made, additional Feature Reduction was done through p-values and Lasso Regression. For p-values, significance level of 5% and 2.5% was tried. For Lasso, Cross validation was done to get the best value of alpha, which came out to be 1. Code through which this was done is present in Appendix C and D.

3 Implementation of Machine Learning Models

I applied the following algorithms on all of the data sets which I had made earlier. One Hot Encoding of Top features gave better results, so I only included that.

3.1 Linear Regression

Linear Regression was applied on all the data sets which I had made, and got the following results:

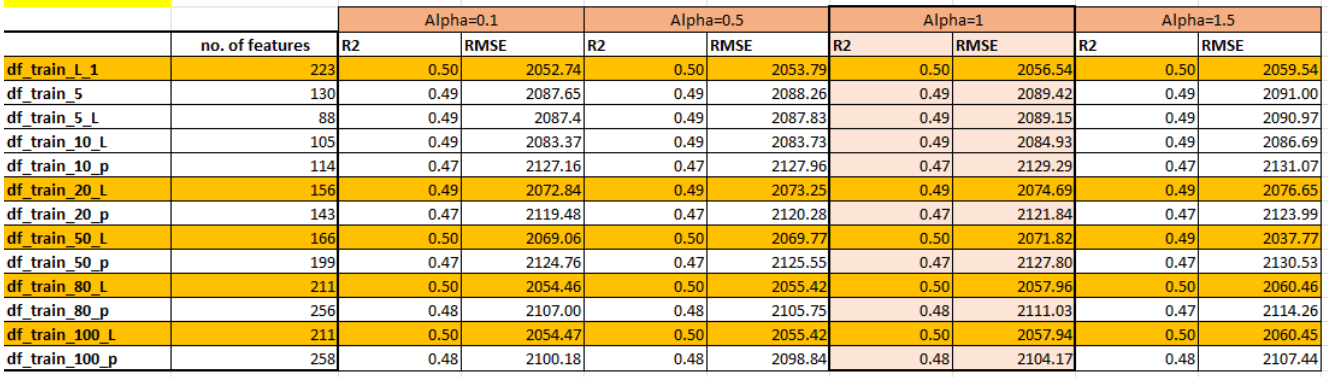


	no. of features	R2	RMSE
df_train	333	-2146271362713480000000.00	42727657032142.90
df_train_L_1	223	0.50	2052.72
df_train_p_2_5	265	-9066785583168060.00	277711226317.39
df_train_p_5	323	-7785621209912440000.00	8137919672183.25
df_train_p_10	394	-36308655112380900.00	555740240778.40
df_train_3	86	0.39	2276.56
df_train_3_L	68	0.39	2277.14
df_train_3_p	74	0.39	2276.52
df_train_5	130	0.49	2087.65
df_train_5_L	88	0.49	2087.39
df_train_5_p	91	0.47	2127.49
df_train_10	195	-1468725935040050.00	111773061204.50
df_train_10_L	105	0.49	2083.39
df_train_10_p	114	0.47	2126.92
df_train_20	229	-13248745303373700000.00	10615834495717.10
df_train_20_L	156	0.49	2072.89
df_train_20_p	143	0.47	2119.08
df_train_50	289	-4562813731675500000.00	6229929519932.92
df_train_50_L	166	0.50	2069.07
df_train_50_p	199	0.47	2130.81
df_train_80	298	-46846404740290100000.00	19962038858104.80
df_train_80_L	211	0.50	2054.45
df_train_80_p	256	0.47	2113.85
df_train_100	316	-131704091202744000000.00	33470823408547.50
df_train_100_L	211	0.50	2054.45
df_train_100_p	258	0.48	2107.01

The highlighted data sets gave the best score, and so further algorithms were only applied on those datasets.

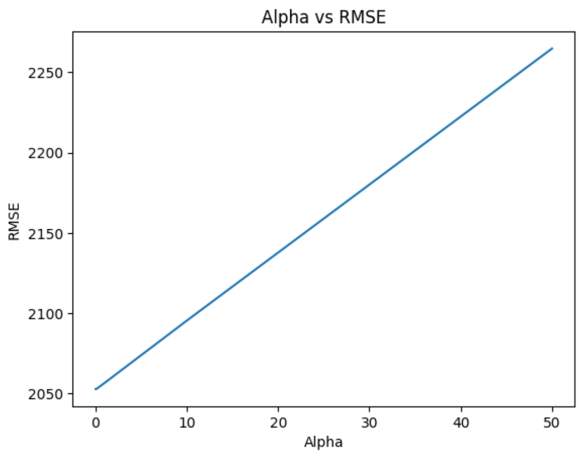
3.2 Lasso Regression

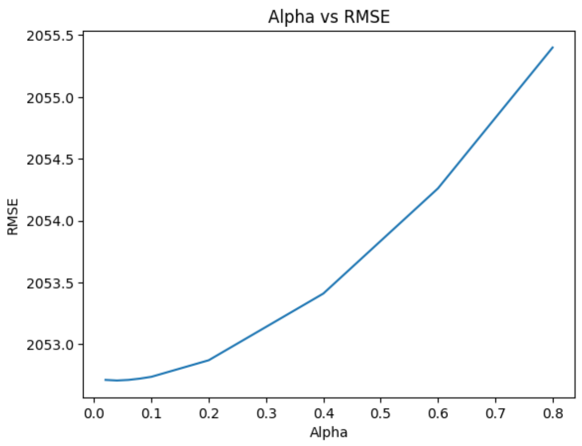
Applying Lasso Regression on the highlighted data sets of section 3.1, with values of α=0.1, 0.5, 1 and 1.5, gave the following results:



