Russian Housing and Missing Data

Laura Lazarescou, John Rodgers, Maysam Mansor and Mel Schwan

March 15, 2021

Introduction

Data Analysts must develop techniques for dealing with missing data in any new dataset. Removing records containing missing values could result in the loss of valuable insights. In most cases, the favored method of managing missing data is to create an estimated value based on the other values in the same dataset, then impute the missing data using available software packages. These imputation packages use methods that preserve the valuable data present, which allows the models to be trained on a complete dataset.

Missing data can follow predictable patterns that can be detected visually or through a data interpretation tool. Discovering why the data is missing allows the analyst to decide on their imputation strategy.

Depending on why the data is missing, we will choose the imputation strategy. Patterns in the existing data and the missing data will determine the choice of imputation method. It is also important to capture any patterns that may exist in the randomness of the data. Is the missing data Missing Completely At Random (MCAR), Missing At Random (MAR) or Missing Not At Random (MNAR)? If the data is MNAR we may want to fit a linear regression or other model to impute the data.

In this project we will evaluate a series of factors that include varying degrees of missing values, then based on their patterns and the distribution of existing data, we will impute each factor. Once all factors are imputed we will fit two models and compare the Root Mean Square Error of each model. In one case the dataset will contain our imputed values. In the second case we will impute all missing values with negative one (-1). Our analysis will compare the RMSE for both models and address the differences in performance.

Types of Missing Data

If data is MAR, it is interpeted at a there is a structured relationship between missing data and other values in the dataset. Other features in the dataset may help predict the missing data.

MCAR is the opposite case. No relationship can be established between existing data and missing data. Missingness of this data has no logical pattern so we look to the data in the same factor to decide how to impute missing values.

Data that is MNAR means that the probability of "missingness" is not random, but the relationship is not known. An example of this phenomenon is missing data due to a aircraft engine monitor equipment becoming inoperable over time. We may not know when this will occur but it will occur.

The Dataset

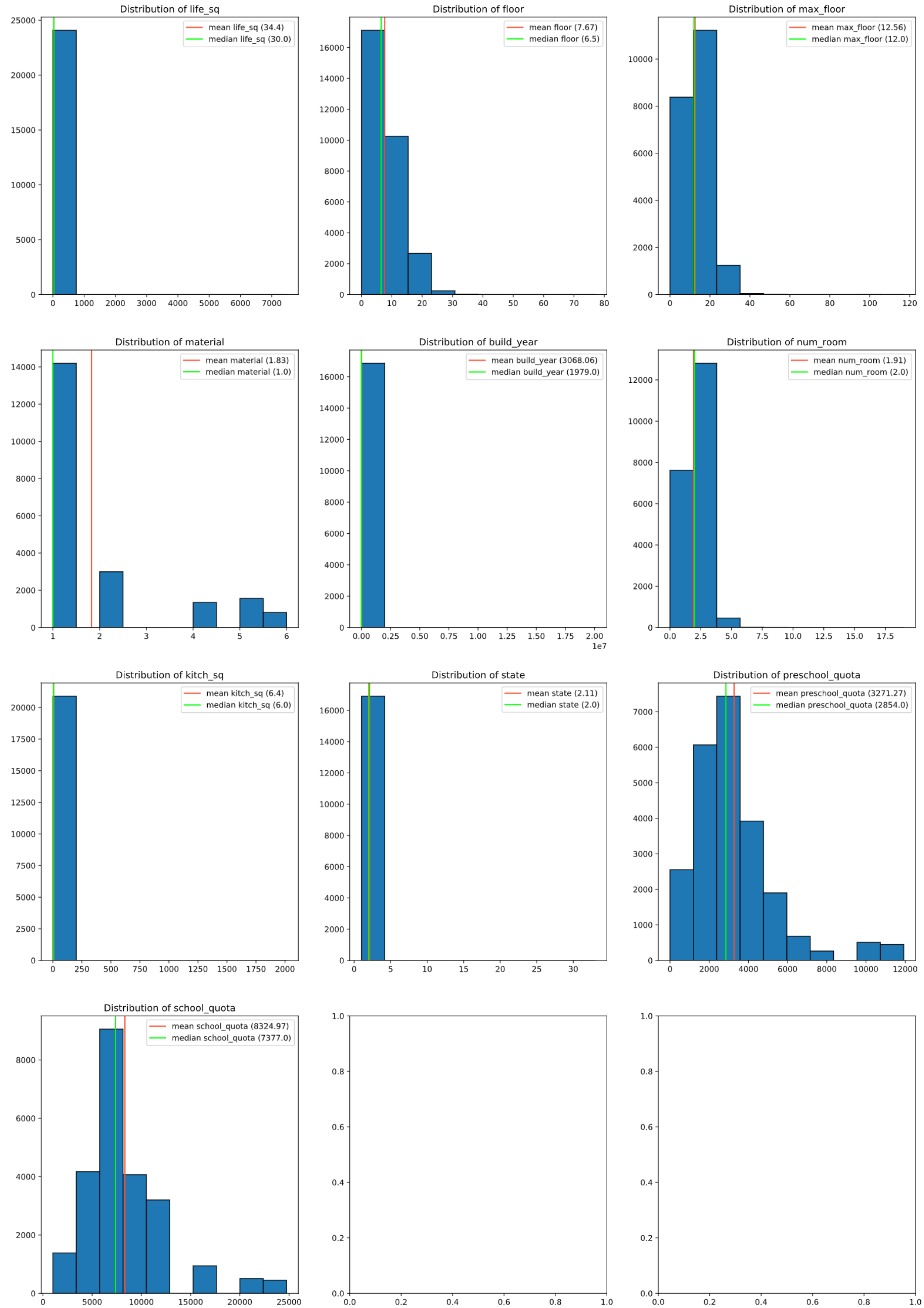
Sberbank, Russia's oldest and largest bank, helps their customers by making predictions about realty prices so renters, developers, and lenders are more confident when they sign a lease or purchase a building. The bank's dataset is composed of 30,471 Russian home prices with 292 continuous and categorical features to inform home price predictions. Approximately 47% of the entire dataset is missing data, distributed over 51 columns.

