Challenger 2
Introduction to Machine Learning

Zuha Aqib 26106

Table of contents

Table of contents

Google Sheets link

https://docs.google.com/spreadsheets/d/1unhRuxaMYy5XK87DKrpjBpaWKnct0YIMMf9FlweNMMI/edit?usp=sharing

GitHub repository link

https://github.com/z-aqib/machine-learning-algorithms.git


01
About the
Challenge

157 entries

17 on the first day, and 20
every day to the end

Lowest RMSE: 12437836.73841
Achieved by RandomForestRegressor
21st Position on Leaderboard

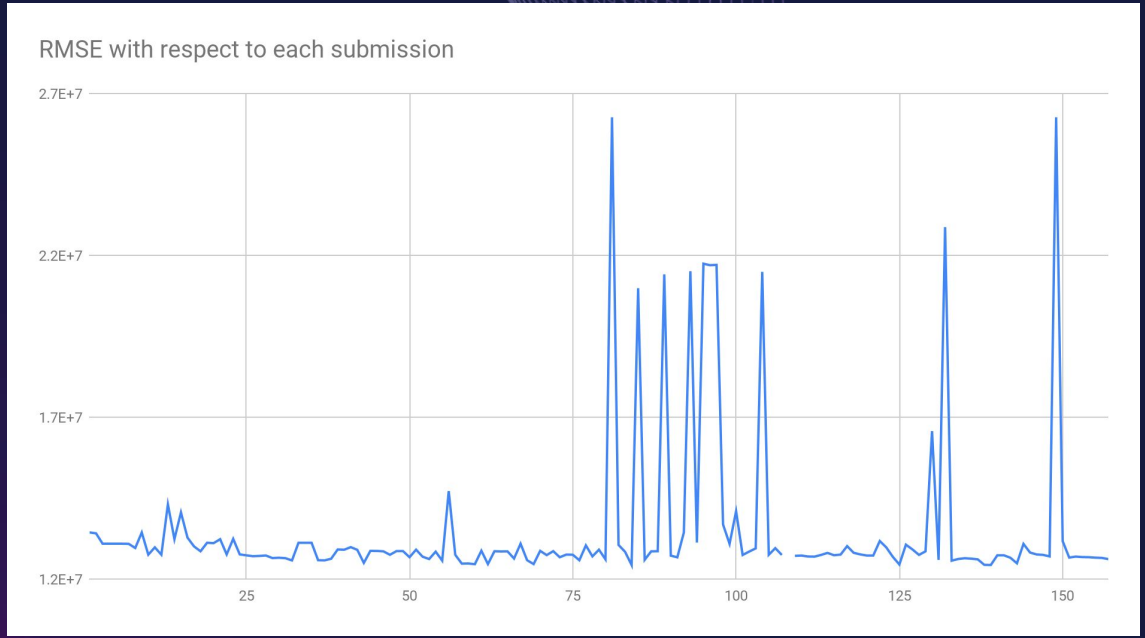
21	Zuha Aqib		12437836.73841	157	2d

RMSE Analysis

- The majority of my entries lie between 12.5M and 13M.
- There were some exceptional entries that shot up to 20M.
- And there were some excellent entries that went lower than 12.5M.

(M here is denoted as 10 with the power of 6)

kaggle score	count
< 12,500,000	10
12,500,000 - 13,000,000	102
13,000,000 - 14,000,000	30
14,000,000 - 20,000,000	5
> 20,000,000	10



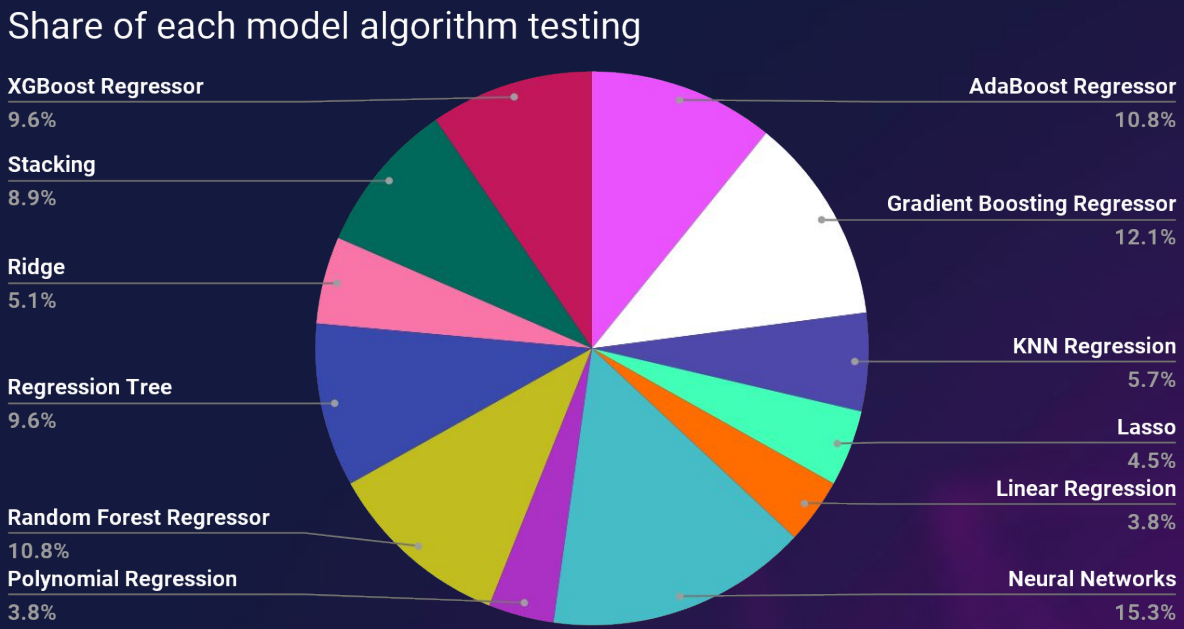
	Count
AdaBoost Regressor	17
Gradient Boosting Regressor	19
KNN Regression	9
Lasso	7
Linear Regression	6
Neural Networks	24
Polynomial Regression	6
Random Forest Regressor	17
Regression Tree	15
Ridge	8
Stacking	14
XGBoost Regressor	15
total	**157**

Number of entries for each algorithm

Daily Accuracy Analysis

Day	Highest Accuracy
Day 1: 25th	12756207.92
Day 2: 26th	12578391.07
Day 3: 27th	12501962.35
Day 4: 28th	12462904.72
Day 5: 29th	12437836.74
Day 6: 30th	12696306.98
Day 7: 1st	12452109.13
Day 8: 2nd	12438069.98

Division of Entries

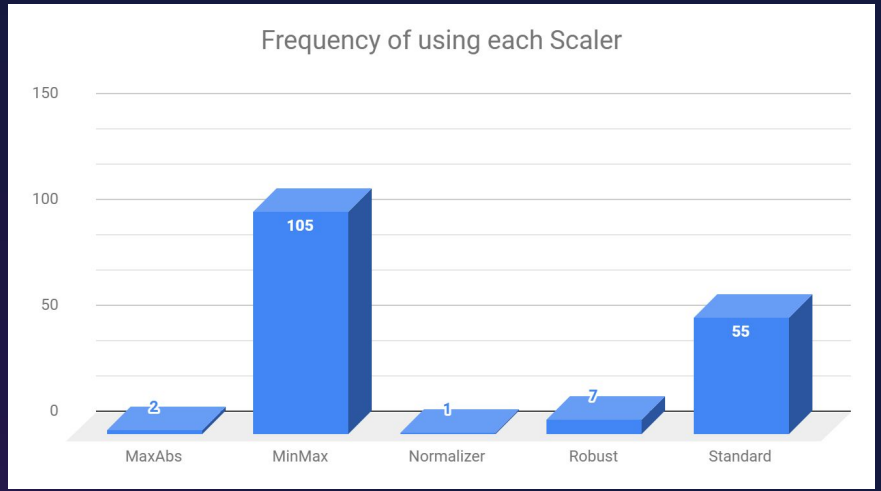


Analyzing Scalers

There are 5 types of scalers:

- Min Max: most used, gave a smooth PCA curve, gave good results.
- Standard: nice results, sometimes better than MinMax. The PCA curve was a bit rocky and not too smooth.
- Normalizer: used only once, did not give a good result.
- Robust: gave okay results, not as good as MinMax or Standard.
- Max Abs: gave the same result as MinMax.

	count
MaxAbs	2
MinMax	103
Normalizer	1
Robust	7
Standard	57



```python
# count number of null values in data
null_values = train_data.isnull().sum()
print("No. of null values for each column: \n", null_values)

total_nulls = train_data.isnull().sum().sum()
print("Total null values in train_data: ", total_nulls)

total_nulls_2 = test_data.isnull().sum().sum()
print("Total null values in test_data: ", total_nulls_2)
```

