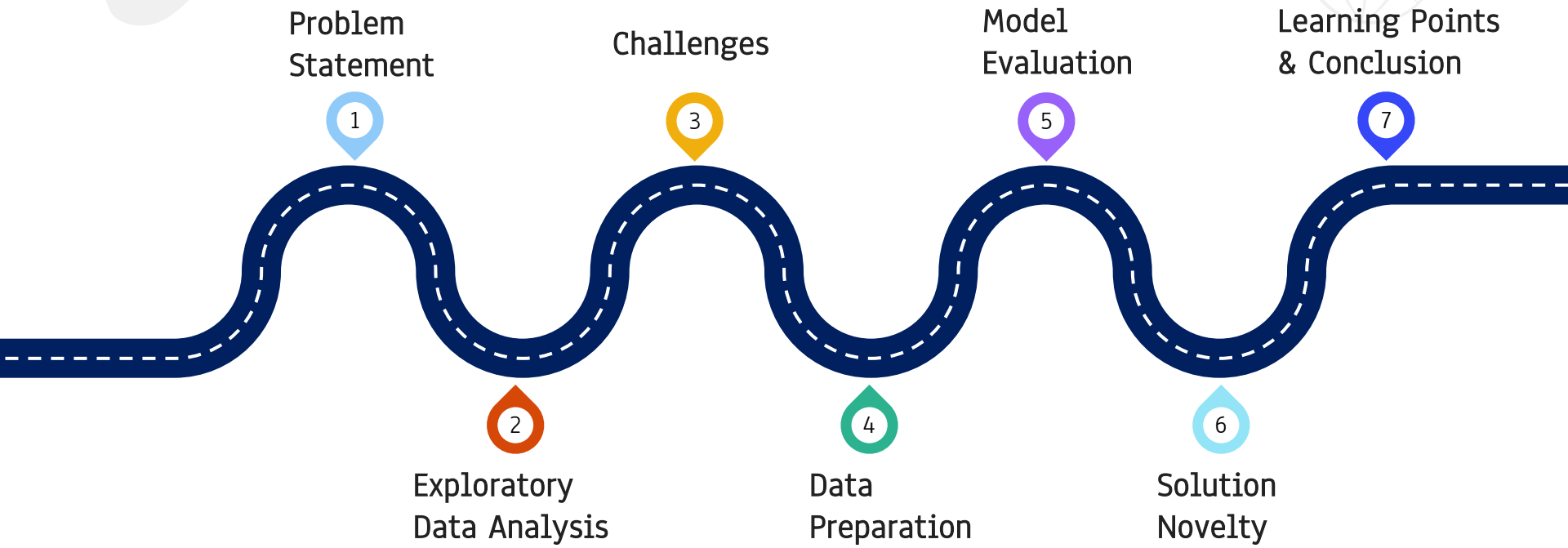
Realty Price Prediction for Sberbank Russian Housing Market

Rank 532/3265 (Top 17%)

By Group 50:
Liew Zhi Li (Sherna)
Huang NengQi
Mark Tan Rong Hui
Goh Wei Jie Benjamin

Outline


1
Problem Statement
2
Exploratory Data Analysis
3
Challenges
4
Data Preparation
5
Model Evaluation
6
Solution Novelty
7
Learning Points & Conclusion

01

Problem Statement

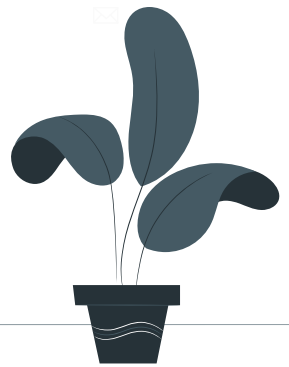
Can we make accurate predictions on Russian realty prices?

Introduction

- Competition created by Sberbank
- Sberbank offers realty prices prediction
- Realty price prediction is a complex task
- Myriad of housing features
- Russia's volatile economy
- Goal: Utilize Russian housing market dataset to develop models to predict realty prices

02

Exploratory Data Analysis



Overview of Datasets

Macro.csv

- Size: 2484, 100
- Float64: 94
- Int64: 2
- Object: 4

Train.csv

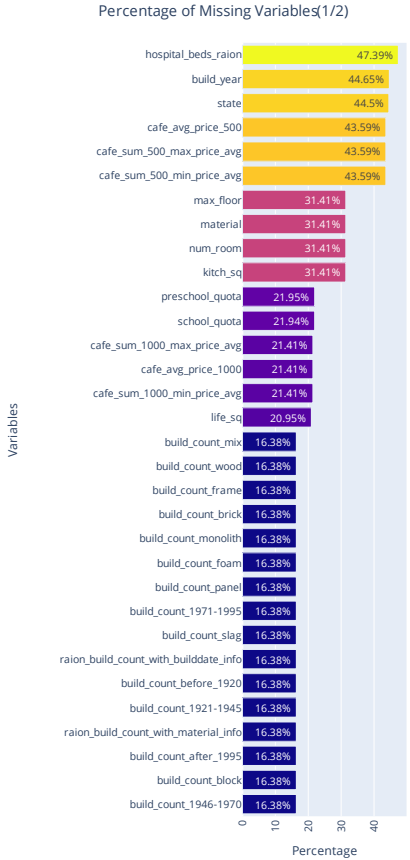
- Size: 30471, 292
- Float64: 119
- Int64: 157
- Object:16

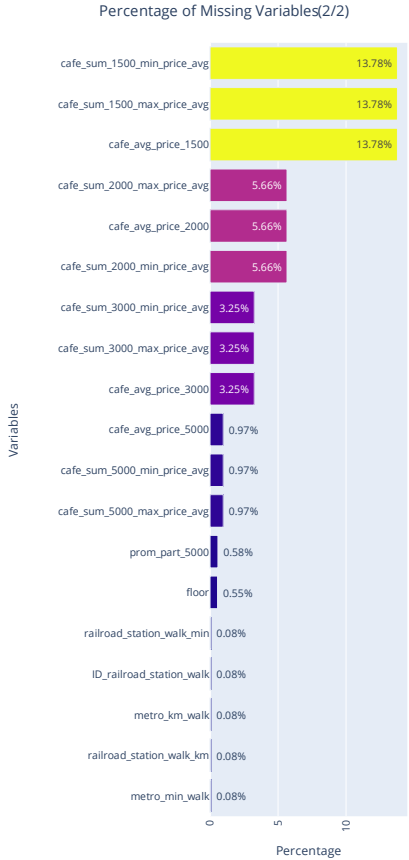
Test.csv

- Size: 7662, 291
- Float64: 116
- Int64: 159
- Object:16

Missing Values

- 51 out of 292 features contains missing values
- Negatively affects ML models
- Requires imputation