		Alpha=0.1		Alpha=0.5		Alpha=1		Alpha=1.5	
	no. of features	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE
df_train_L_1	223	0.50	2052.74	0.50	2053.79	0.50	2056.54	0.50	2059.54
df_train_5	130	0.49	2087.65	0.49	2088.26	0.49	2089.42	0.49	2091.00
df_train_5_L	88	0.49	2087.4	0.49	2087.83	0.49	2089.15	0.49	2090.97
df_train_10_L	105	0.49	2083.37	0.49	2083.73	0.49	2084.93	0.49	2086.69
df_train_10_p	114	0.47	2127.16	0.47	2127.96	0.47	2129.29	0.47	2131.07
df_train_20_L	156	0.49	2072.84	0.49	2073.25	0.49	2074.69	0.49	2076.65
df_train_20_p	143	0.47	2119.48	0.47	2120.28	0.47	2121.84	0.47	2123.99
df_train_50_L	166	0.50	2069.06	0.50	2069.77	0.50	2071.82	0.49	2037.77
df_train_50_p	199	0.47	2124.76	0.47	2125.55	0.47	2127.80	0.47	2130.53
df_train_80_L	211	0.50	2054.46	0.50	2055.42	0.50	2057.96	0.50	2060.46
df_train_80_p	256	0.48	2107.00	0.48	2105.75	0.48	2111.03	0.47	2114.26
df_train_100_L	211	0.50	2054.47	0.50	2055.42	0.50	2057.94	0.50	2060.45
df_train_100_p	258	0.48	2100.18	0.48	2098.84	0.48	2104.17	0.48	2107.44

Increasing α beyond 1.5 only decreases the RMSe. Also α=1 gave the best results, and data sets `df_train_L_1`, `df_train_20_L`, `df_train_50_L`, `df_train_80_L` and `df_train_100_L` gave the best results.
To further investigate the best range of α plots were made of RMSE vs alpha: The plots

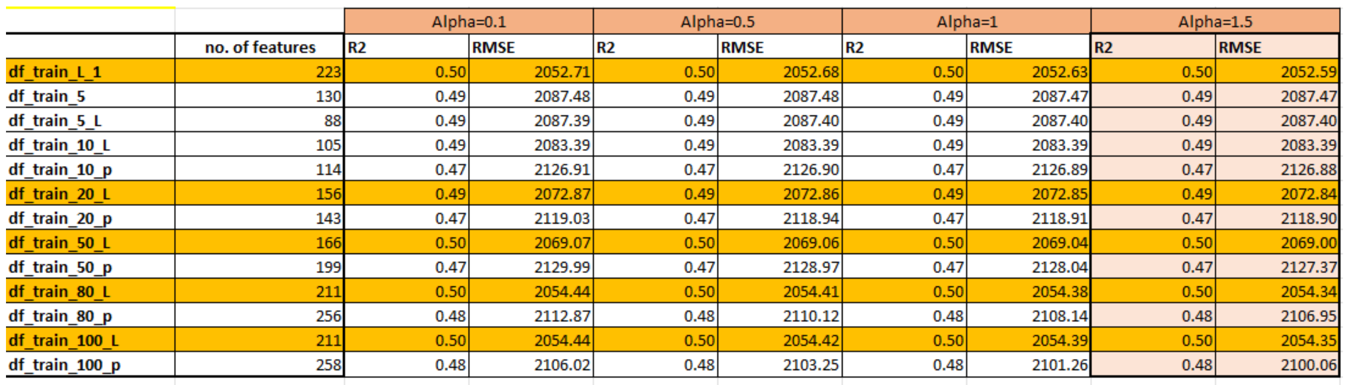




show setting α=0.01 to be the optimum value, and on `df_train_L_1` it gave r^2=0.50, and RMSE=2052.72.