Outliers are present in a number of factors in this dataset. For example, build_year contains values less than 1000 and greater than 2021. We know these values are likely errors, however for the sake of consistency between our two models, and to maximize the use of all data, we did not remove any outliers. We did prefer a median or mode value for imputation in these cases.

The team broke up the total dataset into four groups for the purpose of analyzing each factor.

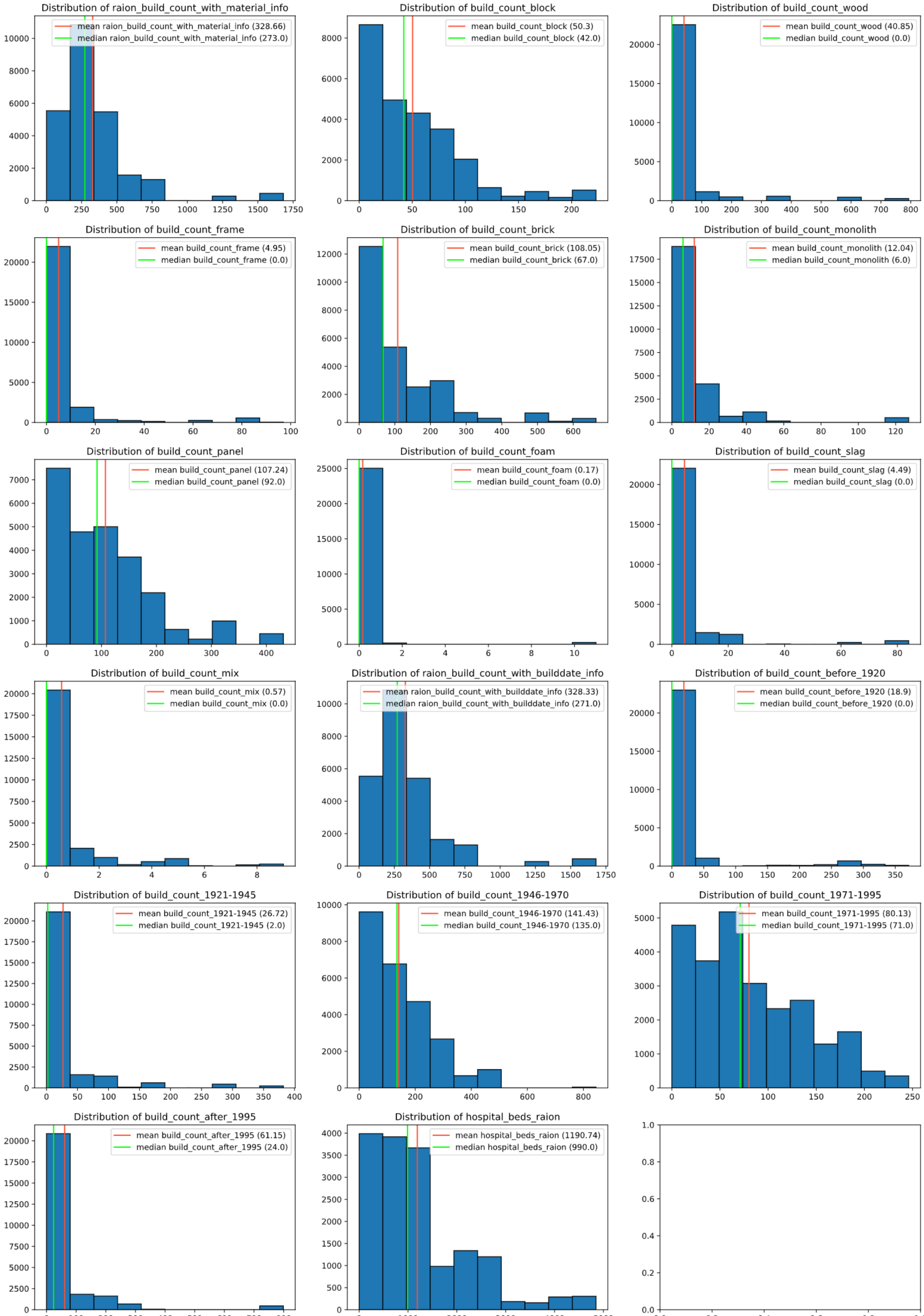
Figures 1 - 4 Display the normality, mean and median for each factor that is not categorical and has missing values.

Group one factors (Figure 1) include life_sq, floor, max_floor, material, build_year, num_room, kitch_sq, state, preschool_quota, school_quota. All columns were imputed using the median except kitch_sq and material. In the case of kitch_sq, the data was normal so we chose to impute missing values using the mean. Material became a categorical value since it represents the type of building material for each property. All other factors could have been perceived as continuous since they are numerical. life_sq, build_year, preschool_quota and school_quota were treated as continuous variables and the amount of skew led us to choose median. In the cases of floor, max_floor, num_room and state, the value must be a whole number in reality, so we chose the median over converting them to categorical.