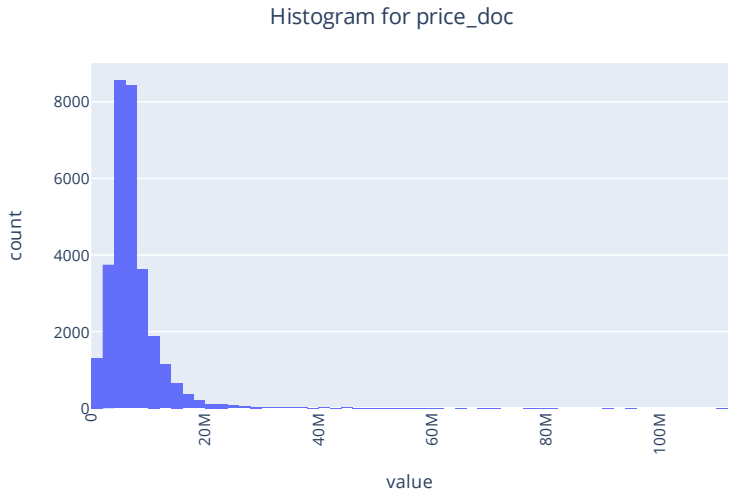




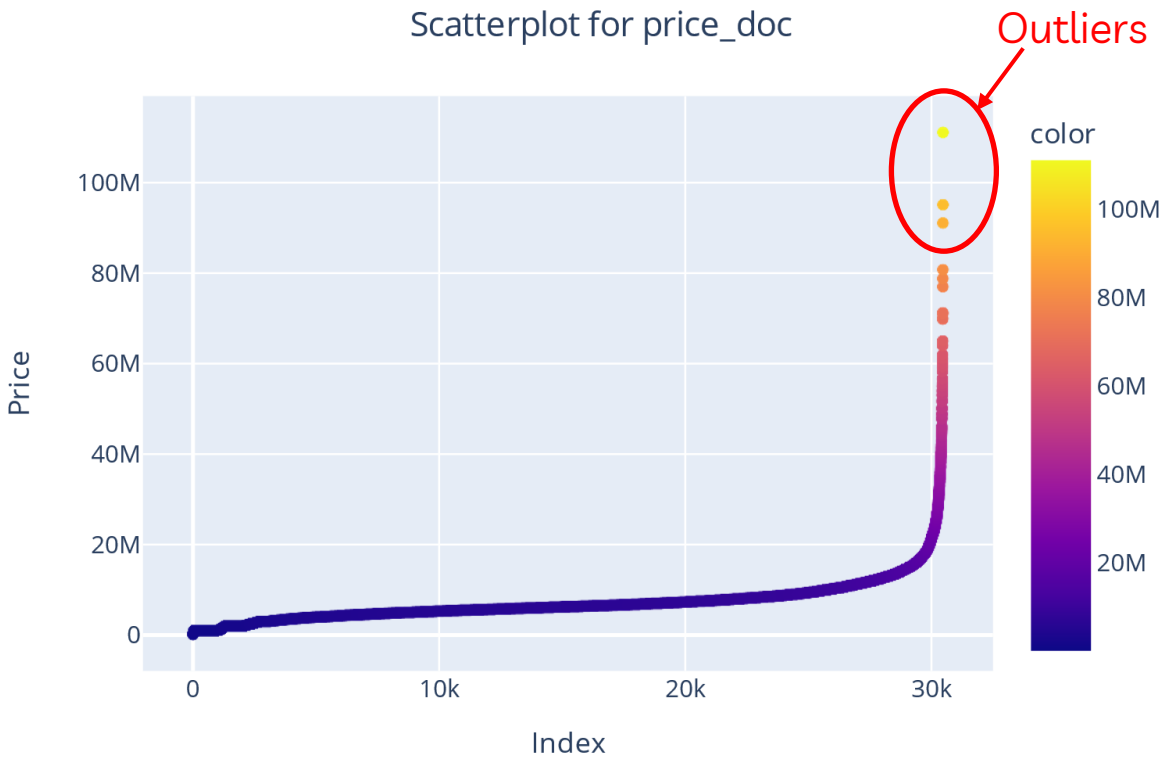
Univariate Analysis of Target Variable price_doc

Stats & Histogram of price_doc

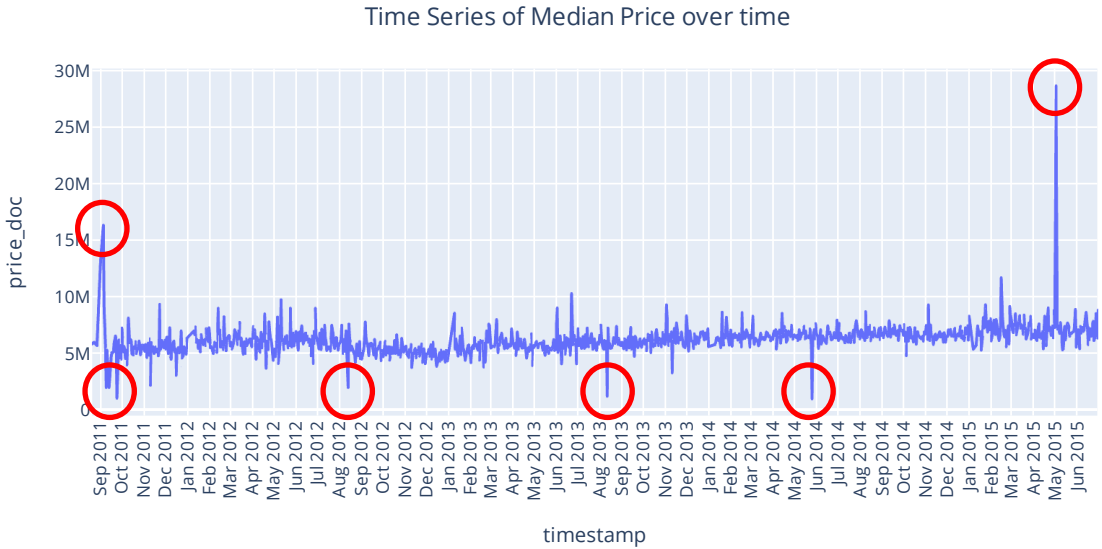
Descriptive Statistics for price_doc	
count	30,471.00
mean	7,123,035.28
std	4,780,111.33
min	100,000.00
25%	4,740,002.00
50%	6,274,411.00
75%	8,300,000.00
max	111,111,112.00