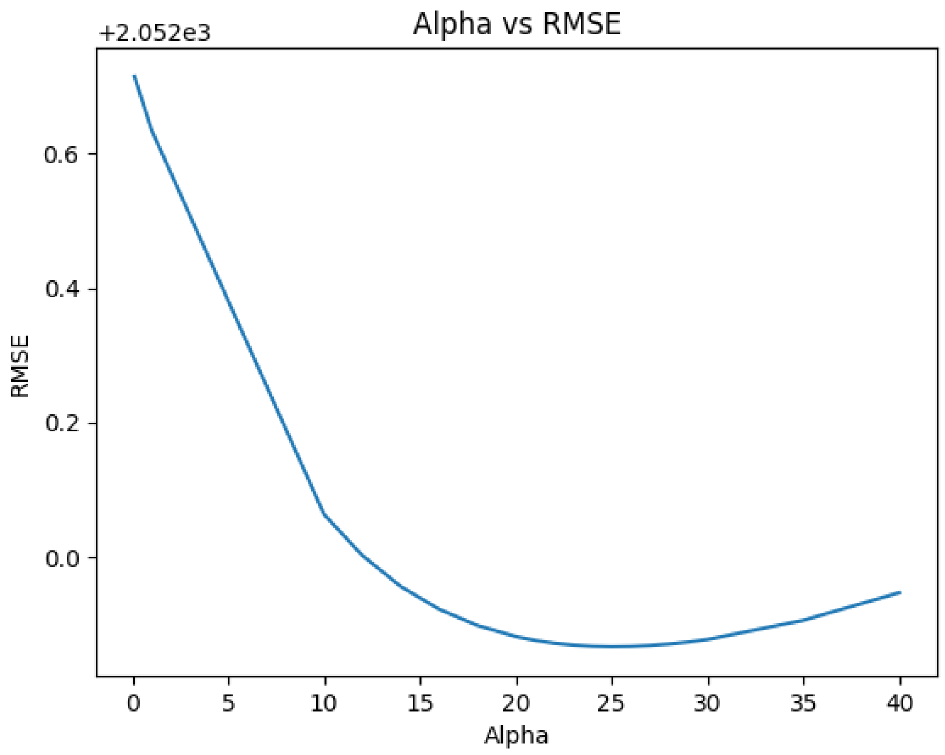
3.3 Ridge Regression

Applying Ridge Regression on the highlighted data sets of section 3.1, with values of α=0.1, 0.5, 1 and 1.5, gave the following results:



		Alpha=0.1		Alpha=0.5		Alpha=1		Alpha=1.5	
	no. of features	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE
df_train_L_1	223	0.50	2052.71	0.50	2052.68	0.50	2052.63	0.50	2052.59
df_train_5	130	0.49	2087.48	0.49	2087.48	0.49	2087.47	0.49	2087.47
df_train_5_L	88	0.49	2087.39	0.49	2087.40	0.49	2087.40	0.49	2087.40
df_train_10_L	105	0.49	2083.39	0.49	2083.39	0.49	2083.39	0.49	2083.39
df_train_10_p	114	0.47	2126.91	0.47	2126.90	0.47	2126.89	0.47	2126.88
df_train_20_L	156	0.49	2072.87	0.49	2072.86	0.49	2072.85	0.49	2072.84
df_train_20_p	143	0.47	2119.03	0.47	2118.94	0.47	2118.91	0.47	2118.90
df_train_50_L	166	0.50	2069.07	0.50	2069.06	0.50	2069.04	0.50	2069.00
df_train_50_p	199	0.47	2129.99	0.47	2128.97	0.47	2128.04	0.47	2127.37
df_train_80_L	211	0.50	2054.44	0.50	2054.41	0.50	2054.38	0.50	2054.34
df_train_80_p	256	0.48	2112.87	0.48	2110.12	0.48	2108.14	0.48	2106.95
df_train_100_L	211	0.50	2054.44	0.50	2054.42	0.50	2054.39	0.50	2054.35
df_train_100_p	258	0.48	2106.02	0.48	2103.25	0.48	2101.26	0.48	2100.06

Again data sets df_train_L_1, df_train_20_L, df_train_50_L, df_train_80_L and df_train_100_L gave the best results. However this time higher value of alpha gave better results. So to further investigate the best value of α in ridge regression, a plot was made of RMSE vs α on df_train_L_1 which gave the best results. The following plot was obtained.

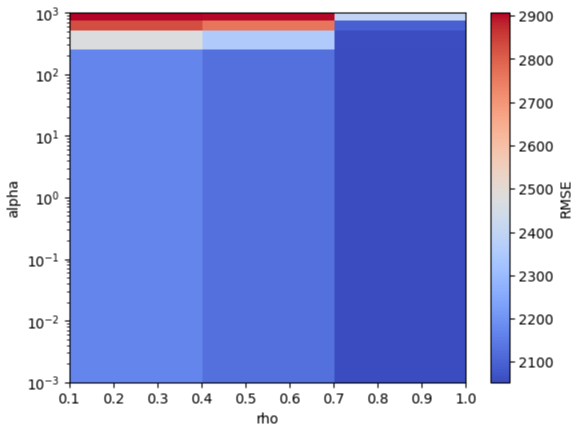


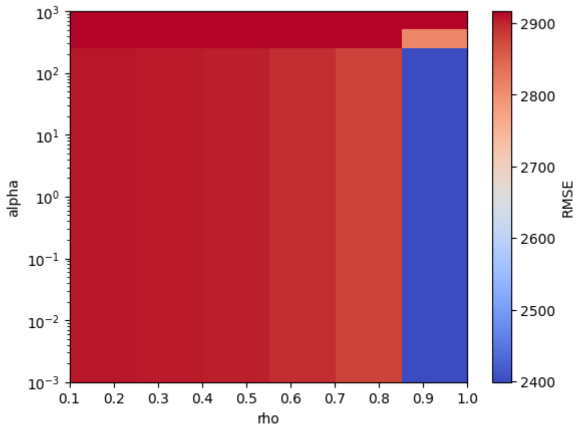
The plots show setting α=25 to be the optimum value, and on df_train_L_1 it gave r^2=0.51, and RMSE=2051.87. The code of the plot can be found in Appendix E.