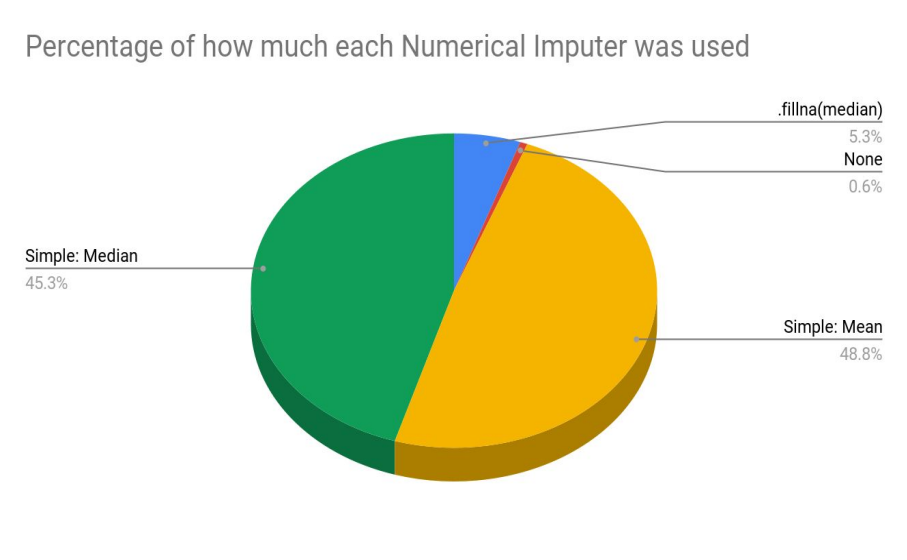
[10] ✓ 0.5s

```
No. of null values for each column:
 full_sq               0
life_sq                0
floor                  0
product_type           0
sub_area               0
                      ..
mosque_count_5000      0
leisure_count_5000     0
sport_count_5000       0
market_count_5000      0
price_doc              0
Length: 272, dtype: int64
Total null values in train_data:  0
Total null values in test_data:  0
```

Imputation

There are no null/empty values in both the train and test data sets.

	count
.fillna(median)	9
None	1
Simple: Mean	83
Simple: Median	77



Analyzing Numerical Imputers

There are 4 types of numerical imputation:

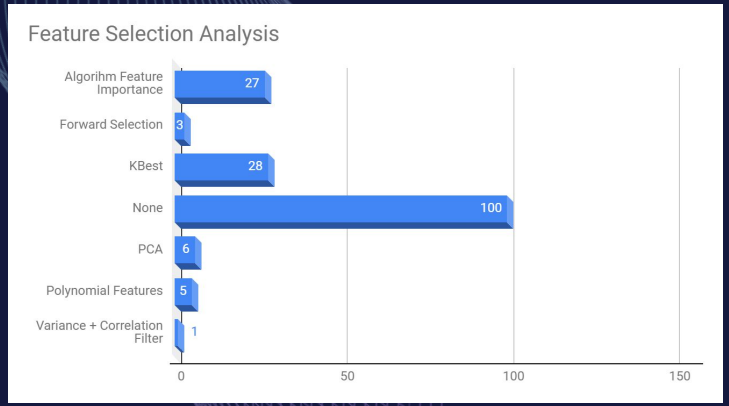
(There may be more, but I have tested only these; there were NO missing values; hence any kind of imputation does not matter.)

- Simple('mean') & Simple('median'): these two were used alternatively; both gave the same result.
- None: only done once, in Neural Networks, no major difference noticed.
- .fillna(median): this fills all null values with the median of the column; it is the same as the simple (median) imputer.

Analyzing Feature Selection

There were 7 ways to perform feature selection:

- Forward Selection: only done 3 times (twice on Linear Regression and once on Regression Tree), took over 3 hours on LR and 12 hours on RT, even after taking a sample of 10% of data
- Backward Selection: tried on LR but even overnight did not have progress to select one feature. Program was terminated.
- KBest: worked very well with KNN and XGB but performed very badly with LR, Ridge, Lasso, and RT. It was used even in stacking.
- PCA & Variance-Correlation Filter: When PCA alone was applied, it gave an error that all values captured 100% variance. This was when we had applied one-hot or getdummies encoding. To cater this, we performed a variance filter, reduced columns from 2000 to 200-400 columns, and then performed PCA of 95% variance capturing. There was not much difference in values.
- Polynomial Features: There is a method in scikit-learn that was used; data was sampled to 1%, a variance filter was applied, and then polynomial features of degree 2 were applied, and only 1% of train_data was used as features were exponential. Very bad accuracy.
- Algorithm feature importance: many were used, some improved, and some did not.



	count
Algorihm Feature Importance	27
Forward Selection	3
KBest	28
None	100
PCA	6
Polynomial Features	5
Variance + Correlation Filter	1


02
Linear
Regression

Linear Regression

Total: 6 entries
Best RMSE: 13442363.54 (case 92)

- Using normal linear regression with no feature selection gave RMSE in 130M.
- Forward selection took way too much time and did not improve the score; rather, it made it worse. Lower features made it worse. Backward selection did not work even overnight. For forward, we had to take a 10% sample of data and perform feature selection on that.
- KBest completely ruined the RMSE
- Pipeline or no pipeline, accuracy remained the same
- For full features, it was okay, but when features were reduced, the more the number of features, the worse; however, too few features were also bad.

Case Number	Model Name	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	train test split	Pipeline	Model params	Sample taken	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
1	Linear Regression	None	271	Simple: Median	Standard	Simple: most_frequent	OneHot	X, Y, test_size=0.3, random_state=2	Pipeline	LinearRegression()	-	191699960168873	13845575.47	6682687.71	0.6611981838	13443448.17
85	Linear Regression	KBest	200	Simple: Median	MinMax	Simple: most_frequent	OneHot	X, Y, test_size=0.3, random_state=2	Pipeline	LinearRegression()	-	445472403693967	21106217.18	12611617.42	0.07577281727	20986342.15
89	Linear Regression	KBest	100	Simple: Median	MinMax	Simple: most_frequent	OneHot	X, Y, test_size=0.3, random_state=2	Pipeline	LinearRegression()	-	464005048513003	21540776.41	13045122.22	0.03604177888	21413824.17
92	Linear Regression	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	X, Y, test_size=0.3, random_state=2	Regular	LinearRegression()	-	191656155099850	13843993.47	6681690.71	0.6612014261	13442363.54
98	Linear Regression	Forward Selection	10	Simple: Median	MinMax	Simple: most_frequent	OneHot	X, Y, test_size=0.3, random_state=2	Regular	LinearRegression()	for forward selection: train_data.sample(frac=0.1)	192403144980823	13870946.07	7041594.36	0.6019278395	13689504.36
100	Linear Regression	Forward Selection	5	Simple: Mean	Standard	Simple: most_frequent	Dummy	X, Y, test_size=0.3, random_state=2	Regular	LinearRegression()	forward selection sample: sample_train = train_data.sample(frac=0.1)	204458095850321	14298884.43	7365368.83	0.5781924234	14101310.75

02a
Ridge

Total: 12 entries
Best RMSE: 13091172.07 (case 7)

- Only simple (mean) and min-max scalers and one-hot and simple (most frequent) were used.
- The best alpha value after the grid was 100 (0.01 to 10000 was checked).
- Lsqr was the best solver.
- Max_iterations did not make a difference; whether they were 1000, 50, or 100, the model was the same, and therefore the file did not change.
- K-best feature selection completely ruined the RMSE.
- RidgeCV did not have much difference from ridge alone.
- Lower tol value helped us achieve a good RMSE.

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Grid	Grid Best	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
Ridge	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Ridge(alpha=0.5)	-	-	189920563566693	13781166.99	6672565.38	0.6610617032	13414895.67
Ridge	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Ridge()	{ 'model__alpha': [0.01, 0.1, 1.0, 10.0, 100.0] }	{'model__alpha': 100.0}	178446807239989	13358398.38	6620569.43	0.6429330719	13095760.56
Ridge	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Ridge()	{ 'model__alpha': [98.0, 99.0, 100.0, 101.0, 102.0, 105.0] }	{'model__alpha': 99.0}	178445219796271	13358338.96	6620198.39	0.6429840577	13095804.98
Ridge	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Ridge()	{ "model__alpha": [100, 500, 1000, 5000, 10000] }	{'model__alpha': 100}	-	-	-	-	not submitted as same as case 4
Ridge	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Ridge(alpha=100)	{ 'model__solver': ['auto', 'svd', 'lsqr', 'sparse_cg'] }	{'model__solver': 'lsqr'}	178422961193358	13357505.8	6620163.56	0.6428676657	13095300.22
Ridge	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Ridge(alpha=100, solver='lsqr')	{ 'model__fit_intercept': [True, False], 'model__positive': [True, False] }	{'model__fit_intercept': True, 'model__positive': False}	-	-	-	-	not submitted as same as default values; same as case 5.2
Ridge	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Ridge(alpha=100, solver='lsqr')	{ 'model__max_iter': [1000, 5000, 10000] }	{'model__max_iter': 1000}	178422961193358	13357505.8	6620163.56	0.6428676657	13095300.22
Ridge	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Ridge(alpha=100, solver='lsqr')	{ 'model__max_iter': [100, 200, 300, 500, 900, 1000] }	{'model__max_iter': 100}	178422961193358	13357505.8	6620163.56	0.6428676657	FILE DID NOT CHANGE. ENTRY IS NOT SUBMITTED.
Ridge	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Ridge(alpha=100, solver='lsqr')	{ 'model__max_iter': [1, 10, 20, 50, 90, 100] }	{'model__max_iter': 50}	178422961193358	13357505.8	6620163.56	0.6428676657	FILE DID NOT CHANGE. ENTRY WAS NOT SUBMITTED.
Ridge	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Ridge(alpha=100, solver='lsqr')	{ 'model__tol': [1e-1, 1e-2, 1e-3, 1e-4, 1e-5] }	{'model__tol': 0.001}	178399494100404	13356627.35	6627800.65	0.6409230973	13091172.07
Ridge	Day 4: 28th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RidgeCV(alphas=[0.1, 1.0, 10.0, 100.0], fit_intercept=True, cv=5)	-	-	178446807239989	13358398.38	6620569.43	0.642931656	13095757.52
Ridge	Day 5: 29th	KBest	200	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	Ridge(alpha=100, solver='lsqr', tol=0.001)	-	-	470769261792904	21697217.84	13284759.95	0.02420298818	21511049.8

02b
Lasso

Total: 8 entries
Best RMSE: 13124597.22 (case 34)