Scatterplot for price_doc

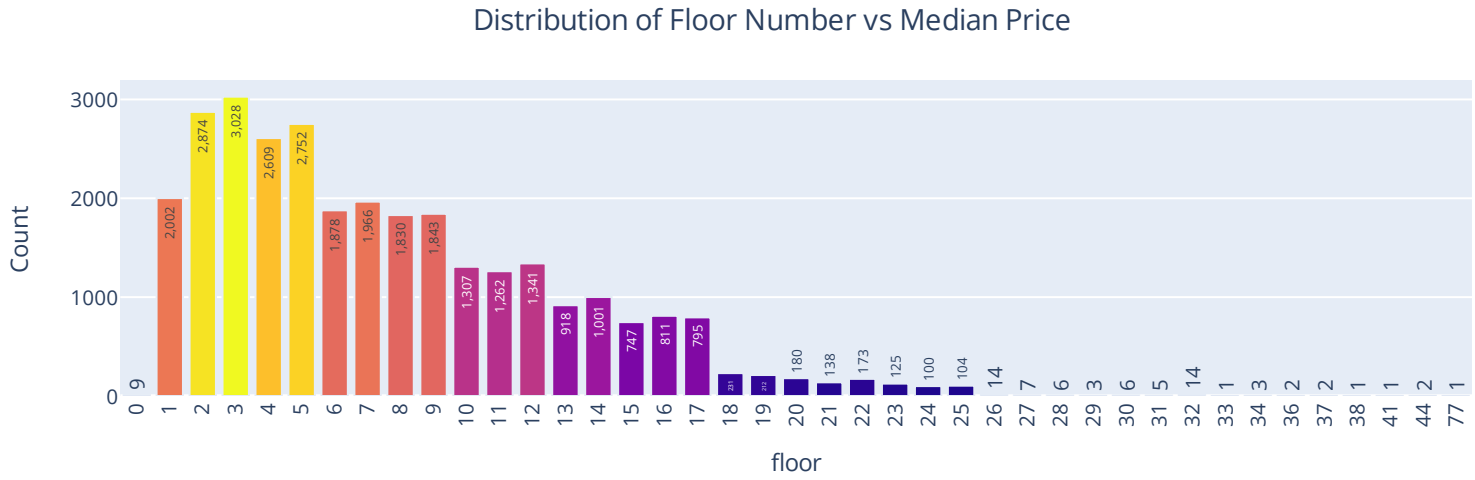


Median Price over Time

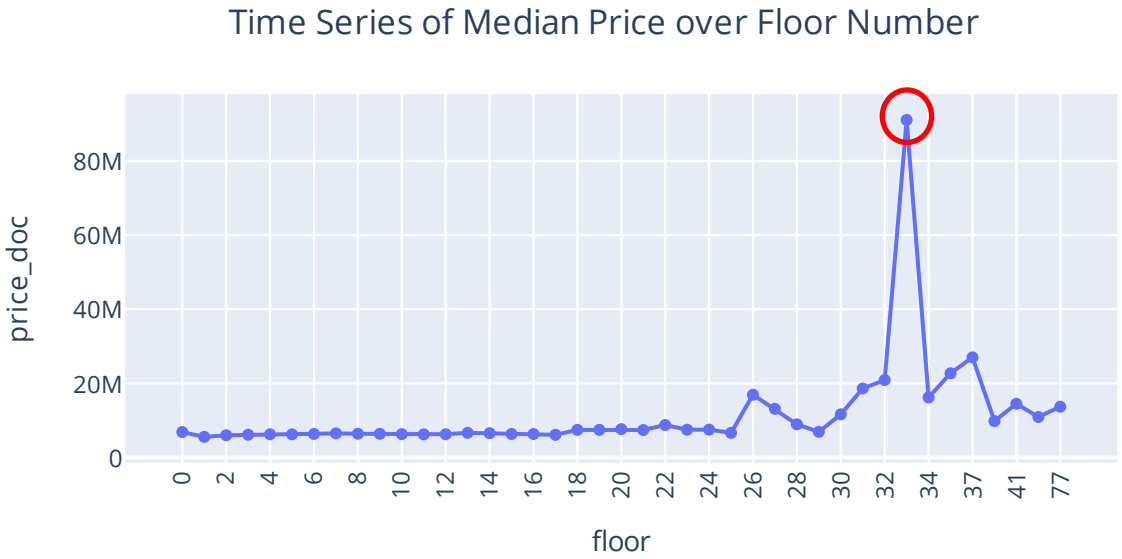


Bivariate Analysis of Different Features against Target Variable price_doc

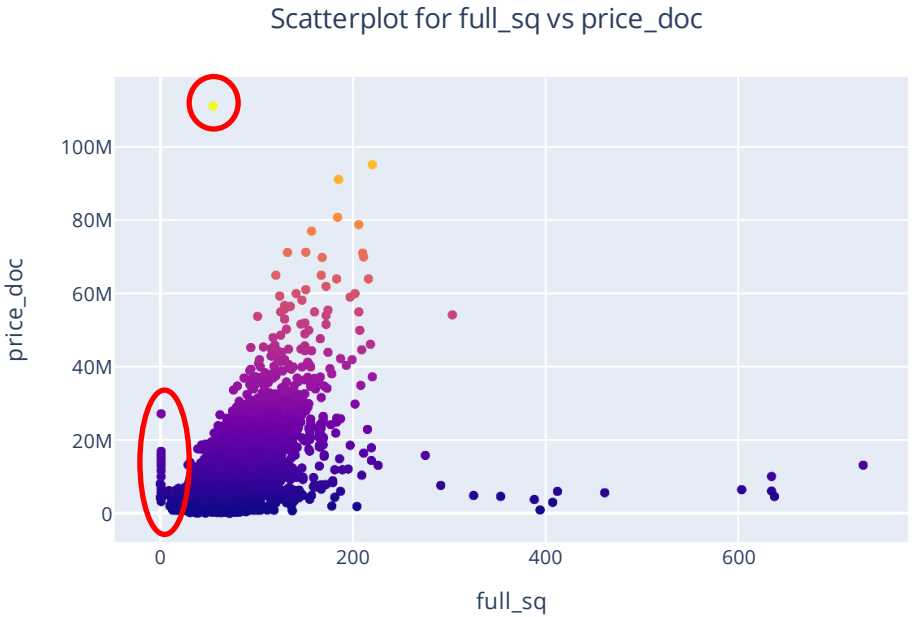
Floor Number vs Price



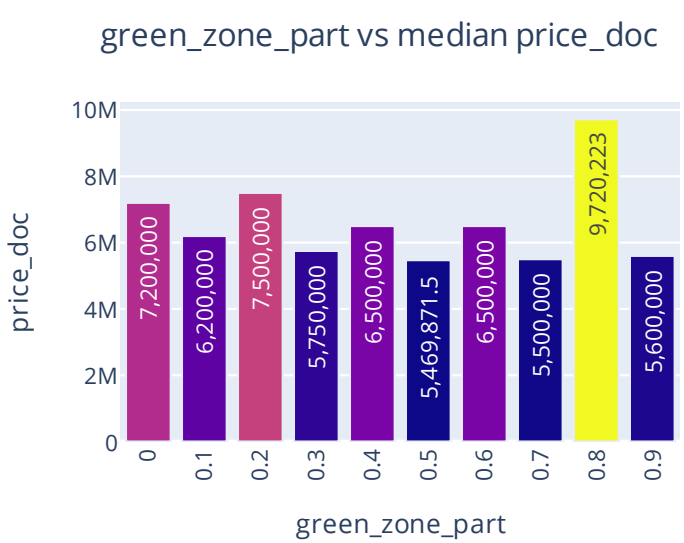
Median Price vs Floor Number