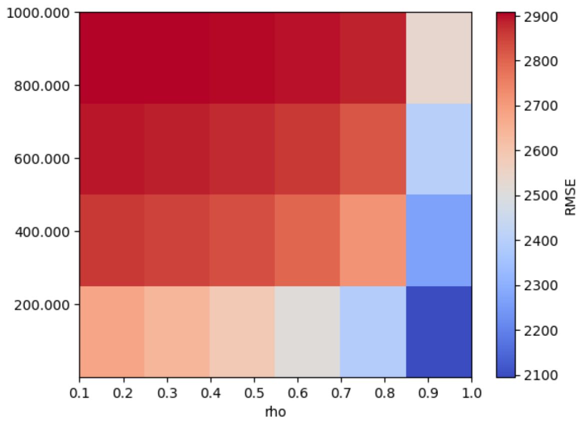
3.4 Elastic-Net Regression

Elastic-Net Regression was only performed on data sets df_train_L_1, df_train_20_L, df_train_50_L, df_train_80_L and df_train_100_L as they gave the

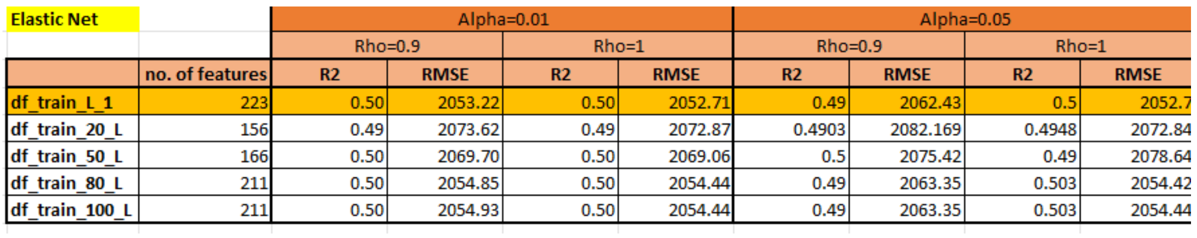
best results in the previous 3 sections.
To find the optimal Range of hyper-parameters of elastic net regression, a heatmap was made of RMSE vs α and L1 ratio on `df_train_L_1`. The following results were obtained.







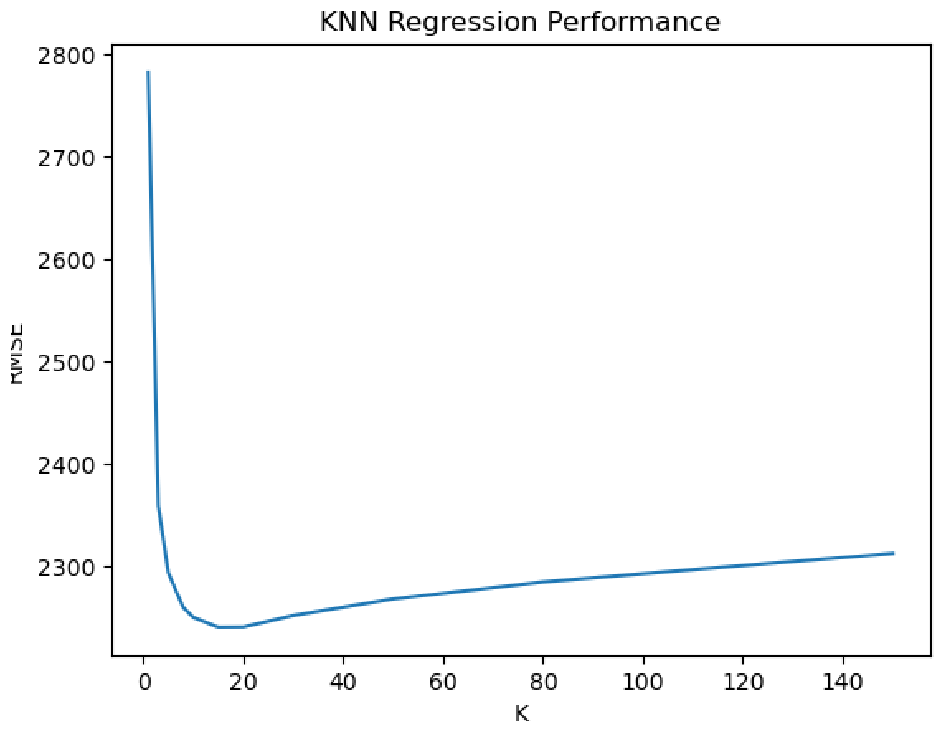
Parameters `alphas = [0.01, 0.05] and rhos = [ 0.9, 1]` were applied to data sets `df_train_L_1`, `df_train_20_L`, `df_train_50_L`, `df_train_80_L` and `df_train_100_L` and the following results were obtained



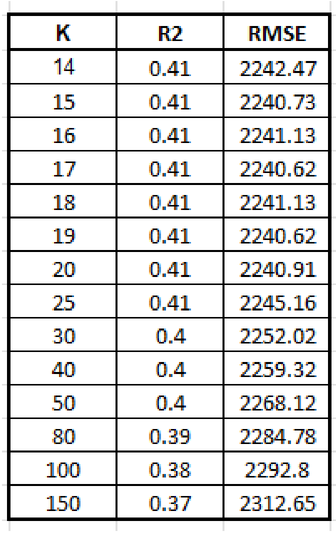
Elastic Net		Alpha=0.01				Alpha=0.05			
		Rho=0.9		Rho=1		Rho=0.9		Rho=1	
	no. of features	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE
df_train_L_1	223	0.50	2053.22	0.50	2052.71	0.49	2062.43	0.5	2052.7
df_train_20_L	156	0.49	2073.62	0.49	2072.87	0.4903	2082.169	0.4948	2072.84
df_train_50_L	166	0.50	2069.70	0.50	2069.06	0.5	2075.42	0.49	2078.64
df_train_80_L	211	0.50	2054.85	0.50	2054.44	0.49	2063.35	0.503	2054.42
df_train_100_L	211	0.50	2054.93	0.50	2054.44	0.49	2063.35	0.503	2054.44

3.5 k-NN Regressor

It can be clearly seen from the previous 4 sections that the data set `df_train_L_1` outperforms all other data sets consistently. And so from this section onwards only this data set would be used.
In order to find the optimal range value for k in KNN Regression, different values of k were plotted against RMSE. Code for this plot is present in appendix H.