- Overall, the model was slow and took hours and hours in grid search—took samples for it. Simple alone running took an hour.
- Same scalers, imputers, and encoders were used in all cases.
- Kbest ruined the RMSE
- Grid search proved that higher alpha resulted in better results; however, upon Kaggle RMSE, too high an alpha was bad; the neutral value was 10000.
- Best selection criteria was ‘random.’
- No major effect on changing max_iterations
- No major effect on changing tol value

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Grid	Grid Best	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
Lasso	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Lasso()	-	-	191668945999741	13844455.42	6682058.36	0.6611991036	13442390.39
Lasso	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Lasso()	{ 'model__alpha': [100, 1000, 10000] }	{'model__alpha': 10000}	178437965939460	13358067.45	6738128.97	0.6336621905	13124664.68
Lasso	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	Lasso()	{ 'model__alpha': [10000, 20000, 50000, 100000] }	{'model__alpha': 50000}	181387236371446	13468007.88	6947525.65	0.6260249179	13245107.6
Lasso	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	Lasso(alpha=1000 0)	{ 'max_iter': [1000, 5000, 10000] }	{'max_iter': 5000}	178445511124957	13358349.87	6738142.7	0.6336565805	13124809.79
Lasso	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	Lasso(alpha=1000 0)	{ "selection": ["cyclic", "random"] }	{'selection': 'random'}	178442238660973	13358227.38	6738059.58	0.6336676025	13124597.22
Lasso	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	Lasso(alpha=1000 0, selection='random ')	param_grid = { "tol": [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.05, 0.1, 1] }	{'tol': 0.001}	178441627053489	13358204.48	6738000.85	0.6336635394	13124614.49
Lasso	Day 5: 29th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	LassoCV(alphas=[0.1, 0.5, 1.0, 5.0, 10.0], fit_intercept=True, max_iter=1000, cv=5)	-	-	-	-	-	-	too long, 6h on traindata and no update
Lasso	Day 5: 29th	KBest	200	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	Lasso(alpha=1000 0, selection='random ')	-	-	478331947015584	21870801.24	13415352.03	0.0008033744264	21743808.64


03
Polynomial
Regression

- Polynomial features. Didnt work with full data or full features. Had to do variance filter, than sample data, then perform polynomial features
- After that, model was trained on only 1% of the data
- Very bad with linear regression
- Turned a 12.43M random forest RMSE to 130M rmse
- Very fast in general
- RMSE improved in RF when estimators increased
- DT gave bad RMSE so stacked DT was used, still very bad
- KNN was also bad
- Same scalers imputers encoders were used in all cases

Case Number	Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Sample taken	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
108	Polynomial Regression	Day 6: 30th	Polynomial Features	-	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	LinearRegression()	poly = PolynomialFeatures(degree=2, include_bias=False) selector = VarianceThreshold(threshold=0.9)	sample_train = train_data.sample(frac=0.01)	4.9772E+24	2230964430069	783351980734	0.9999837164	5499265940162064
126	Polynomial Regression	Day 7: 1st	Polynomial Features	-	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	RandomForestRegressor(max_depth=39, n_estimators=400, max_features='sqrt', verbose=2, n_jobs=-1)	poly = PolynomialFeatures(degree=2, include_bias=False) selector = VarianceThreshold(threshold=0.9)	X Y is 3% sample of train	189663005652428	13771819.26	6575550.63	0.9502352627	13065940.62
127	Polynomial Regression	Day 7: 1st	Polynomial Features	-	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	RandomForestRegressor(max_depth=35, n_estimators=500, max_features='sqrt', verbose=2, n_jobs=-1, min_samples_split=7)	poly = PolynomialFeatures(degree=2, include_bias=False) selector = VarianceThreshold(threshold=0.9)	sample_train = train_data.sample(frac=0.01)	151440466941947	12306115.02	5501600.22	0.9302153975	12918342.46
130	Polynomial Regression	Day 7: 1st	Polynomial Features	-	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	StackingRegressor(estimators=[('model4', DecisionTreeRegressor(max_depth=11)), final_estimator=DecisionTreeRegressor(max_depth=2), passthrough=False, n_jobs=-1, verbose=2)	poly = PolynomialFeatures(degree=2, include_bias=False) sample_train = train_data.sample(frac=0.005) selector = VarianceThreshold(threshold=0.999999) 123 columns	sample_train = train_data.sample(frac=0.005)	399497786442319	19987440.72	10181137.66	0.5160564308	16973091.4
132	Polynomial Regression	Day 7: 1st	Polynomial Features	-	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(n_neighbors=50, algorithm='auto', leaf_size=70, p=2, metric='minkowski', n_jobs=-1, weights='uniform')	poly = PolynomialFeatures(degree=2, include_bias=False) sample_train = train_data.sample(frac=0.1) selector = VarianceThreshold(threshold=0.999)	sample_train = train_data.sample(frac=0.1)	545826372861946	2336292732	9904841.71	-0.0671444937	22873886.79
150	Polynomial Regression	Day 8: 2nd	None	271	Simple: Median	Standard	Simple: most_frequent	OneHot	Pipeline	Ridge(alpha=1000)	-	-	-	-	-	-	13175268.83

04

KNN Regression

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
KNN Regression	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	KNeighborsRegressor(n_neighbors=5)	-	-	-	-	-	crashed after 600mins
KNN Regression	Day 3: 27th	KBest	100	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(n_neighbors=35, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='euclidean', n_jobs=-1)	-	171533481002074	13097079.1	5987080.87	0.6652201843	12917545.46
KNN Regression	Day 3: 27th	KBest	200	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(n_neighbors=35, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='euclidean', n_jobs=-1)	-	172838676912414	13146812.42	6037527.93	0.6647465045	12910355.25
KNN Regression	Day 3: 27th	KBest	50	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(n_neighbors=35, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='euclidean', n_jobs=-1)	-	175478062684853	13246813.3	6048657.72	0.6598568005	12989247.62
KNN Regression	Day 3: 27th	KBest	100	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(n_neighbors=50, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='euclidean', n_jobs=-1)	-	171632824649793	13100871.14	6059310.66	0.6596135202	12905957.5
KNN Regression	Day 3: 27th	KBest	100	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='euclidean', n_jobs=-1)	-	-	-	-	-	not submitted as same as case 42
KNN Regression	Day 3: 27th	KBest	200	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='euclidean', n_jobs=-1)	-	171728848815368	13104535.43	6152053.05	0.6565880782	12873928.69

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
KNN Regression	Day 3: 27th	KBest	200	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='euclidean', n_jobs=-1)	-	171750939833772	13105378.28	6145186.69	0.6569848212	12871457.07
KNN Regression	Day 3: 27th	KBest	200	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(weights='uniform', algorithm='auto', leaf_size=30, p=2, n_neighbours=67, n_jobs=-1)	-	-	-	-	-	file didnt change, not submitted
KNN Regression	Day 3: 27th	KBest	200	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(n_neighbors=67, algorithm='auto', leaf_size=30, p=2, metric='euclidean', n_jobs=-1)	-	168584198858619	12983997.8	5829392.19	0.9999971891	12752650.1
KNN Regression	Day 3: 27th	KBest	200	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(n_neighbors=67, weights='distance', algorithm='auto', p=2, metric='euclidean', n_jobs=-1)	-	168584198858619	12983997.8	5829392.19	0.9999971891	not submitted. file is same.
KNN Regression	Day 3: 27th	KBest	200	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(n_neighbors=67, weights='distance', leaf_size=30, p=2, metric='euclidean', n_jobs=-1)	-	168584204034058	12983998	5829397.78	0.9999971968	12752648.99
KNN Regression	Day 6: 30th	PCA	267, 95%	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	KNeighborsRegressor(n_neighbors=67, weights='distance', leaf_size=30, p=2, metric='euclidean', = n_jobs=-1)	variance filter: selector = VarianceThreshold(threshold=0.01) 273 columns pca = PCA(n_components=0.95) 267 columns	167815929949379	12954378.79	5813474.85	0.9999973415	12759323.06

Best RMSE:
Best RMSE: 12,953,978.79 (Case 116)
Key Findings:
Feature Selection:
KBest:

Increasing KBest from 50 (Case 41) to 200 (Case 40) improved RMSE and generalization.
Best results were achieved with KBest=200 in most cases. Lower values removed too many critical features.
PCA (Case 116):

With 95% variance retention, PCA reduced features to 267 columns, slightly improving RMSE and stabilizing performance.
PCA was beneficial when combined with KNeighborsRegressor with robust hyperparameters.
Model Parameters:
n_neighbors:

Increasing n_neighbors initially from 5 to 35 (Case 39) improved results drastically.
Grid search found the optimal n_neighbors=67 (Case 45). Beyond this, no significant improvements.
Weights:

Switching from 'uniform' to 'distance' (Case 47) reduced RMSE and improved R^2. Weights='distance' consistently worked better for non-uniform feature distributions.
Metric:

Grid search confirmed 'euclidean' as the optimal distance metric (Case 46.1). Other metrics (e.g., 'manhattan') performed worse or provided no additional benefit.
Algorithm:

Changing algorithms (Case 57) to 'ball_tree' made no impact, as 'auto' was already optimizing effectively.
Leaf Size:

Grid search for leaf_size (Case 51.1) showed no meaningful change, confirming that default values were sufficient.
Scaling and Imputation:
MinMax Scaling:
Necessary to normalize feature ranges and stabilize performance.
Imputation (Simple Mean):
Standard mean imputation sufficed, with no observable difference between imputation techniques.
Observations:
Small Dataset (Case 25.1):

Crashed after 600 minutes due to high computational cost. Sampling was critical for high-dimensional data or feature selection.
Optimal Configuration:

Best RMSE achieved in Case 116: n_neighbors=67, weights='distance', metric='euclidean', PCA (95% variance), KBest=200, and MinMax scaling.
Balancing dimensionality reduction (via PCA) and feature importance (via KBest) was key to stabilizing results.