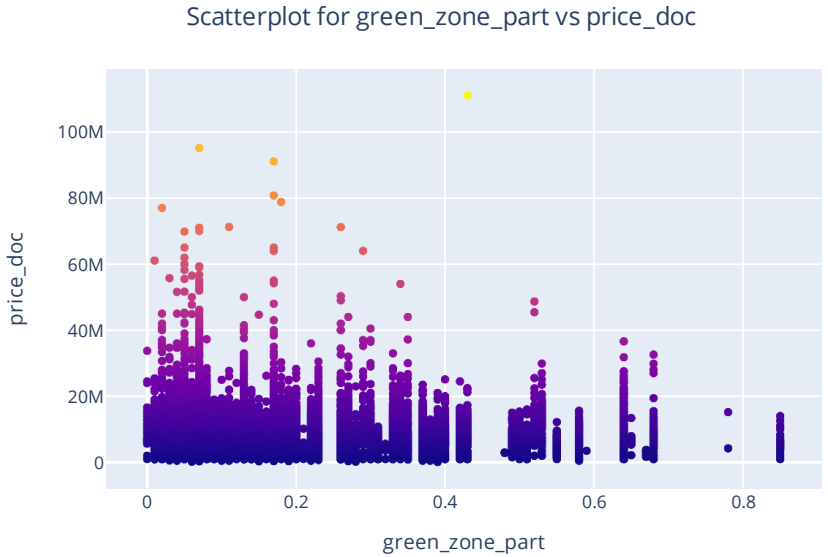


Total Area vs Price

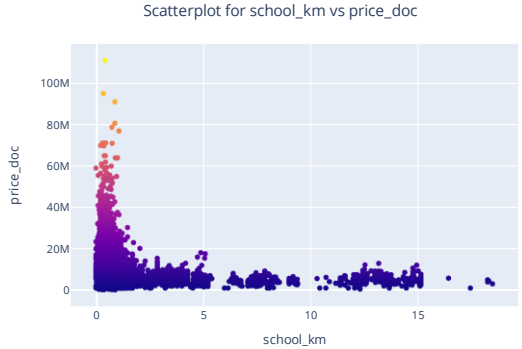


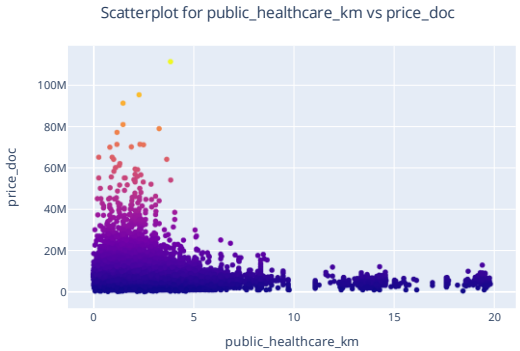
Green Zone vs Price

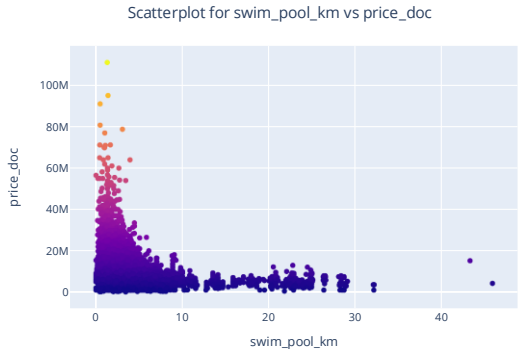


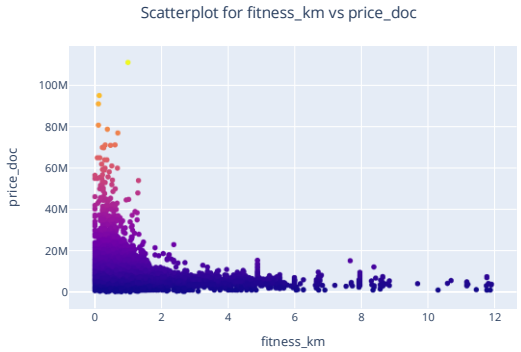


9 Facilities vs Price (1/2)

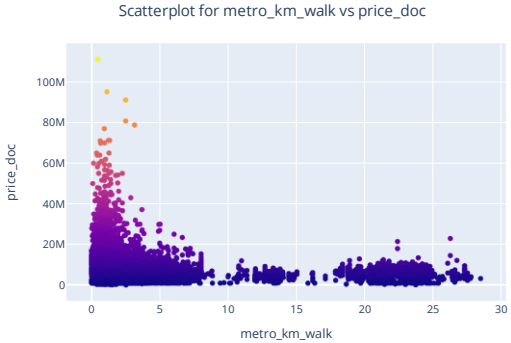


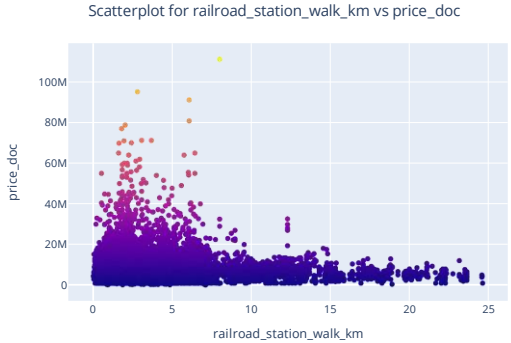