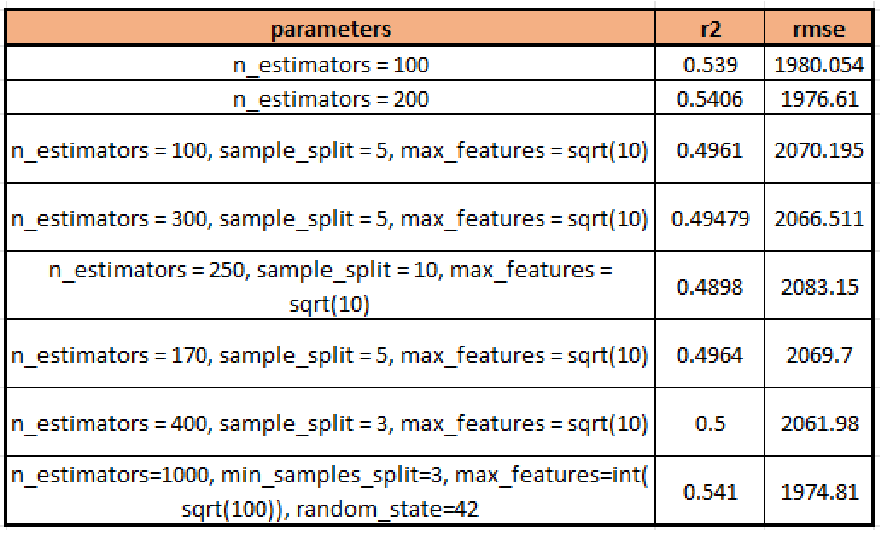
And so K-NN Regression was run on the dataset with `k= [15, 20, 25, 30, 40, 50, 80, 100, 150]`. The following results were obtained.



K	R2	RMSE
14	0.41	2242.47
15	0.41	2240.73
16	0.41	2241.13
17	0.41	2240.62
18	0.41	2241.13
19	0.41	2240.62
20	0.41	2240.91
25	0.41	2245.16
30	0.4	2252.02
40	0.4	2259.32
50	0.4	2268.12
80	0.39	2284.78
100	0.38	2292.8
150	0.37	2312.65

3.6 Random Forest Regressor

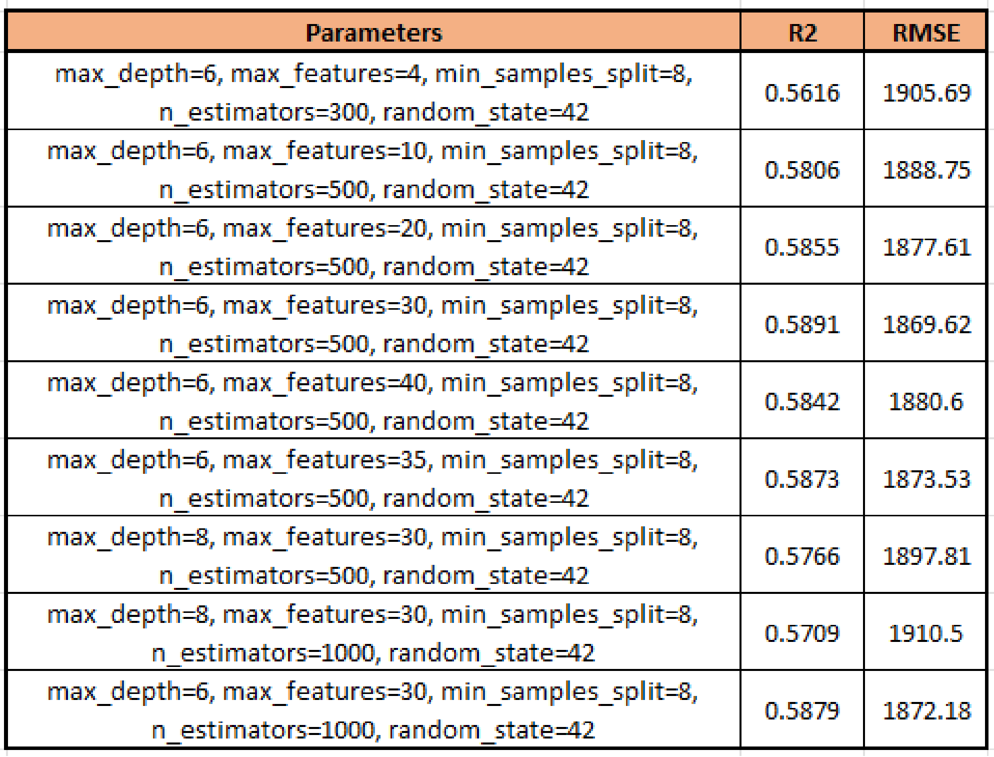
Random Forest Regressor was applied on `df_train_L_1` with different parameters. The following results were obtained:



parameters	r2	rmse
n_estimators = 100	0.539	1980.054
n_estimators = 200	0.5406	1976.61
n_estimators = 100, sample_split = 5, max_features = sqrt(10)	0.4961	2070.195
n_estimators = 300, sample_split = 5, max_features = sqrt(10)	0.49479	2066.511
n_estimators = 250, sample_split = 10, max_features = sqrt(10)	0.4898	2083.15
n_estimators = 170, sample_split = 5, max_features = sqrt(10)	0.4964	2069.7
n_estimators = 400, sample_split = 3, max_features = sqrt(10)	0.5	2061.98
n_estimators=1000, min_samples_split=3, max_features=int(sqrt(100)), random_state=42	0.541	1974.81