05 Regression Tree

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Grid	Grid Best	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
Regression Tree	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	DecisionTreeRegressor (max_depth=10, random_state=0)	-	-	174156211510543	13196825.81	5696000.7	0.6796795919	12963023.57
Regression Tree	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	DecisionTreeRegressor (random_state=0)	{ 'model__max_depth': [1, 2, 3, 4, 5, 10] }	{'model__max_depth': 5}	167977657854855	12960619.5	6016174.11	0.6562927031	12756207.92
Regression Tree	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	DecisionTreeRegressor (random_state=0)	{ 'model__max_depth': [5, 6, 7, 8, 9, 10] }	{'model__max_depth': 5}	167977657854855	12960619.5	6016174.11	0.6562927031	12756207.92
Regression Tree	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	DecisionTreeRegressor (random_state=0, max_depth=5)	{ 'model__max_features': ['auto', 'sqrt', 'log2'] }	{'model__max_features': 'sqrt'}	183853745621936	13559267.89	7056495.43	0.6293951358	13234262.49
Regression Tree	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	DecisionTreeRegressor (random_state=0, max_depth=5, max_features='sqrt')	{ 'model__criterion': ['squared_error', 'friedman_mse', 'absolute_error', 'poisson'] }	{'model__criterion': 'squared_error'}	183853745621936	13559267.89	7056495.43	0.6293951358	13234262.49
Regression Tree	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	DecisionTreeRegressor (random_state=0, max_depth=5)	{ 'model__criterion': ['squared_error', 'friedman_mse', 'absolute_error', 'poisson'] }	{'model__criterion': 'poisson'}	-	-	-	-	12765094.57
Regression Tree	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	DecisionTreeRegressor (random_state=0, max_depth=5, criterion='poisson')	{ 'min_samples_split': [10, 30, 50, 100, 200, 1000] }	{'min_samples_split': 10}	167260228624135	12932912.61	5984409.15	0.6550849218	12765094.57
Regression Tree	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	DecisionTreeRegressor (random_state=0, max_depth=5, criterion='poisson')	{ 'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] }	{'min_samples_split': 2}	-	-	-	-	same as default, not submitted, FILE DIDNT CHANGE
Regression Tree	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	DecisionTreeRegressor (random_state=0, max_depth=5, criterion='poisson')	{ 'splitter': ['best', 'random'] }	{'splitter': 'best'}	-	-	-	-	file didnt change default, didnt submit
Regression Tree	Day 5: 29th	KBest	200	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	DecisionTreeRegressor (random_state=0, max_depth=5, criterion='poisson')	-	-	477725537825579	21856933.4	13379169.34	0.005283516977	21696485.17
Regression Tree	Day 5: 29th	KBest	100	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	DecisionTreeRegressor (random_state=0, max_depth=5, criterion='poisson')	-	-	477658555118472	21855401.05	13385455.52	0.004962547274	21707456.2
Regression Tree	Day 6: 30th	PCA	205	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	DecisionTreeRegressor (random_state=0, max_depth=5, criterion='poisson')	-	-	177197379480826	13311550.6	710773.96	0.6363686	13089744.38
Regression Tree	Day 7: 1st	Forward Selection	10	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	DecisionTreeRegressor (random_state=0, max_depth=5, criterion='poisson')	-	-	169740577777018	13028452.62	6097042.84	0.6500690353	12814602.6
Regression Tree	Day 7: 1st	Variance + Correlation Filter	419	Simple: Mean	MaxAbs	Simple: most_frequent	Dummy	Regular	DecisionTreeRegressor (random_state=0, max_depth=5, criterion='poisson')	-	-	167281092333096	12933719.2	5984727.18	0.6550849218	12765094.57
Regression Tree	Day 7: 1st	Algorihm Feature Importance	200	Simple: Mean	MaxAbs	Simple: most_frequent	Dummy	Regular	DecisionTreeRegressor (random_state=0, max_depth=5, criterion='poisson')	-	-	167260228624135	12932912.61	5984409.15	0.6555538166	12730638.82
Regression Tree	Day 7: 1st	Algorihm Feature Importance	100	Simple: Mean	Robust	Simple: most_frequent	Dummy	Regular	DecisionTreeRegressor (random_state=0, max_depth=5, criterion='poisson')	-	-	167257164304757	12932794.14	5984224.86	0.6555538166	12730638.82
Regression Tree	Day 7: 1st	Algorihm Feature Importance	200	Simple: Mean	Normalizer	Simple: most_frequent	Dummy	Regular	DecisionTreeRegressor (random_state=0, max_depth=5, criterion='poisson')	-	-	179202537001242	13386655.18	6801823.75	0.6289503354	13178538.38

Total: 17 entries
BEST RMSE: 12730638.82 (case 120)

- max_depth: increasing depth beyond 5 led to higher variance and worse performance.
- Max_features: grid said ‘sqrt,’ but it was very bad.
- Min_samples_split: no change on the model upon changing values
- Splitter: changing splitter technique also had no change on the model
- criterion ('poisson'): squared_error has no impact, whereas poisson improved
- KBest: ruined
- Forward Selection: VERY, VERY LONG. no major difference
- Variance + Correlation Filter: fast. No major difference
- PCA: worsened RMSE at 95% variance
- Scaling (MinMax, Standard): MinMax scaling improved the performance of the decision tree, preventing overfitting by normalizing the range of features, while standard scaling showed a slight performance drop when not properly tuned.
- Normalizer: very bad
- Maxabs: good!
- Robust: same as maxabs and minmax
- Algo feature imp: no difference in model when the number of features changed. BUT OVERALL GREAT, slightly improved RMSE
- One-Hot Encoding: It improved the handling of categorical variables by enabling the model to use them as numerical inputs without introducing unnecessary bias or complexity.

06

Random Forest Regressor

Case umber	Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Grid	Grid Best	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
13	Random Forest Regressor	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_depth=6, max_features=4, min_samples_split=8, n_estimators=300)	-	-	210712625322840	14515943.83	8082056.71	0.564325367	143115675
15	Random Forest Regressor	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_depth=7, max_features=4, min_samples_split=8, n_estimators=300)	-	-	201350035576811	14189786.31	7820540.79	0.5795379082	140626113.8
16	Random Forest Regressor	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_depth=10, max_features=4, min_samples_split=8, n_estimators=300)	-	-	182064047829679	13493111.12	7126153.18	0.6266156941	13283359.82
17	Random Forest Regressor	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_depth=15, max_features=4, min_samples_split=8, n_estimators=300)	-	-	173977492829199	13190052.8	6625772.14	0.6527342752	13012520.36
18.1	Random Forest Regressor	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_features=4, min_samples_split=8, n_estimators=300)	{ 'model__max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] }	{'model__max_depth': 10}	-	-	-	-	did not submit as same as case 16
18.2	Random Forest Regressor	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_features=4, min_samples_split=8, n_estimators=300)	{ 'model__max_depth': [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20] }	{'model__max_depth': 20}	170485082730009	13056993.63	6315450.37	0.6802655875	12861372.96
29	Random Forest Regressor	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_features=4, min_samples_split=8, n_estimators=300)	{ 'model__max_depth': [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30] }	{'model__max_depth': 30}	164994167532686	12845005.55	5787791.82	0.7501690798	12652092.93
34.1	Random Forest Regressor	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_features=4, min_samples_split=8, n_estimators=300)	{ 'model__max_depth': [30, 31, 32, 33, 34, 35] }	{'model__max_depth': 35}	-	-	-	-	not submitted as will do further grid
36	Random Forest Regressor	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_features=4, min_samples_split=8,	{ 'model__max_depth': [35, 36, 37, 38, 39, 40] }	{'model__max_depth': 39}	163742544360962	12796192.57	5584219.52	0.7963392035	12585886.64