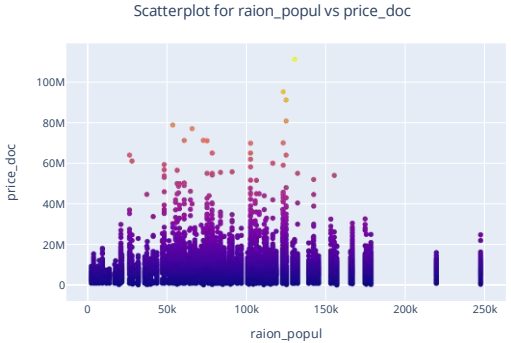


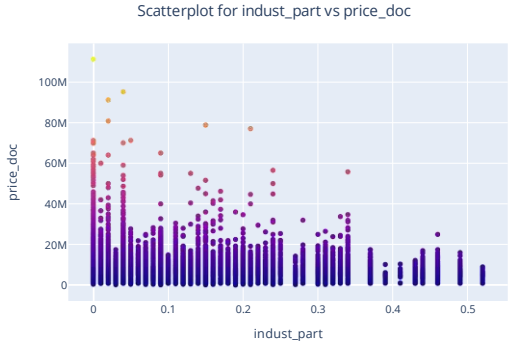


9 Facilities vs Price (2/2)



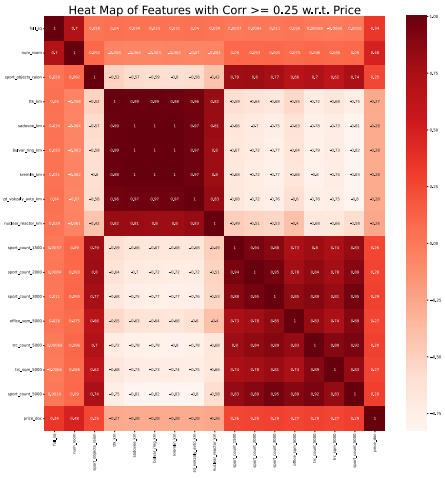


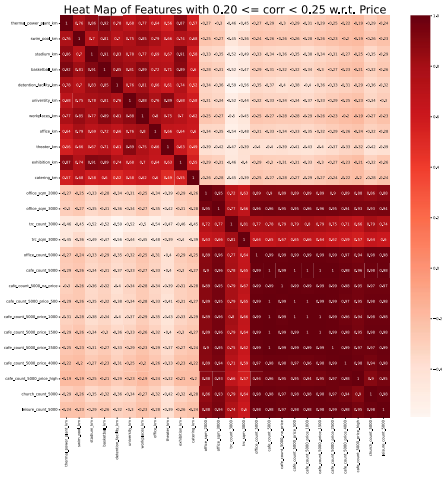


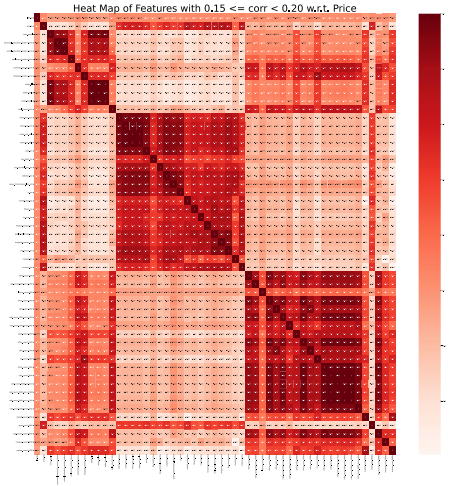


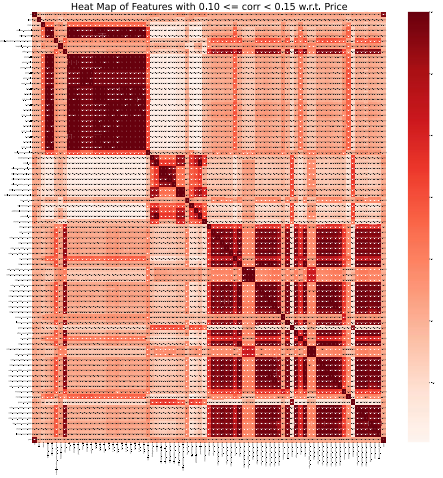
Correlation Analysis of Different Features against Target Variable price_doc