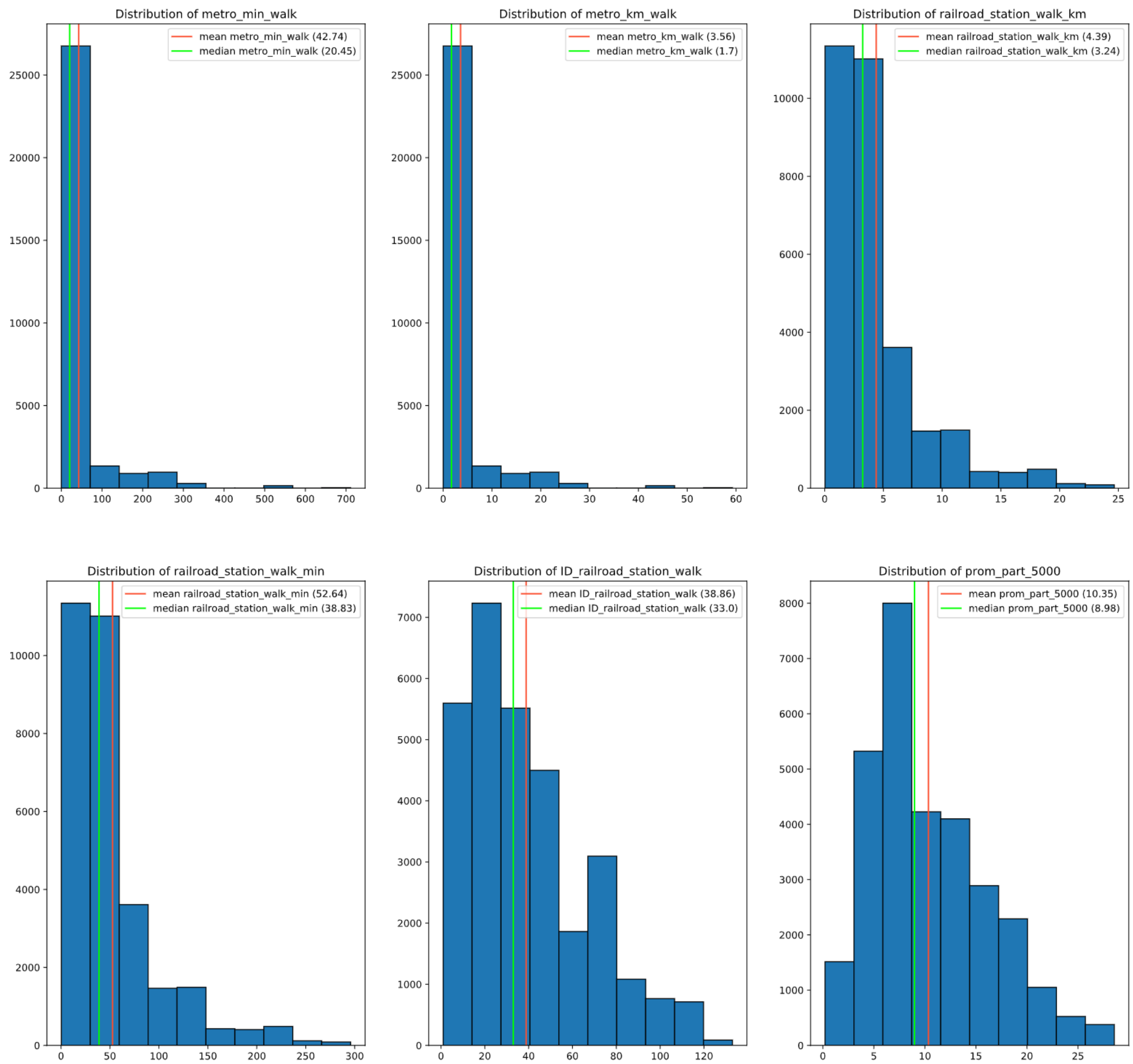
Group 1 Distribution (Figure 1)

Group two columns (Figure 2) include hospital_beds_raion, raion_build_count_with_material_info, build_count_block, build_count_wood, build_count_frame, build_count_brick, build_count_monolith, build_count_panel, build_count_foam, build_count_slag, build_count_mix, raion_build_count_with_builddate_info, build_count_before_1920, build_count_1921-1945, build_count_1946-1970, build_count_1971-1995, and build_count_after_1995. The first column, hospital_beds_raion, has a total of 14441 null values. This column was reviewed in comparison to other columns to identify if there was any form of pattern to when these values were null. This column was identified to be missing at random due to the fact that there was no specific pattern in the remaining columns to identify the data as not missing at random, but it was identified that these values were only null when the values of oil_chemistry_raion and culture_objects_top_25 were valued as "no". The remaining columns, all containing the word "build", all have a total of 4991 null values. Based on the similarity in names and identical number of null values, they were evaluated and it was identified they were all consistently null when the other columns in that group were null. Additionally, the data was evaluated to identify if there was any pattern with values in other features when these values were null and this was not found to be true. Because of these two assessments, build_count_block, build_count_wood, build_count_frame, build_count_brick, build_count_monolith, build_count_panel, build_count_foam, build_count_slag, build_count_mix, raion_build_count_with_builddate_info, build_count_before_1920, build_count_1921-1945, build_count_1946-1970, build_count_1971-1995, and build_count_after_1995 were all found to be missing at random. With all of these columns missing data at random and all being continuous values, the distribution of the data was evaluated to determine normality. All of the columns were non-normal with a skew to the left. Due to this lack of normality, the median values were chosen for the purpose of imputing the missing data.