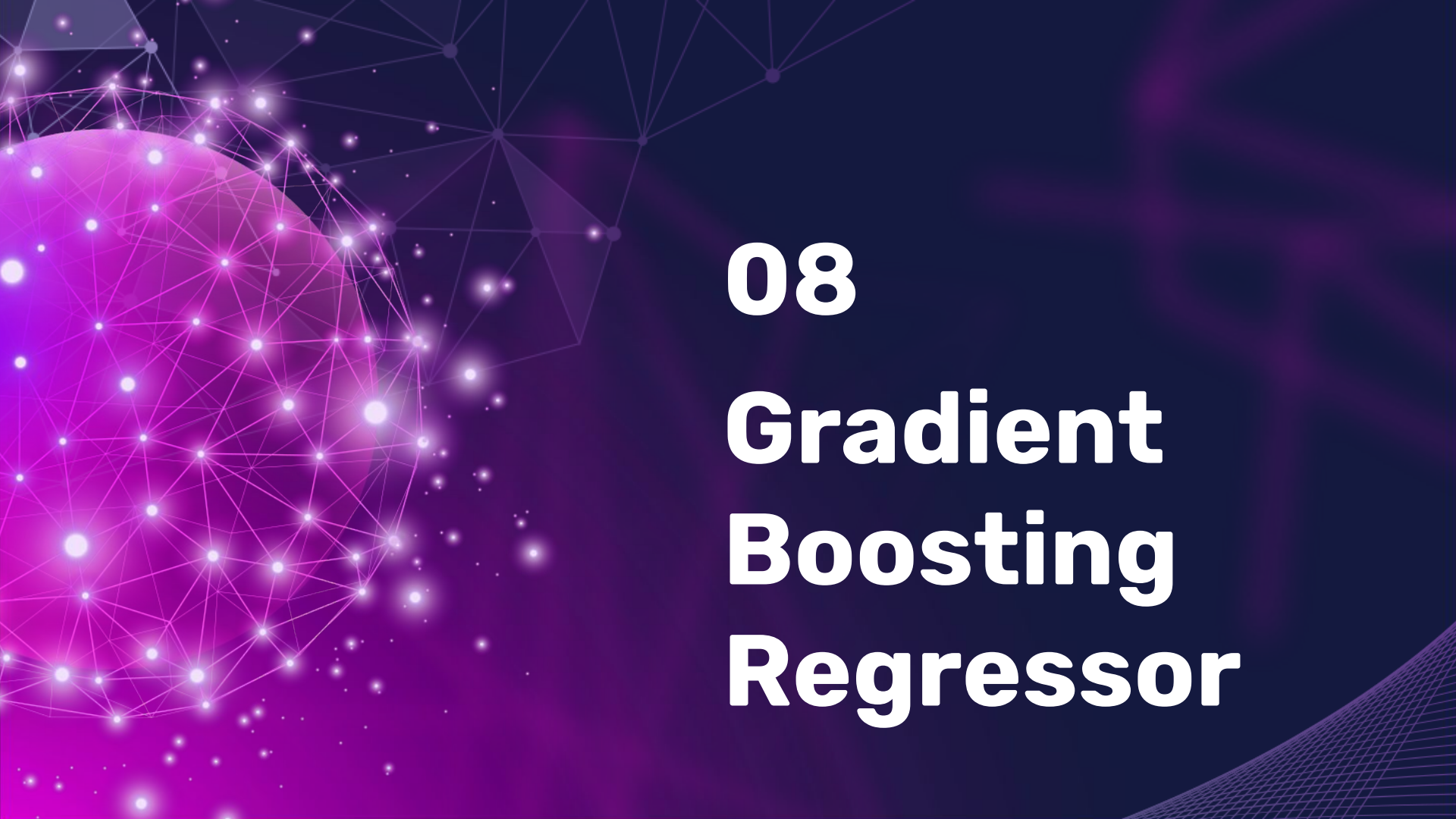
Case number	Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Grid	Grid Best	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
37	Random Forest Regressor	Day 2: 26th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_depth=39, max_features=4, min_samples_split=8, verbose=1, n_jobs=-1)	{'model__n_estimators': [50, 100, 200, 300, 400, 500] }	{'model__n_estimators': 400}	163464054696822	12785306.2	5571375.52	0.7958969399	12580644.37
43	Random Forest Regressor	Day 3: 27th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_depth=39, n_estimators=400, min_samples_split=8, verbose=1, n_jobs=-1)	{ 'model__max_features': [1, 'sqrt', 'log2'] }	{'model__max_features': 'sqrt'}	161225691081200	12697467.9	5243660.99	0.9150250875	12501962.35
58	Random Forest Regressor	Day 4: 28th	None	271	Simple: Mean	Standard	drop	Nothing	Pipeline	RandomForestRegressor(max_depth=31, n_estimators=1400, max_features='log2', min_samples_leaf=2, min_samples_split=3, bootstrap=True, verbose=2, n_jobs=-1)	-	-	160883805931493	12683998.03	5189756.85	0.9186423893	12479819.76
59	Random Forest Regressor	Day 4: 28th	None	271	Simple: Mean	MinMax	drop	Nothing	Regular	RandomForestRegressor(max_depth=31, n_estimators=1400, max_features='log2', min_samples_leaf=2, min_samples_split=3, bootstrap=True, verbose=2, n_jobs=-1)	-	-	160869442400043	12683431.81	5192165.99	0.9192710031	12484520.87
60	Random Forest Regressor	Day 4: 28th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	RandomForestRegressor(max_depth=39, n_estimators=400, max_features='sqrt', verbose=2, n_jobs=-1)	{ 'model__min_samples_split': [7, 8, 9, 10] }	{'model__min_samples_split': 7}	160274940665930	12659973.96	5047843.18	0.9334141991	12462904.72
62	Random Forest Regressor	Day 4: 28th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot		RandomForestRegressor(max_depth=39, n_estimators=400, max_features='sqrt', verbose=2, n_jobs=-1)	{ 'model__min_samples_split': [4, 5, 6, 7] }	{'model__min_samples_split': 6}	160523155422913	12669773.3	5041015.38	0.9384206599	12466009.75
84	Random Forest Regressor	Day 5: 29th	None	271	Simple: Median	Standard	nothing	Nothing	Pipeline	RandomForestRegressor(max_depth=36, n_estimators=1800, min_samples_split=2, min_samples_leaf=1, max_features=0.45, bootstrap=True,	-	-	159463537422322	12627887.29	4930593.56	0.9525272632	12437836.74

Case umber	Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Grid	Grid Best	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
										verbose=2, n_jobs=-1)							
125	Random Forest Regressor	Day 7: 1st	Algorihm Feature Importance	100	Simple: Median	Standard	nothing	Dummy	Regular	RandomForestRegressor(max_depth=36, n_estimators=1800, min_samples_split=2, min_samples_leaf=1, max_features=0.45, bootstrap=True, verbose=2, n_jobs=-1)	-	-	159625169771914	12634285.49	4947876.31	-	12452109.13
138	Random Forest Regressor	Day 8: 2nd	Algorihm Feature Importance	100	Simple: Median	Standard	nothing	Dummy	Regular	RandomForestRegressor(max_depth=36, n_estimators=1850, min_samples_split=2, min_samples_leaf=1, max_features=0.45, bootstrap=True, verbose=2, n_jobs=-1)	-	-	159635519508012	12634695.07	4946789.83	0.9523381486	12442789.3
139	Random Forest Regressor	Day 8: 2nd	None	271	Simple: Median	Standard	nothing	Nothing	Pipeline	RandomForestRegressor(max_depth=36, n_estimators=1800, min_samples_split=2, min_samples_leaf=1, max_features=0.5, bootstrap=True, verbose=2, n_jobs=-1)	-	-	-	-	-	0.9528415053	12438069.98


07
AdaBoost
Regressor

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Grid	Grid Best	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
AdaBoost Regressor	Day 3: 27th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(n_estimators=50,learning_rate=1.0)	-	-	-	170579321926767	13060601.9	6522100.2	0.6470543608	12859588.16
AdaBoost Regressor	Day 3: 27th	KBest	200	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(n_estimators=50,learning_rate=1.0)	-	-	-	170426388675063	13054745.83	6502154.98	0.6467723521	12866587.86
AdaBoost Regressor	Day 3: 27th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(n_estimators=100,learning_rate=1.0)	-	-	-	170844482328987	13070749.11	6559888.1	0.6467437679	12866229.08
AdaBoost Regressor	Day 3: 27th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(learning_rate=1.0)	-	{ 'n_estimators': [100, 150, 200, 250] }	{'n_estimators': 150}	170697294577910	13065117.47	6523891.08	0.6450971855	12911230.64
AdaBoost Regressor	Day 4: 28th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(n_estimators=500, learning_rate=1.0)	-	-	-	170937601415745	13074310.74	6569128.59	0.646705517	12879336.66
AdaBoost Regressor	Day 4: 28th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(learning_rate=0.5, n_estimators=1000)	-	-	-	170435288226422	13055086.68	6510890.39	0.6473047035	12862538.19
AdaBoost Regressor	Day 4: 28th	None	271	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(n_estimators=1200, learning_rate=0.5)	-	-	-	170738647766173	13066699.96	6582025.1	0.6474534045	12855862.28
AdaBoost Regressor	Day 4: 28th	None	271	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(n_estimators=1500, learning_rate=0.5)	-	-	-	170961650403776	13075230.41	6600065.8	0.6473999734	12858613.51
AdaBoost Regressor	Day 4: 28th	KBest	70	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(n_estimators=100, estimator = DecisionTreeRegressor (max_depth=10, splitter='best', criterion='poisson', min_samples_leaf=5, min_samples_split=3))	-	-	-	-	12774943.59	-	-	12584851.34
AdaBoost Regressor	Day 4: 28th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=3), n_estimators=100, learning_rate=0.5)	-	-	-	170489722078728	13057171.29	6522477	0.6465356142	12874165.01
AdaBoost Regressor	Day 4: 28th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(n_estimators=50, learning_rate=1.0)	-	-	-	170625153398977	13062356.35	6564950.21	0.6470294876	12865378.75

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
AdaBoost Regressor	Day 4: 28th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=2), n_estimators=200, learning_rate=0.1)	-	175228180857318	13237378.17	6941579.37	0.6367449795	13043233.81
AdaBoost Regressor	Day 5: 29th	None	271	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(estimator=DecisionTreeRegressor(max_depth=10), n_estimators=50, learning_rate=1.0)	-	164174620580665	12813064.45	5610232.6	0.686733332	12597515.56
AdaBoost Regressor	Day 5: 29th	None	271	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(estimator=DecisionTreeRegressor(max_depth=3), n_estimators=500, learning_rate=0.5)	-	170259012465408	13048333.7	6493107.63	0.6476490892	12860041.39
AdaBoost Regressor	Day 5: 29th	None	271	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	AdaBoostRegressor(estimator=RandomForestRegressor(n_estimators=10, max_depth=3), n_estimators=100, learning_rate=0.8)	-	169506153648570	13019452.89	6382041.83	0.6474311343	12862143.14
AdaBoost Regressor	Day 6: 30th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	VerboseAdaBoostRegressor(estimator=DecisionTreeRegressor(max_depth=4), n_estimators=1000, learning_rate=0.05)	class VerboseAdaBoostRegressor(AdaBoostRegressor): def _boost(self, iboost, X, y, sample_weight, random_state): print(f"Training estimator {iboost + 1}/{self.n_estimators}") return super()._boost(iboost, X, y, sample_weight, random_state)	167240887330895	12932164.84	6138320.2	0.6541378355	12746850.37
AdaBoost Regressor	Day 6: 30th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	model = make_pipeline(SelectFromModel(RandomForestRegressor(n_estimators=10)), VerboseAdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=3), n_estimators=100, learning_rate=0.5))	class VerboseAdaBoostRegressor(AdaBoostRegressor): def _boost(self, iboost, X, y, sample_weight, random_state): print(f"Training estimator {iboost + 1}/{self.n_estimators}") return super()._boost(iboost, X, y, sample_weight, random_state)	170347705223805	13051731.89	6504873.65	0.647672466	12852251.23