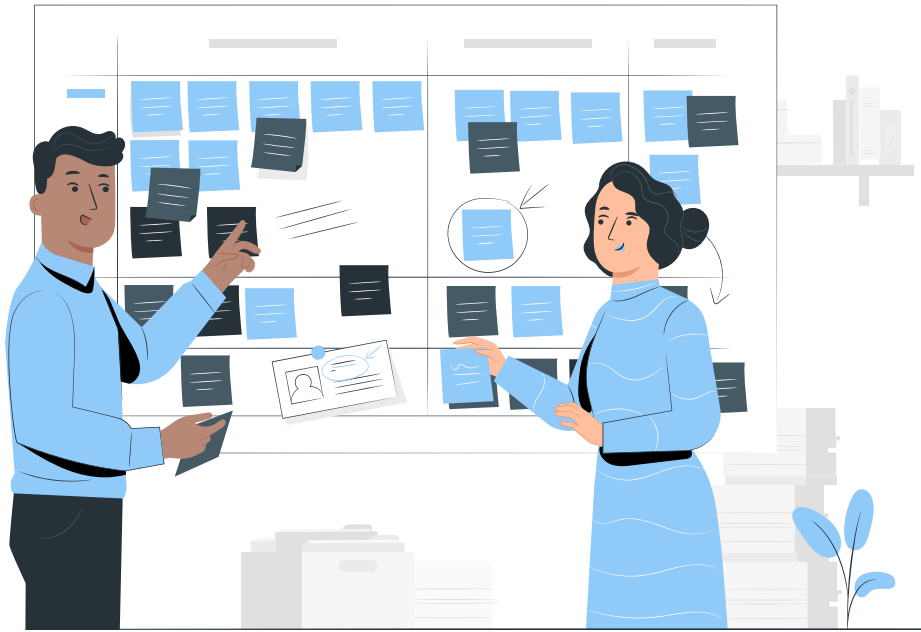
Correlation Heat Maps











03

Challenges

Challenges

Missing Values

- 51 features with missing values
- Causes bias
- Leads to reduced model accuracy

Potential Anomalies

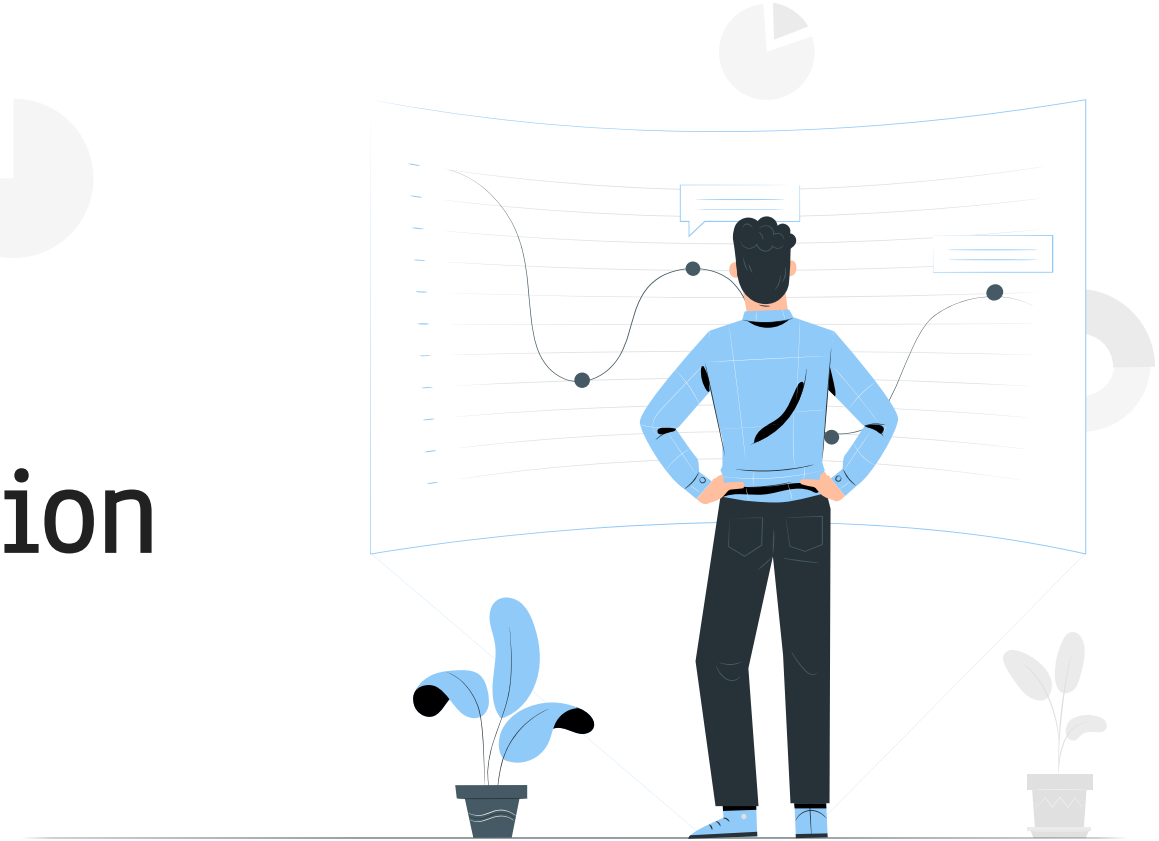
- Leads to reduced model accuracy

High Collinearity

- Model learns noise from collinear features
- Leads to overfitting

04

Data Preparation



Data Pre-Processing: Cleaning of Data

01. Identify all the data types

```
====== Dataset size - train: (30471, 292)
Number of distinct datatypes:
<class 'numpy.dtype[int64]'>      157
<class 'numpy.dtype[float64]'>    119
<class 'numpy.dtype[object_]'>     16
dtype: int64
```

02.