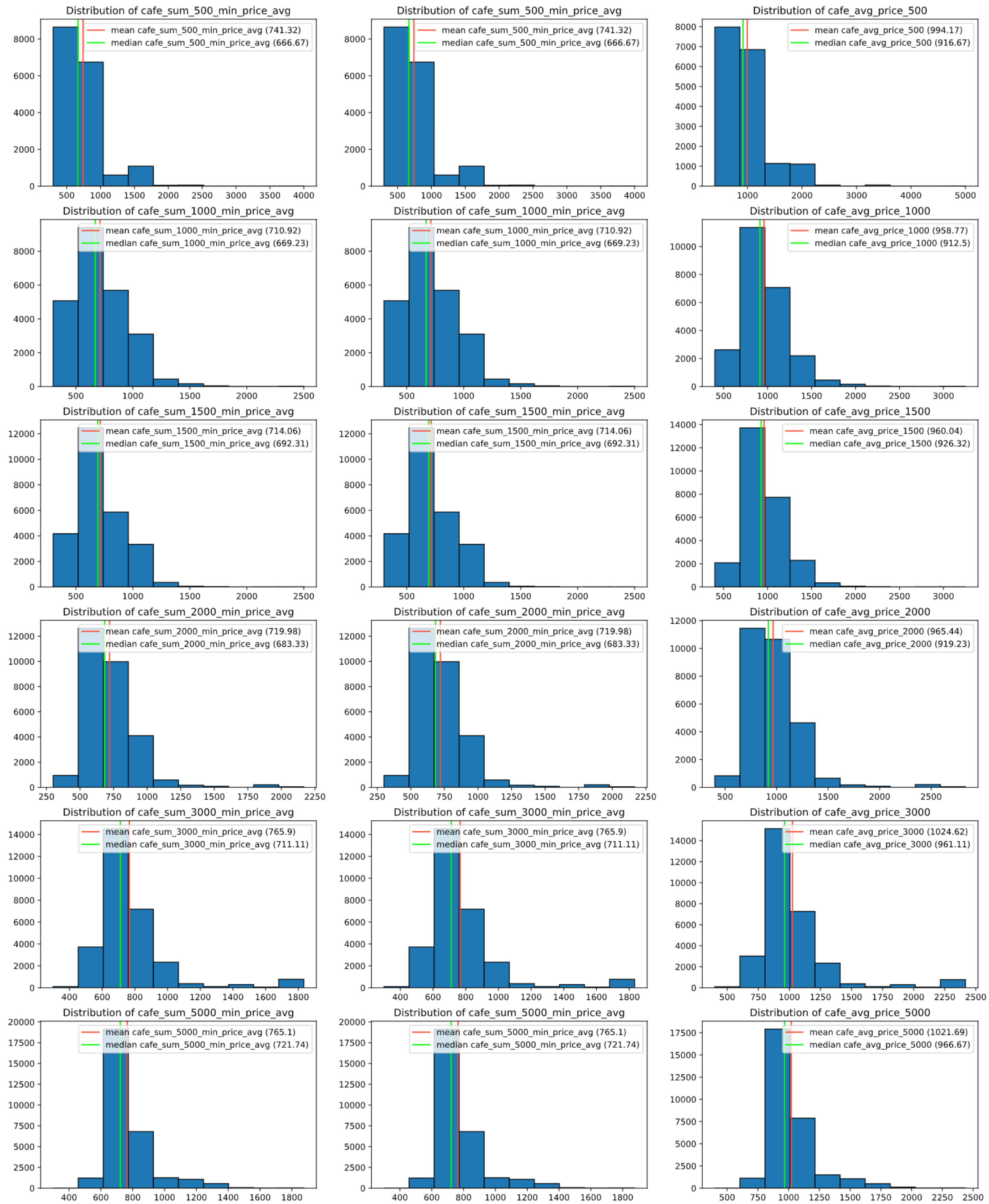
Group 2 Distribution (Figure 2)

In group three (Figure 3), we used 'Median' for prom_part_5000,metro_min_walk,metro_km_walk,railroad_station_walk_km , railroad_station_walk_min because these columns were not normal and we used 'Mode' for ID_railroad_station_walk because this column was a categorical type data.



Group 3 Distribution (Figure 3)

Group four parameters (Figure 4) were largely focused on neighborhood characteristics. There were several groups of three attributes that had the same number of missing values so we suspect they are inter-related. However we do not see the potential source of missingness.