3.7 Gradient Boosting Regressor

Gradient Boost was applied on `df_train_L_1` with different parameters. The following results were obtained:



Parameters	R2	RMSE
max_depth=6, max_features=4, min_samples_split=8, n_estimators=300, random_state=42	0.5616	1905.69
max_depth=6, max_features=10, min_samples_split=8, n_estimators=500, random_state=42	0.5806	1888.75
max_depth=6, max_features=20, min_samples_split=8, n_estimators=500, random_state=42	0.5855	1877.61
max_depth=6, max_features=30, min_samples_split=8, n_estimators=500, random_state=42	0.5891	1869.62
max_depth=6, max_features=40, min_samples_split=8, n_estimators=500, random_state=42	0.5842	1880.6
max_depth=6, max_features=35, min_samples_split=8, n_estimators=500, random_state=42	0.5873	1873.53
max_depth=8, max_features=30, min_samples_split=8, n_estimators=500, random_state=42	0.5766	1897.81
max_depth=8, max_features=30, min_samples_split=8, n_estimators=1000, random_state=42	0.5709	1910.5
max_depth=6, max_features=30, min_samples_split=8, n_estimators=1000, random_state=42	0.5879	1872.18

3.7.1 Results through Randomized Search

Randomized Search was performed as in Appendix G, and the following optimal best parameters were obtained:

- n estimators : 300
- min samples split: 4
- max features: None
- max depth: 5

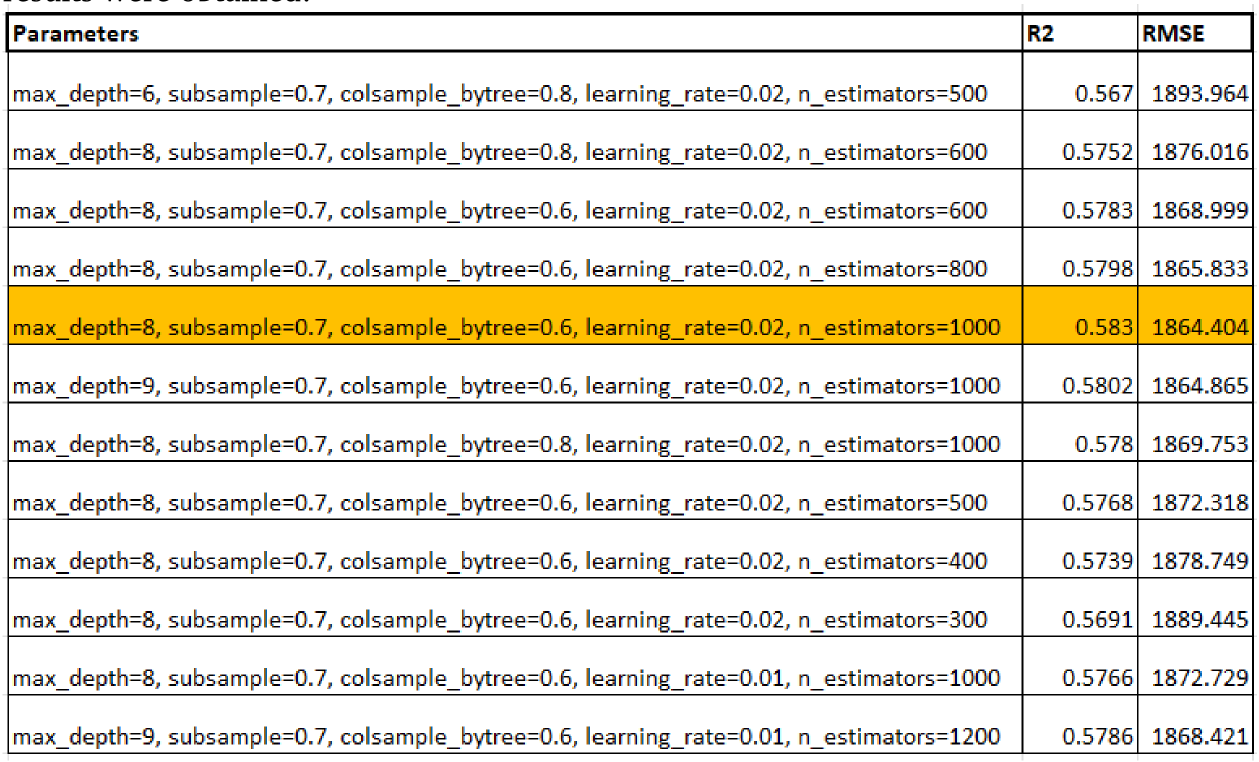
r^2 obtained through these parameters was 0.5655

3.7.2 Foward Feature Selection on Gradient Boosting

Forward feature selection was applied along with Gradient Boost, of which the code is in Appendix I. The value of K was set as 40. However this gave a lower r^2 (0.57).