- Extract a copy of 'id' and 'price_doc' from train dataset
- Merging train dataset minus 'price_doc' with test dataset

03. Dropping features or columns with low correlation of less than 5%

build_year	0.0022
kitch_sq	0.0287
school_quota	0.0140
culture_objects_top_25_raion	0.0443
full_all	0.0253
male_f	0.0264
female_f	0.0243
16_29_all	0.0223
16_29_male	0.0231
16_29_female	0.0216
build_count_block	0.0315
build_count_wood	0.0425
build_count_frame	0.0303
build_count_panel	0.0201
build_count_foam	0.0107
build_count_slag	0.0240
build_count_mix	0.0330
build_count_1921-1945	0.0203
build_count_1971-1995	0.0097
build_count_after_1995	0.0259
cemetery_km	0.0249
ID_railroad_station_walk	0.0218
water_km	0.0266
mkad_km	0.0206
big_market_km	0.0483
prom_part_500	0.0090
trc_sqm_500	0.0004
cafe_sum_500_min_price_avg	0.0364
cafe_sum_500_max_price_avg	0.0379
cafe_avg_price_500	0.0374
cafe_count_500_na_price	0.0492
big_church_count_500	0.0262
church_count_500	0.0147
mosque_count_500	0.0185
market_count_500	0.0404
trc_sqm_1000	0.0416
cafe_count_1000_price_4000	0.0363
prom_part_3000	0.0227
cafe_sum_3000_min_price_avg	0.0051
cafe_sum_3000_max_price_avg	0.0022
cafe_avg_price_3000	0.0033
cafe_sum_5000_min_price_avg	0.0322
cafe_sum_5000_max_price_avg	0.0333
cafe_avg_price_5000	0.0329

Feature Engineering

Original Features	Modified Features
'timestamp'	'year' 'year_month'
'year_month'	'sales_year_month'
'life_sq' 'full_sq'	'living_area_ratio' 'non_living_area' 'non_living_area_ratio'
'life_sq' 'num_room'	'room_area_avg'
'floor' 'max_floor'	'relative_floor'
'max_floor' 'sub_area'	'sub_area_building_height_avg'
'sub_area' 'kremlin_km'	'sub_area_kremlin_dist_avg'

Missing Values Imputation

01. Obtain the number of missing values in each feature

```
hospital_beds_raion                  17859
room_area_avg                        14897
state                                14253
relative_floor                       10355
max_floor                            10355
num_room                              9586
material                              9572
preschool_quota                       8284
cafe_sum_1000_min_price_avg           7746
cafe_avg_price_1000                   7746
cafe_sum_1000_max_price_avg           7746
non_living_area_ratio                 7562
non_living_area                       7562
living_area_ratio                     7562
life_sq                               7559
raion_build_count_with_material_info  6209
build_count_brick                     6209
build_count_before_1920               6209
raion_build_count_with_builddate_info 6209
build_count_monolith                  6209
build_count_1946-1970                 6209
cafe_avg_price_1500                   5020
cafe_sum_1500_min_price_avg           5020
cafe_sum_1500_max_price_avg           5020
cafe_sum_2000_max_price_avg           2149
cafe_sum_2000_min_price_avg           2149
cafe_avg_price_2000                   2149
prom_part_5000                         270
floor                                  167
metro_min_walk                          59
railroad_station_walk_km                59
railroad_station_walk_min               59
metro_km_walk                           59
product_type                            33
green_part_2000                         19
full_sq                                  3
dtype: int64

Missing Values Count: 36
```

02. Median imputation is used for features with less than 30% missing values

03. KNN Imputer is used for the remaining features

```
hospital_beds_raion    0.4683
room_area_avg          0.3907
state                  0.3738
dtype: float64

Missing Values Count: 3
```

Prepare Train and Test Datasets

01. Separate the merged data back to the train and test datasets.

02.

- Create yTrain, containing 'price_doc'
- Create xTrain, dropping 'id', 'timestamp' and 'price_doc'.
- Use sklearn's train_test_split function to obtain the training and cross-validation datasets (x_tr, y_tr, x_cv, y_cv).

03. Process the data as categorical or numerical using label encoder from sklearn preprocessing

```
# categoricals
for cat in categoricals:
  le = preprocessing.LabelEncoder()
  le.fit(x_tr[cat])

  x_cv[cat] = x_cv[cat].map(lambda s: '<unknown>' if s not in le.classes_ else s)
  le.classes_ = np.append(le.classes_, '<unknown>')

  x_tr[cat] = le.transform(x_tr[cat])
  x_cv[cat] = le.transform(x_cv[cat])

# numericals
for num in numericals:
  min = x_tr[num].min()
  max = x_tr[num].max()
  x_tr[num] = (x_tr[num] - min)/(max-min)
  x_cv[num] = (x_cv[num] - min)/(max-min)
```

05
Model Evaluation

Approach

1 Random Forest

Random Forest

0.0878

Train MSE

0.2250

Test MSE

0.2962

Train RMSE

0.4743

Test RMSE

0.35275

Kaggle RMSLE

- Ensemble based on Decision trees
- Bootstrapping features for each tree
- Does not suffer from overfitting
- The final result base on voting/ averaging

parameters	n_estimators	max_depth
values	300	15

Approach

2 Decision Tree

Decision Tree

0.2341

Train MSE

0.2449

Test MSE

0.4838

Train RMSE

0.4949

Test RMSE

0.36380

Kaggle RMSLE

- Simple to understand and interpret
- Requires little data preparation
- Computational inexpensive (O(number of data points used in the tree))

parameters	max_depth
values	14

Approach

3 XGBoost

XGBoost

0.0063

Train MSE

0.2355

Test MSE

0.0794

Train RMSE

0.4853

Test RMSE

0.36423

Kaggle RMSLE

- Ensemble based on Decision trees
- Gradient Boosting Framework: Trees made iteratively learning from previous trees errors using gradient descent
- Fast execution speed
- Good model performance

parameters	colsample_bytree	learning_rate	max_depth	n_estimators	subsample
values	0.25	0.1	15	100	1

Approach

4 SGD Regressor

SGD Regressor

91.1125

Train MSE

90.4085

Test MSE

9.5453

Train RMSE

9.5083

Test RMSE

9.07412

Kaggle RMSLE

- Efficient for linear classifiers and regressors under convex loss functions
- Efficient and easy to implement
- Good at Handle data with large training and feature size.

parameters	alpha	max_iter
values	1e4	1000

Approach

5 AdaBoost

AdaBoost

0.5598

Train MSE

0.5538

Test MSE

0.7482

Train RMSE

0.7442

Test RMSE

0.70313

Kaggle RMSLE

- Ensemble learning method
- Iterative algorithm that learns and avoid mistakes
- Easy to implement with minimum parameters to adjust
- Not prone to overfitting

parameters	n_estimators
values	50

Approach

6 LightGBM

LightGBM

0.1414

Train MSE

0.2117

Test MSE

0.3760

Train RMSE

0.4601

Test RMSE

0.32355

Kaggle RMSLE

- Gradient boosting model with tree-based algorithms
- Fast training speed
- Efficient on large-scale data