08
Gradient
Boosting
Regressor

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
Gradient Boosting Regressor	Day 3: 27th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, verbose=2)	-	165936078475250	12881617.85	5838357.88	0.6682961789	12678243.2
Gradient Boosting Regressor	Day 3: 27th	KBest	200	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, verbose=2)	-	166346106634387	12897523.28	5848139.33	0.6689946727	12694114.14
Gradient Boosting Regressor	Day 3: 27th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=1, verbose=2)	-	170893869736288	13072638.21	6162825.49	0.6472367879	12849989.27
Gradient Boosting Regressor	Day 3: 27th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=10, learning_rate=0.1, max_depth=1, verbose=2)	-	221794355757278	14892761.86	8448052.63	0.5378510641	1472038.14
Gradient Boosting Regressor	Day 4: 28th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=100, max_depth=3, learning_rate=0.1, subsample=1.0, verbose=3)	-	166009546737258	12884469.21	5839338.77	0.6682961789	12677408.57
Gradient Boosting Regressor	Day 4: 28th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=200, learning_rate=0.1, min_samples_leaf=4, min_samples_split=2, random_state=2,verbose=2, max_features='log2', max_depth= 35)	-	-	12929883.89	-	-	12758903.08

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
Gradient Boosting Regressor	Day 4: 28th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=500, learning_rate=0.1,min_samples_leaf=4, min_samples_split=2, random_state=2,verbose=2,max_features='log2')	-	-	12977232.75	-	-	12753991.47
Gradient Boosting Regressor	Day 4: 28th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=600, learning_rate=0.01,min_samples_leaf=5, min_samples_split=3, random_state=2,verbose=2,max_features='log2',max_depth= 12)	-	-	12785491.59	-	-	12584218.94
Gradient Boosting Regressor	Day 5: 29th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=50, max_depth=2, learning_rate=0.2, subsample=0.8, verbose=3)	-	167300856852361	12934483.25	5922902.55	0.659472992	12727187.19
Gradient Boosting Regressor	Day 5: 29th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=300, max_depth=4, learning_rate=0.05, subsample=0.9, verbose=3)	-	166411087802967	12900042.16	5816306.79	0.6899587895	12673824.44
Gradient Boosting Regressor	Day 6: 30th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=200, max_depth=5, learning_rate=0.1, subsample=0.85, verbose=3)	-	167918807493816	12958348.95	5779233.04	0.7234083877	12719710.06
Gradient Boosting Regressor	Day 6: 30th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=150, max_depth=2, learning_rate=0.15,	-	168175441514300	12968247.43	5928197.06	0.6665440366	12729943.37

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
Gradient Boosting Regressor	Day 6: 30th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=150, max_depth=2, learning_rate=0.15, subsample=0.7, verbose=3)	-	168175441514300	1296847.43	5928197.06	0.6665440366	12729943.37
Gradient Boosting Regressor	Day 6: 30th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=100, max_depth=4, learning_rate=0.1, subsample=0.8, max_features=0.3, verbose=3)	-	166345885246589	12897514.69	5828300.91	0.6797302754	12699980.41
Gradient Boosting Regressor	Day 6: 30th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=100, max_depth=3, learning_rate=0.1, subsample=0.9, min_samples_split=10, verbose=3)	-	166088771575068	12887543.27	5848968.5	0.6683665057	12696306.98
Gradient Boosting Regressor	Day 6: 30th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=50, max_depth=3, learning_rate=0.3, subsample=0.75, verbose=3)	-	169678225768784	13026059.49	5929284.63	0.6727837631	12748711.96
Gradient Boosting Regressor	Day 6: 30th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=120, max_depth=4, learning_rate=0.08, subsample=0.85, max_features='sqrt', verbose=3)	-	168921601061000	12996984.31	6003153.51	0.6636951318	12810788.63

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
Gradient Boosting Regressor	Day 6: 30th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=120, max_depth=4, learning_rate=0.08, subsample=0.85, max_features='sqrt', verbose=3)	-	168921601061000	12996984.31	6003153.51	0.6636951318	12810788.63
Gradient Boosting Regressor	Day 6: 30th	None	271	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=200, max_depth=3, learning_rate=0.1, subsample=0.6, verbose=3)	-	-	-	-	-	12738919.99
Gradient Boosting Regressor	Day 6: 30th	PCA	95%	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=120, max_depth=4, learning_rate=0.08, subsample=0.85, max_features='sqrt', verbose=3)	selector = VarianceThreshold(threshold=0.001) pca = PCA(n_components=0.95)	177446742877784	1332913.74	6629774.05	0.6616145252	13021222
Gradient Boosting Regressor	Day 7: 1st	KBest	200	Simple: Mean	MinMax	Simple: most_frequent	Dummy	Regular	GradientBoostingRegressor(n_estimators=600, learning_rate=0.01,min_samples_leaf=5, min_samples_split=3, random_state=2,verbose=2,max_features='log2',max_depth= 12)	-	-	-	-	-	12595325.58


09
XGB
Regressor

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
XGBoost Regressor	Day 1: 25th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	xgb.XGBRegressor()		177766344966137	13332904.6	5799980.84	0.7777357347	12982571.93
XGBoost Regressor	Day 2: 26th	Algorihm Feature Importance	20	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	xgb.XGBRegressor()		178515304193197	13360961.95	5808192.51	0.816158131	13112676.12
XGBoost Regressor	Day 2: 26th	Algorihm Feature Importance	40	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	xgb.XGBRegressor()	-	167743096759781	12951567.35	5903178.89	0.6620939104	12737222.66
XGBoost Regressor	Day 2: 26th	Algorihm Feature Importance	40	Simple: Median	MinMax	Simple: most_frequent	OneHot	Pipeline	xgb.XGBRegressor(max_depth=2)	-	166853913780964	12917194.5	5887385.84	0.6567607657	12709984.52
XGBoost Regressor	Day 2: 26th	Algorihm Feature Importance	40	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	xgb.XGBRegressor(max_depth=2, learning_rate=0.1)	-	166965016240907	12921494.35	5895809.67	0.6567594926	12717549.89
XGBoost Regressor	Day 2: 26th	Algorihm Feature Importance	40	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	xgb.XGBRegressor(max_depth=2, learning_rate=0.1)	-	167210261942516	12930980.7	5951696.16	0.6540416179	12731027.64
XGBoost Regressor	Day 2: 26th	Algorihm Feature Importance	40	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	xgb.XGBRegressor(max_depth=2, learning_rate=0.1)	-	165326527443054	12857936.36	5799900.84	0.6658882422	12662370.11
XGBoost Regressor	Day 2: 26th	Algorihm Feature Importance	40	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	xgb.XGBRegressor(max_depth=10, learning_rate=0.01, n_estimators=1000, subsample=0.8, colsample_bytree=0.8, reg_lambda=1, reg_alpha=0, random_state=42)	-	165149204783138	12851039.05	5519579.84	0.8922213222	12648162.24
XGBoost Regressor	Day 2: 26th	Algorihm Feature Importance	40	Simple: Mean	Standard	Simple: most_frequent	OneHot	Regular	XGBRegressor(max_depth=10, learning_rate=0.01, n_estimators=1000, subsample=0.8, colsample_bytree=0.8, reg_lambda=1, reg_alpha=0, random_state=42)	-	160024759236690	12650089.3	5364775.15	0.8686829979	12578391.07

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
									random_state=42)						
XGBoost Regressor	Day 3: 27th	KBest	40	Simple: Mean	Standard	Simple: most_frequent	OneHot	Regular	XGBRegressor(max_depth=10, learning_rate=0.01, n_estimators=1000, subsample=0.8, colsample_bytree=0.8, reg_lambda=1, reg_alpha=0, random_state=42)	-	161226712153427	12697508.11	5474978.07	0.9024452166	12631527.71
XGBoost Regressor	Day 3: 27th	KBest	88	Simple: Mean	Standard	Simple: most_frequent	OneHot	Regular	XGBRegressor(max_depth=10, learning_rate=0.01, n_estimators=1000, subsample=0.8, colsample_bytree=0.8, reg_lambda=1, reg_alpha=0, random_state=42)	kbest loop: 50 to 250 best: 88	159480942168010	12628576.41	5378167.48	0.9200141192	12621819.11
XGBoost Regressor	Day 3: 27th	None	271	Simple: Mean	Standard	Simple: most_frequent	OneHot	Regular	XGBRegressor(max_depth=10, learning_rate=0.01, n_estimators=1500, subsample=0.85, reg_lambda=0.2, reg_alpha=0.8)	-	161018423401054	12689303.5	5364447.09	0.8839744537	12572870.14
XGBoost Regressor	Day 4: 28th	None	271	Simple: Mean	Standard	Simple: most_frequent	OneHot	Regular	XGBRegressor(max_depth=18, learning_rate=0.009, n_estimators=2000, subsample=0.85, colsample_bytree=0.8, reg_lambda=0.2, reg_alpha=0.8, device = 'cuda', verbose = 2, tree_method = 'gpu_hist', predictor = 'gpu_predictor')	-	-	12682809.27	-	-	12466344.8

Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
XGBoost Regressor	Day 7: 1st	PCA	95%	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	max_depth=10, learning_rate=0.009, n_estimators=1400, subsample=0.8, colsample_bytree=0.7, reg_lambda=0.2, reg_alpha=0.8, verbose = 2, tree_method='hist', predictor='auto')	selector = VarianceThreshold(threshold=0.001) pca = PCA(n_components=0.95)	165138032534075	12850604.36	6067012.38	0.9717697147	12749993.52
XGBoost Regressor	Day 7: 1st	PCA	205, 95%	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	model = XGBRegressor(max_depth=10, learning_rate=0.009, n_estimators=1400, subsample=0.8, colsample_bytree=0.7, reg_lambda=0.2, reg_alpha=0.8, verbose = 2, tree_method='hist', predictor='auto')	selector = VarianceThreshold(threshold=0.001) 405 columns pca = PCA(n_components=0.95) 205 columns	160932596212293	12685921.18	5603029.43	0.9982051964	12570455.68


10
Stacking

Case Number	Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Params	Sample taken	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
66	Stacking	Day 4: 28th	None	271	Simple: Median	Standard	Simple: most_frequent	OneHot	Pipeline	rf1 = RandomForestRegressor(max_depth=39, n_estimators=400, max_features='sqrt', verbose=2, n_jobs=-1, min_samples_split=7) rf2 = RandomForestRegressor(max_depth=31, n_estimators=1400, max_features='log2', min_samples_leaf=2, min_samples_split=3, bootstrap=True, verbose=2, n_jobs=-1)	meta_regressor = RandomForestRegressor(max_depth=32, n_estimators=1500, max_features='log2', min_samples_leaf=2, min_samples_split=3, bootstrap=True, verbose=2, n_jobs=-1) stacking = StackingRegressor(estimators=[('rf1', rf1), ('rf2', rf2)], final_estimator=meta_regressor, passthrough=False)	-	-	167351990230424	1293649.73	5138216.02	-	1263762779
71	Stacking	Day 4: 28th	None	271	Simple: Median	Standard	Simple: most_frequent	OneHot	Pipeline	rf1 = RandomForestRegressor(max_depth=39, n_estimators=400, max_features='sqrt', verbose=2, n_jobs=-1, min_samples_split=7) dt2 = DecisionTreeRegressor (max_depth=31, max_features='log2', min_samples_leaf=2, min_samples_split=3)	meta_regressor = RandomForestRegressor(max_depth=32, n_estimators=1500, max_features='log2', min_samples_leaf=2, min_samples_split=3, bootstrap=True, verbose=2, n_jobs=-1) stacking = StackingRegressor(estimators=[('rf1', rf1), ('dt2', dt2)], final_estimator=meta_regressor, passthrough=False)	-	-	165907217060102	1288049755	5108580.96	0.7467883723	1274349748
78.2	Stacking	Day 5: 29th	None	271	Simple: Median	Standard	Simple: most_frequent	OneHot	Pipeline	rf1 = RandomForestRegressor(max_depth=39, n_estimators=400, max_features='sqrt', verbose=2, n_jobs=-1, min_samples_split=7) dt2 = DecisionTreeRegressor (max_depth=31, max_features='log2', min_samples_leaf=2, min_samples_split=3)	meta_regressor = RandomForestRegressor(max_depth=32, n_estimators=1500, max_features='log2', min_samples_leaf=2, min_samples_split=3, bootstrap=True, verbose=2, n_jobs=-1) stacking = StackingRegressor(estimators=[('rf1', rf1), ('dt2', dt2)], final_estimator=meta_regressor, passthrough=False)	-	-	167418138128401	1293916.12	5114919.8	0.6841359514	1270429023
80	Stacking	Day 5: 29th	None	271	Simple: Median	Standard	Simple: most_frequent	OneHot	Pipeline	rf1 = RandomForestRegressor(max_depth=39, n_estimators=400, max_features='sqrt', verbose=2, n_jobs=-1, min_samples_split=7) dt2 = DecisionTreeRegressor (max_depth=31, max_features='log2', min_samples_leaf=2, min_samples_split=3) xgb = XGBRegressor(max_depth=10, learning_rate=0.01, n_estimators=1000, subsample=0.8, colsample_bytree=0.8, reg_lambda=1, reg_alpha=0, random_state=42, verbose=True)	meta_regressor = RandomForestRegressor(max_depth=32, n_estimators=1500, max_features='log2', min_samples_leaf=2, min_samples_split=3, bootstrap=True, verbose=2, n_jobs=-1) stacking = StackingRegressor(estimators=[('rf1', rf1), ('dt2', dt2), ('xgb1', xgb)], final_estimator=meta_regressor, passthrough=False)	-	-	163785884848689	1279788595	5030737.99	0.8204545348	1261218202

Case Number	Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Params	Sample taken	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
94	Stacking	Day 5: 29th	None	271	Simple: Median	Standard	Simple: most_frequent	OneHot	Pipeline	rf1 = Ridge(alpha=100, solver='lsqr', tol=0.001, max_iter=1000) dt2 = Lasso(alpha=10000, max_iter=1000) xgb = LinearRegression(n_jobs=-1)	meta_regressor = RandomForestRegressor(max_depth=32, n_estimators=1500, max_features='log2', min_samples_leaf=2, min_samples_split=3, bootstrap=True, verbose=2, n_jobs=-1) stacking = StackingRegressor(estimators=[('rf1', rf1), ('dt2', dt2), ('xgb1', xgb)], final_estimator=meta_regressor, passthrough=False, n_jobs=-1, verbose=2)	-	-	176640562174842	13290619.33	6466013.71	0.6314408655	13131537.11
105	Stacking	Day 6: 30th	None	271	Simple: Median	Standard	Simple: most_frequent	OneHot	Pipeline	rf1 = GradientBoostingRegressor(n_estimators=50, max_depth=2, learning_rate=0.2, subsample=0.8, verboser=3) dt2 = VerboseAdaBoostRegressor(estimator=DecisionTreeRegressor(max_depth=10), n_estimators=50, learning_rate=1.0) xgb = XGBRegressor(verbosity=1, n_jobs=-1)	meta_regressor = RandomForestRegressor(max_depth=32, n_estimators=1500, max_features='log2', min_samples_leaf=2, min_samples_split=3, bootstrap=True, verbose=2, n_jobs=-1) stacking = StackingRegressor(estimators=[('rf1', rf1), ('dt2', dt2), ('xgb1', xgb)], final_estimator=meta_regressor, passthrough=False, n_jobs=-1, verbose=2)	-	-	1678157933178897	12954373.52	5709140.71	0.6538738122	12753400.84
107	Stacking	Day 6: 30th	Algorihm Feature Importance	100	Simple: Median	Standard	Simple: most_frequent	Dummy	Pipeline	rf1 = GradientBoostingRegressor(n_estimators=50, max_depth=2, learning_rate=0.2, subsample=0.8, verboser=3) dt2 = VerboseAdaBoostRegressor(estimator=DecisionTreeRegressor(max_depth=10), n_estimators=50, learning_rate=1.0) xgb = XGBRegressor(verbosity=1, n_jobs=-1)	meta_regressor = RandomForestRegressor(max_depth=32, n_estimators=1500, max_features='log2', min_samples_leaf=2, min_samples_split=3, bootstrap=True, verbose=2, n_jobs=-1) model = StackingRegressor(estimators=[('rf1', rf1), ('dt2', dt2), ('xgb1', xgb)], final_estimator=meta_regressor, passthrough=False, n_jobs=-1, verbose=2)	class VerboseAdaBoostRegressor(AdaBoostRegressor): def _boost(self, iboost, X, y, sample_weight, random_state): print(f"Training estimator {iboost + 1}/{self.n_estimators}") return super()._boost(iboost, X, y, sample_weight, random_state)	xgb, X, trainX, trainY, testX, test_data = featureImportance(xgb, 100, X, trainX, trainY, testX, test_data)	1672373983382O3	12932029.94	5665898.98	0.6629559532	12748650.72
123	Stacking	Day 7: 1st	Algorihm Feature Importance	200	Simple: Mean	Robust	Simple: most_frequent	Dummy	Regular	rf1 = GradientBoostingRegressor(n_estimators=50, max_depth=2, learning_rate=0.2, subsample=0.8, verboser=3) dt2 = DecisionTreeRegressor(random_state=0, max_depth=5, criterion='poisson') xgb = KNeighborsRegressor(n_neighbors=67, algorithm='auto', leaf_size=30, p=2, metric='euclidean', n_jobs=-1)	meta_regressor = RandomForestRegressor(max_depth=32, n_estimators=1500, max_features='log2', min_samples_leaf=2, min_samples_split=3, bootstrap=True, verbose=2, n_jobs=-1) model = StackingRegressor(estimators=[('rf1', rf1), ('dt2', dt2), ('xgb1', xgb)], final_estimator=meta_regressor, passthrough=False, n_jobs=-1, verbose=2)	decision tree algorithm feature importance 200 features	-	173474665659447	13170978.16	5765039.99	0.6464478155	12980226.73