Group 4 Distribution (Figure 4)

Imputation Stategy

Our study is on imputation methods, so we will not delete records from the dataset. Instead, we will explore the dataset for patterns that allow us to interpolate the missing data we observe.

We choose to do iterative imputation using the mean, median or mode technique. We plot these values or all features containing missing values.

The set of columns with missing values was divided between the team members to process. After the imputation technique was choosen for the features, the missing data was replaced with the value of the mean, median or mode.

Model Validation

Models

Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost) is laser focused on computational speed and model performance. (Jason Brownlee; 2016, machine learning mastery) The Sherbank real estate dataset is very large and it has a number of categorical features that must be one-hot encoded to comply with the XGBoost model parameters. This creates a relatively sparse dataset which may lead to overfitting. With this and the features below in mind, we chose XGBoost as our preferred algorithm.

Model Features

Three main forms of gradient boosting are supported:

- Gradient Boosting algorithm also called gradient boosting machine including the learning rate.
- Stochastic Gradient Boosting with sub-sampling at the row, column and column per split levels.
- Regularized Gradient Boosting with both L1 and L2 regularization.

Algorithm Features

Some key algorithm implementation features include:

- Sparse Aware implementation with automatic handling of missing data values.
- Block Structure to support the parallelization of tree construction.
- Continued Training so that you can further boost an already fitted model on new data.

XGBoost Hyperparameter Tuning

To find an optimal combination of hyperparameters for an XGBoost model, a randomized search of combinations was performed to identify the best performing model based on the value of log loss. Each of these hyperparameter combinations was evaluated using 5-fold cross validation of the training data set. The following hyper-paramaters and values were incorporated into the randomized grid search. (Table 1)

To find an optimal combination of hyperparameters for an XGBoost regression model, a randomized search of combinations was performed to identify the best performing model based on the value of log loss. Each of these hyperparameter combinations were evaluated using 5-fold cross validation of the training data set. The following hyper-paramaters and values were incorporated into the randomized grid search.

Hyperparameter	Values
max_depth	6, 10, 15, 20
learning_rate	0.001, 0.01, 0.1, 0.2, 0.3
subsample	0.5, 0.6, 0.7, 0.8, 0.9, 1.0
colsample_bytree	0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
colsample_bylevel	0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
min_child_weight	0.5, 1.0, 3.0, 5.0, 7.0, 10.0
gamma	0, 0.25, 0.5, 1.0
reg_lambda	0.1, 1.0, 5.0, 10.0, 50.0, 100.0
n_estimators	100, 200

XGBoost Hyperparameter Values (Table 1)

The search model selected 20 hyperparameter combinations at random from the list above. With each of these 20 models being evaluated with a 5 cross-fold cross-validation of the dataset, a total of 100 models were evaluated to determine the best-performing combination of hyperparameters. RMSE was used to identify the best-performing model. The model was then used to predict the value of price_doc using the entire dataset.

Results

Factors Containing Missing Data and Imputation Strategy

Table 2 includes a complete list of all factors that contained missing data, the quantity of missing values, the missing values as a percent of all values and whether they are considered MCAR, MAR or MNAR. In addition, the method of imputation is described for each factor.

The greater majority of factors were imputed using the median value. In some cases like num_room and max_floor these could have been handled as categorical values and we might have chosen the mode as the imputed value. However, the range of values for these integer factors was broad enough that we preferred to use the Median to preserve the whole number nature of the attribute. This helped us maintain a more dense matrix which will perform better in the model.

Median was also chosen when the data contained a significant percentage of outliers or when the data was skewed. Rather than normalize the data, we felt that chosing the median would be more resistant to outliers and retain the original shape of the data.

Mode was chosen for all categorical factors: Table 2 Factors and Missing Values

Mean was the least frequently-chosen method of imputation. As we review Table 2, we see that very few continuous factors were normally distributed, which was our primary requirement for imputing with the mean.