Roadblock

- Best Kaggle score thus far only **0.32355**
- Pre-processed data could be negatively affecting score
- Further research discovering "magic numbers"
    - Lower average prices in test set
    - Possible fake prices in train set
    - Altering price values with "magic numbers" improves performance after training

```
xtrainnxgb = dfs["train"]
xtrainnxgb=xtrainnxgb[(xtrainnxgb.price_doc>1e6) & (xtrainnxgb.price_doc!=2e6) & (xtrainnxgb.price_doc!=3e6)]
xtrainnxgb.loc[(xtrainnxgb.product_type=='Investment') & (xtrainnxgb.build_year<2000),'price_doc']*=0.895
xtrainnxgb.loc[xtrainnxgb.product_type!='Investment','price_doc']*=0.96
```

Approach

7

Naive XGBoost (Raw)

(Minimally-Processed Dataset)

Naive XGBoost (Raw)

- Predict on minimally processed dataset (min-data)
- Hyperparameters tuned (RandomSearchCV)
    - colsample_bytree
    - learning_rate
    - max_depth
    - n_estimators
    - subsample
    - min_child_weight
- Compare performance with "magic numbers"

0.31978	0.31275
min-data	**min-data w/ magic numbers**

Approach

8 LightGBM (Raw)

(Minimally-Processed Dataset)

LightGBM (Raw)

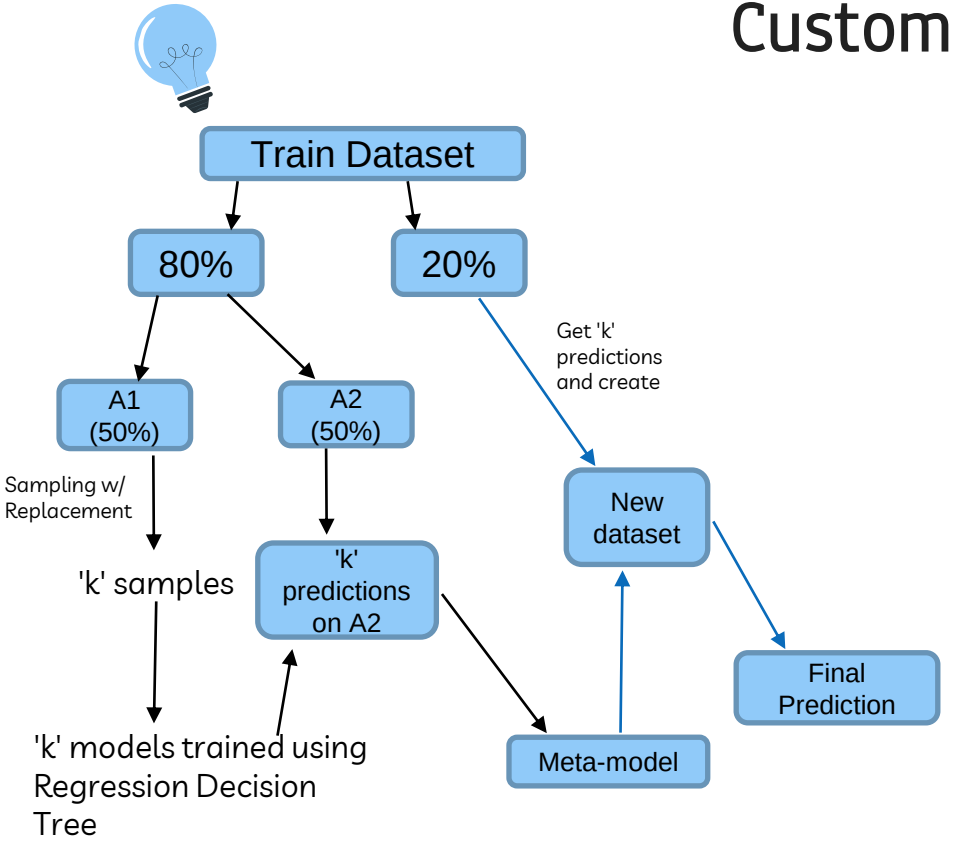
- Predict on minimally processed dataset (min-data)
- Compare performance with "magic numbers"

0.32789	0.31041
min-data	**min-data w/ magic numbers**

06

Solution Novelty

Custom Model



- Regression Decision Tree and sampling
- Preparation
  - Train dataset 80-20 split, 80 split further into 50-50 (A1 & A2)
- 'k' models created
  - 'k' predictions on A2
  - Create new dataset and train Meta-model
- Get 'k' predictions on 20% test split
  - Create new dataset
- Use Meta-model to predict on new dataset to get final prediction

Custom Model

0.2323

Test MSE

0.4820

Test RMSE

0.36742

Kaggle RMSLE

Leaderboard

Model	Private score	Public score	Public Rank (Out of 3265)
Random Forest	0.35739	0.35275	2668
Decision Tree	0.36366	0.36380	2734
XGBoost	0.37180	0.36423	2735
SGD Regressor	9.09261	9.07412	3260
AdaBoost	0.70015	0.70313	3220
LightGBM	0.32451	0.32355	1831
Naïve XGBoost (Raw)	0.32162	0.31741	1384
LightGBM (Raw)	0.31443	**0.31041**	532
Custom Model	0.36969	0.36742	2751

07

Learning Points & Conclusion

Learning points

Importance of Exploratory Data Analysis

- Helps to identify all challenges faced in the datasets
- Forming different solutions to each challenges

Influence of Data Pre-Processing

- Different expectations between theory and result
- Lesser pre-processing might have a better result

Conclusion

- Machine Learning Workflow
  - Exploratory Data Analysis
  - Data Pre-processing
  - Feature Engineering
  - Model tuning, training, evaluation
- Unexpected Findings
  - Level of data pre-processing
  - Presence of unreliable data in dataset
    - Fake prices
    - Magic Numbers
- Results
  - LightGBM (min-data)
  - Score 0.31041
  - Rank 532/3265 (Top 17%)

Thank You!