Case Number	Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Params	Sample taken	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
										n_jobs=-1)								
124	Stacking	Day 7: 1st	KBest	200	Simple: Mean	Robust	Simple: most_frequent	Dummy dropFirst = False	Regular	model1 = KNeighborsRegressor(n_neighbors=30, algorithm='auto', leaf_size=50, p=3, metric='euclidean', n_jobs=-1, weights='distance') model2 = KNeighborsRegressor(n_neighbors=50, algorithm='auto', leaf_size=70, p=2, metric='minkowski', n_jobs=-1, weights='uniform') model3 = KNeighborsRegressor(n_neighbors=80, algorithm='auto', leaf_size=30, p=2, metric='euclidean', n_jobs=-1, weights='distance')	meta_regressor = KNeighborsRegressor(n_neighbors=67, algorithm='auto', leaf_size=10, p=2, metric='euclidean', n_jobs=-1, weights='distance') model = StackingRegressor(estimators=[('model1', model1), ('model2', model2), ('model3', model3)], final_estimator=meta_regressor, passthrough=False, n_jobs=-1, verbose=2)	-	-	164870186705817	12840178.61	5372458.49	0.8268361791	12689253.23
129	Stacking	Day 7: 1st	KBest	200	Simple: Mean	Standard	Simple: most_frequent	Dummy	Pipeline	model1 = KNeighborsRegressor(n_neighbors=30, algorithm='auto', leaf_size=50, p=3, metric='euclidean', n_jobs=-1, weights='distance') model2 = KNeighborsRegressor(n_neighbors=50, algorithm='auto', leaf_size=70, p=2, metric='minkowski', n_jobs=-1, weights='uniform') model4 = DecisionTreeRegressor (max_depth=5)	meta_regressor = KNeighborsRegressor(n_neighbors=67, algorithm='auto', leaf_size=10, p=2, metric='euclidean', n_jobs=-1, weights='distance') model = StackingRegressor(estimators=[('model4', model4)], final_estimator=meta_regressor, passthrough=False, n_jobs=-1, verbose=2)	-	-	169763989219714	13029351.07	6047039.85	0.6466727683	12858957.9
140	Stacking	Day 8: 2nd	None	271	Simple: Mean	Robust	Simple: most_frequent	Dummy	Regular	model = StackingRegressor(estimators=[('rf', RandomForestRegressor(n_estimators=100, max_depth = 5, random_state=42, verbose=2)), ('gb', GradientBoostingRegressor(n_estimators=10, random_state=42, verbose=1))], final_estimator=meta_regressor, passthrough=False, n_jobs=-1, verbose=2)	meta_regressor = AdaBoostRegressor(n_estimators=10)	-	-	-	-	-	-	12738231.25

Case Number	Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Params	Sample taken	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
141	Stacking	Day 8: 2nd	None	271	Simple: Mean	Robust	Simple: most_frequent	Dummy	Regular	model = StackingRegressor(estimators=[('rf', RandomForestRegressor(max_depth=5, n_jobs=-1, random_state=42, verbose=2)), ('gb', GradientBoostingRegressor(n_estimators=10,r andom_state=42, verbose=1))], final_estimator=meta_regressor, passthrough=False, n_jobs=-1, verbose=2)	meta_regressor = AdaBoostRegressor(n_estimators=10)	-	-	-	-	-	-	12737584.77
142	Stacking	Day 8: 2nd	None	271	Simple: Mean	Robust	Simple: most_frequent	Dummy	Regular	model = StackingRegressor(estimators= [('rf', RandomForestRegressor(n_estimators=100, max_depth = 5, random_state=42, verbose=2,n_jobs=-1)), ('xgb', XGBRegressor(n_estimators=200, max_depth=5,learning_rate=0.05, random_state=42))], final_estimator=meta_regressor, passthrough=False, n_jobs=-1, verbose=2)	meta_regressor = AdaBoostRegressor(n_estimators=10)	-	-	-	-	-	-	12661050.52
143	Stacking	Day 8: 2nd	None	271	Simple: Mean	Robust	Simple: most_frequent	Dummy	Regular	model = StackingRegressor(estimators= [('rf', RandomForestRegressor(n_estimators=200, max_depth=20, min_samples_split=5, min_samples_leaf=2, max_features='sqrt', bootstrap=True, random_state=42, n_jobs=-1)), ('ridge', Ridge(alpha=10.0, solver='saga', tol=1e-4, max_iter=10000, random_state=42))], final_estimator=meta_regressor, passthrough=False, n_jobs=-1, verbose=2)	meta_regressor = LinearRegression()	-	-	-	-	-	-	12490377.09

11

Neural Networks

Case Number	Model Name	Day	Feature Selection	No. of Features	Imputer on Numerical	Scaler on Numerical	Imputer on Categorical	One Hot	Pipeline	Model params	Model params	Params	Sample taken	Mean squared error	Root Mean squared error	Mean absolute error	model test score	kaggle score
79	Neural Networks	Day 5: 29th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	nn_model.fit(trainX, trainY, validation_data=(testX, testY), epochs=100, batch_size=32, verbose=1, callbacks=[lr_scheduler, early_stopping])	lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, verbose=1) early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=1)	The neural network consists of 4 layers: an input layer with 128 nodes (ReLU activation), two hidden layers with 64 nodes (ReLU) and 32 nodes (ReLU), followed by an output layer with a single node for regression. Batch normalization and dropout (0.2 rate) are applied after the first two layers. The model is trained using the Adam optimizer (learning rate = 0.001) and mean squared error as the loss function.	-	179050228228766	13380965.15	6277095.75	-	12910899.71
81	Neural Networks	Day 5: 29th	None	271	Simple: Median	MinMax	Simple: most_frequent	OneHot	Regular	nn_model.fit(trainX, trainY, validation_data=(testX, testY), epochs=100, batch_size=32, verbose=1, callbacks=[lr_scheduler, early_stopping])	lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, verbose=1) early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=1)	The neural network consists of 5 layers: an input layer with 128 nodes (ReLU activation), hidden layers with 64 nodes (ReLU), 32 nodes (ReLU), and 16 nodes (tanh), followed by an output layer with a single node for regression. Batch normalization and dropout (0.2–0.3 rates) are applied after some layers. The model is trained using the Adam optimizer (learning rate = 0.001) and mean squared error as the loss function.	-	697070495518187	26402092.64	14772166.09	-	28264436.83
82	Neural Networks	Day 5: 29th	None	271	Simple: Mean	Standard	Simple: most_frequent	OneHot	Regular	nn_model.fit(trainX, trainY, validation_data=(testX, testY), epochs=100, batch_size=32, verbose=1, callbacks=[lr_scheduler, early_stopping])	lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, verbose=1) early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=1)	The neural network consists of 5 layers: an input layer with 128 nodes (ReLU activation), hidden layers with 64 nodes (ReLU), 32 nodes (ReLU), and 16 nodes (ReLU), followed by an output layer with a single node for regression. Batch normalization and dropout (0.2–0.3 rates) are applied after selected layers. The model uses the Adam optimizer (learning rate = 0.001) and mean squared error as the loss function.	-	175078256634941	13231714.05	6347543.5	-	13057066.03
83	Neural Networks	Day 5: 29th	None	271	Simple: Mean	Standard	Simple: most_frequent	OneHot	Regular	nn_model.fit(trainX, trainY, validation_data=(testX, testY), epochs=150, batch_size=16, verbose=2, callbacks=[lr_scheduler, early_stopping])	lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, verbose=1) early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=1)	The neural network has 5 layers: an input layer with 128 nodes (ReLU activation, L2 regularization of 0.001), hidden layers with 64 nodes (ReLU), 32 nodes (ReLU), and 16 nodes (ReLU), followed by an output layer with a single node for regression. Batch normalization and dropout (0.2–0.3 rates) are applied after selected layers. The model uses the Adam optimizer (learning rate = 0.001) and mean squared error as the loss function. The input dimension is specified by the input_dim parameter.	-	171660426605292	13101924.54	6327963.11	-	12845636.43
103	Neural Networks	Day 6: 30th	None	271	Simple: Mean	Standard	Simple: most_frequent	Dummy	Regular	nn_model.fit(trainX, trainY, validation_data=(testX, testY), epochs=150, batch_size=8, verbose=2, callbacks=[lr_scheduler, early_stopping])	lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, verbose=2) early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=2)	The build_nn function constructs a neural network with 5 layers: an input layer with 128 nodes (ReLU activation, L2 regularization of 0.001), hidden layers with 64 nodes (ReLU), 32 nodes (ReLU), and 16 nodes (ReLU), followed by an output layer with a single node for regression. Batch normalization and dropout (0.2–0.3 rates) are applied after selected layers. The model is compiled with the Adam optimizer (learning rate = 0.001) and mean squared error as the loss function. The input dimension is specified by the input_dim parameter.	-	174259745175685	13200747.9	6573614.96	-	12950464.58
104	Neural Networks	Day 6: 30th	KBest	100	Simple: Median	Standard	Simple: most_frequent	Dummy	Regular	nn_model.fit(trainX, trainY, validation_data=(testX, testY), epochs=150, batch_size=16, verbose=2, callbacks=[lr_scheduler, early_stopping])	lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, verbose=2) early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=2)	The build_nn function creates a neural network with 5 layers: an input layer (128 nodes, ReLU activation, L2 regularization of 0.001), hidden layers (64, 32, and 16 nodes, ReLU activation), and an output layer with 1 node for regression. Batch normalization stabilizes training, while dropout (0.2–0.3 rates) prevents overfitting. The model uses the Adam optimizer (learning rate = 0.001) and mean squared error as the loss function, balancing efficient optimization with regularization for better generalization.	-	466469850753343	21597913.11	12689852.71	-	21491407.46
106	Neural Networks	Day 6: 30th	PCA	95%	Simple: Median	Standard	Simple: most_frequent	Dummy	Pipeline	nn_model.fit(trainX, trainY, validation_data=(testX, testY), epochs=150, batch_size=16, verbose=2, callbacks=[lr_scheduler, early_stopping])	lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, verbose=2) early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=2)	The neural network built by build_nn includes an input layer with 128 nodes using ReLU activation and L2 regularization (0.001) to prevent overfitting. The first hidden layer has 64 nodes (ReLU), batch normalization to stabilize learning, and dropout (rate = 0.3). The second hidden layer has 32 nodes (ReLU), followed by a third hidden layer with 16 nodes (ReLU) and dropout (rate = 0.2). The output layer consists of a single node for regression tasks. The model is compiled with the Adam optimizer (learning rate = 0.001) and mean squared error loss, combining regularization, normalization, and dropout for robust and generalizable training.	selector = VarianceThreshold(threshold=0.01) pca = PCA(n_components=0.95)	1732558137992 30	13162667.43	6386024.64	-	12957902.03

Thank you.