3.8 XG Boosting Regressor

XGBoost was applied on `df_train_L_1` with different parameters. The following results were obtained:



Parameters	R2	RMSE
max_depth=6, subsample=0.7, colsample_bytree=0.8, learning_rate=0.02, n_estimators=500	0.567	1893.964
max_depth=8, subsample=0.7, colsample_bytree=0.8, learning_rate=0.02, n_estimators=600	0.5752	1876.016
max_depth=8, subsample=0.7, colsample_bytree=0.6, learning_rate=0.02, n_estimators=600	0.5783	1868.999
max_depth=8, subsample=0.7, colsample_bytree=0.6, learning_rate=0.02, n_estimators=800	0.5798	1865.833
max_depth=8, subsample=0.7, colsample_bytree=0.6, learning_rate=0.02, n_estimators=1000	0.583	1864.404
max_depth=9, subsample=0.7, colsample_bytree=0.6, learning_rate=0.02, n_estimators=1000	0.5802	1864.865
max_depth=8, subsample=0.7, colsample_bytree=0.8, learning_rate=0.02, n_estimators=1000	0.578	1869.753
max_depth=8, subsample=0.7, colsample_bytree=0.6, learning_rate=0.02, n_estimators=500	0.5768	1872.318
max_depth=8, subsample=0.7, colsample_bytree=0.6, learning_rate=0.02, n_estimators=400	0.5739	1878.749
max_depth=8, subsample=0.7, colsample_bytree=0.6, learning_rate=0.02, n_estimators=300	0.5691	1889.445
max_depth=8, subsample=0.7, colsample_bytree=0.6, learning_rate=0.01, n_estimators=1000	0.5766	1872.729
max_depth=9, subsample=0.7, colsample_bytree=0.6, learning_rate=0.01, n_estimators=1200	0.5786	1868.421

The highlighted row shows the parameters which gave the best results on kaggle ($RMSE = 1911.87712$). XGBoost with these parameters was also applied on the entire data set which gave $r^2 = 0.5775$ and RMSE = 1870.909, which shows that reducing features on this data set has little effect on the results of XGBoost in this case.

4 Conclusion

Initially the test and training files were concatenated and On-Hot Encoding of top features were done with a threshold of 100, and then seperated again into training and test files. Then, multiple data sets were created out of the original training file on number of unique features, feature reduction through Lasso, and p values. Multiple models were applied on these data sets by varying parameters. The data set which gave the best result was `df_train_L_1`, which was obtained through feature reduction through Lasso with $\alpha = 1$. The best result was achieved through XGBoost with and RMSE of 1911.877 on Kaggle.

Appendix

A Dropping columns based on number of unique categories.

```
df_with_grtr_thn_50_cats_dropped = df_all_3

# Select only the categorical columns in df_with_grtr_thn_20_cats_dropped
categorical_cols_df_with_grtr_thn_50_cats_dropped =
    df_with_grtr_thn_50_cats_dropped.select_dtypes(include=['object'])

for i in categorical_cols_df_with_grtr_thn_50_cats_dropped:
    if df_with_grtr_thn_50_cats_dropped[i].nunique()>=50:
        df_with_grtr_thn_50_cats_dropped.drop(i, axis=1, inplace=True)

categorical_cols_df_with_grtr_thn_50_cats_dropped =
    df_with_grtr_thn_50_cats_dropped.select_dtypes(include=['object'])

unique_cats_in_each_col_in_df_with_grtr_thn_50_cats_dropped =
    categorical_cols_df_with_grtr_thn_50_cats_dropped.nunique().sort_values(ascending=False)
unique_cats_in_each_col_in_df_with_grtr_thn_50_cats_dropped
```

B One-Hot Encoding Top Features

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Define the threshold
threshold = 1000

# Identify categorical columns
cat_columns = []
for col in df_all.columns:
    if df_all[col].dtype == 'object':
        cat_columns.append(col)

# Create a list to store the top categories for each categorical column
top_categories = []
```

```
# Loop through each categorical column and identify the top categories
for col in cat_columns:
    vc = df_all[col].value_counts()
    top_cats = list(vc[vc >= threshold].index)
    top_categories.append({col: top_cats})

# Create a list of column transformers for OneHotEncoder
column_transformers = []
for col in top_categories:
    col_name = list(col.keys())[0]
    top_cats = list(col.values())[0]
    transformer = (col_name, OneHotEncoder(categories=[top_cats],
        sparse=False, handle_unknown='ignore', drop='first'), [col_name])
    column_transformers.append(transformer)

# Create the column transformer object
from sklearn.compose import ColumnTransformer
preprocessor = ColumnTransformer(transformers=column_transformers)

# Fit and transform the preprocessor on the data
X = preprocessor.fit_transform(df_all)


col_names = []
for transformer in column_transformers:
    # Get the OneHotEncoder transformer
    onehot = transformer[1]
    # Get the input column name
    col_name = transformer[0]
    # Get the top categories for the input column
    top_cats = transformer[1].categories[0][1:]
    # Create column names by combining the input column name with the top
        categories
    col_names += [f"{col_name}_{cat}" for cat in top_cats]


df_X = pd.DataFrame(X, columns=col_names)

# Print the dataframe
print(df_X)
```

C Feature Reduction through P values

```
model = sm.OLS(y, X).fit()
# model.summary()

X_p_5 = X.copy()

for col in X_p_5.columns:
    if model.pvalues[col] >0.05:
        X_p_5.pop(col)
```

D Feature Reduction Through Lasso

```
X = df_train.loc[:, df_train.columns != 'loss']
y = df_train['loss']

# Create a Lasso model
lasso = Lasso(alpha=1)

# Fit the Lasso model on the training data
lasso.fit(X, y)

# Identify the important features
important_features = X.columns[(np.abs(lasso.coef_) > 1e-10)]

# Filter the original dataset to keep only the important features
X_filtered = X[important_features]
df_test_L = df_test[important_features]
df_train_L = pd.concat([X_filtered, df_train['loss']], axis=1)
```

E Plotting α vs RMSE for Ridge Regression

```
# Create a list of alpha values to try
alphas = [0.01, 0.1, 1, 10, 100]

# Initialize a list to store RMSE values for each alpha
```

```
rmse_values = []

# Train a Ridge regression model for each alpha and calculate RMSE
for alpha in alphas:
    # Create Ridge regression object
    ridge_reg = Ridge(alpha=alpha)

    # Train Ridge regression model
    ridge_reg.fit(X_train, y_train)

    # Make predictions on test set
    y_pred = ridge_reg.predict(X_test)

    # Calculate root mean squared error (RMSE)
    rmse = mean_squared_error(y_test, y_pred, squared=False)

    # Append RMSE value to list
    rmse_values.append(rmse)

# Plot alpha values vs RMSE
plt.plot(alphas, rmse_values)
plt.xlabel('Alpha')
plt.ylabel('RMSE')
plt.title('Alpha vs RMSE')
plt.show()
```

F HeatMap for Elastic Net Regression

```
# set up a range of alpha and rho values to test
alphas = [10, 50, 100]
rhos = [0.75, 0.8, 0.85, 0.9, 0.95, 1]


# initialize an empty array to store the RMSE values
rmse_values = np.zeros((len(alphas), len(rhos)))

# loop over the alpha and rho values to fit the model and calculate the RMSE
for i, alpha in enumerate(alphas):
    for j, rho in enumerate(rhos):
        # fit the model
        model = ElasticNet(alpha=alpha, l1_ratio=rho)
        model.fit(X_train, y_train)
        # make predictions on the test set
        y_pred = model.predict(X_test)
        # calculate the RMSE
        rmse = np.sqrt(mean_squared_error(y_test, y_pred))
        rmse_values[i, j] = rmse

# create a heatmap of the RMSE values
fig, ax = plt.subplots()
im = ax.imshow(rmse_values, cmap='coolwarm', extent=[0.1, 1, 1000, 0.001],
    aspect='auto')
ax.set_xlabel('rho')
# Get the y-axis tick labels
ylabels = [tick.get_text() for tick in ax.get_yticklabels()]

# Format the y-axis tick labels with the exact alpha values
ax.set_yticklabels(['{:.3f}'.format(float(label)) for label in ylabels])
ax.invert_yaxis()

# add a color scale legend to the plot
cbar = ax.figure.colorbar(im, ax=ax)
cbar.set_label('RMSE')

plt.show()
```

G Randomized Search for Gradient Boosting

```
# Define parameter grid for randomized search
params = {
    'max_depth': [3, 5, 7],
    'max_features': ['sqrt', 'log2', None],
    'min_samples_split': [2, 4, 8],
    'n_estimators': [100, 300, 500],
}

# Create GradientBoostingRegressor model
regGB = GradientBoostingRegressor(random_state=42)

# Create randomized search object
rs = RandomizedSearchCV(
    estimator=regGB,
    param_distributions=params,
    n_iter=20,
    cv=5,
    random_state=42,
    n_jobs=-1,
    scoring='r2'
)

# Fit randomized search to training data
rs.fit(X_train, y_train)

# Print best parameters and score
print('Best score:', rs.best_score_)
print('Best parameters:', rs.best_params_)
```

H K values vs RMSE

```
# initialize lists to store k values and corresponding RMSE values
k_values = [1,3,5,8,10,15,20, 30, 50, 80, 100, 150]
rmse_values = []

# loop through different k values and fit KNN regressor on the training data
for k in k_values:
    knn = KNeighborsRegressor(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    rmse_values.append(rmse)

# plot the results
import matplotlib.pyplot as plt
plt.plot(k_values, rmse_values)
plt.xlabel('K')
plt.ylabel('RMSE')
plt.title('KNN Regression Performance')
plt.show()
```

I Forward Feature Selection on Gradient Boost

```
# Define the number of features to select
k = 40

# Initialize the set of selected features
selected_features = []

# Define the XGBoost model
gb_model = gb.GBRegressor(objective='reg:squarederror', random_state=42)

# Iterate k times to select k features
for i in range(k):

    # Initialize the best feature and the best score
    best_feature = None
    best_score = -np.inf

    # Iterate over the remaining features to find the best feature
    for feature in X_train.columns:

        # Skip already selected features
        if feature in selected_features:
            continue

        # Add the current feature to the selected features
        candidate_features = selected_features + [feature]

        # Train the GBoost model on the candidate features
        gb_model.fit(X_train[candidate_features], y_train)

        # Evaluate the XGBoost model on the test set
        y_pred = gb_model.predict(X_test[candidate_features])
        score = r2_score(y_test, y_pred)

        # Check if the current feature improves the score
        if score > best_score:
            best_feature = feature
            best_score = score

    # Add the best feature to the selected features
    selected_features.append(best_feature)

    # Print the selected features and the score
    print(f'Selected feature {i+1}: {best_feature} (R2 score:
        {best_score:.4f})')
```