Data Description - Factors with Missing Values					
Complete Dataset: (observations, factors)	30471	291			
Factor	# NAs	% of Total	Missingness MCAR, MAR, MNAR	Imputation Method Mean, Median, Mode	Explanation
life_sq	6383	0.2095	MCAR	Median	Right-skewed data, median is less sensitive to outliers
floor	167	0.0055	MCAR	Median	Balanced data with a few outliers, need whole number
max_floor	9572	0.3141	MAR	Median	Balanced data with a few outliers, need whole number
material	9572	0.3141	MAR	Mode	Categorical - number represents a type of building material
build_year	13605	0.4465	MCAR	Median	Several outliers so median is better choice than mean.
num_room	9572	0.3141	MAR	Median	Whole number, using categorical due to sparse data
kitch_sq	9572	0.3141	MAR	Mean	Well balanced data, continuous factor
state	13559	0.4450	MCAR	Median	Similar to factor but numeric. Median is best whole number
preschool_quota	6688	0.2195	MAR	Median	Skewed data, median is less sensitive
school_quota	6685	0.2194	MAR	Median	Skewed data, median is less sensitive
hospital_beds_raion	14441	0.4739	MAR	Median	Right-skewed data, median is less sensitive to outliers
raion_build_count_with_material_info	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_block	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_wood	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_frame	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_brick	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_monolith	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_panel	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_foam	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_slag	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_mix	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
raion_build_count_with_builddate_info	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_before_1920	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_1921-1945	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_1946-1970	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_1971-1995	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
build_count_after_1995	4991	0.1638	MAR	Median	Right-skewed data, median is less sensitive to outliers
metro_min_walk	25	0.0008	MCAR	Median	Right-skewed data, median is less sensitive to outliers
metro_km_walk	25	0.0008	MCAR	Median	Right-skewed data, median is less sensitive to outliers
railroad_station_walk_km	25	0.0008	MCAR	Median	Right-skewed data, median is less sensitive to outliers
railroad_station_walk_min	25	0.0008	MCAR	Median	Right-skewed data, median is less sensitive to outliers
ID_railroad_station_walk	25	0.0008	MCAR	Mode	Categorical - number represents ID of Stations
prom_part_5000	178	0.0058	MCAR	Median	Right-skewed data, median is less sensitive to outliers
cafe_sum_500_min_price_avg	13281	0.4359	MAR	Median	Data is not normally distributed
cafe_sum_500_max_price_avg	13281	0.4359	MAR	Median	Data is not normally distributed
cafe_avg_price_500	13281	0.4359	MAR	Median	Data is not normally distributed
cafe_sum_1000_min_price_avg	6524	0.2141	MAR	Median	Data is not normally distributed
cafe_sum_1000_max_price_avg	6524	0.2141	MAR	Median	Data is not normally distributed
cafe_avg_price_1000	6524	0.2141	MAR	Median	Data is not normally distributed
cafe_sum_1500_min_price_avg	4199	0.1378	MAR	Median	Data is not normally distributed
cafe_sum_1500_max_price_avg	4199	0.1378	MAR	Median	Data is not normally distributed
cafe_avg_price_1500	4199	0.1378	MAR	Median	Data is not normally distributed
cafe_sum_2000_min_price_avg	1725	0.0566	MAR	Median	Data is not normally distributed
cafe_sum_2000_max_price_avg	1725	0.0566	MAR	Median	Data is not normally distributed
cafe_avg_price_2000	1725	0.0566	MAR	Median	Data is not normally distributed
cafe_sum_3000_min_price_avg	991	0.0325	MAR	Median	Data is not normally distributed
cafe_sum_3000_max_price_avg	991	0.0325	MAR	Median	Data is not normally distributed
cafe_avg_price_3000	991	0.0325	MAR	Median	Data is not normally distributed
cafe_sum_5000_min_price_avg	297	0.0097	MAR	Median	Data is not normally distributed
cafe_sum_5000_max_price_avg	297	0.0097	MAR	Median	Data is not normally distributed
cafe_avg_price_5000	297	0.0097	MAR	Median	Data is not normally distributed

Factors and Missing Values (Table 2)

Conclusion

Mean is not always the best approach

Until this project, we frequently assumed that the mean value of a factor might be the best value to use as a replacement for null values. Now that we see how many cases warrant the use of the median value, we will be more thoughtful in our imputation methods.

Factor vs. Numeric and the use of Median

It is possible to maintain a more dense or less sparse dataset by using the median to impute a set of values that should remain whole numbers. One may be tempted to convert a score or a class value to a factor then use one-hot encoding to impute missing values and prepare for modeling. However, it is just as viable to retain the original data in tall format and use median to capture the mid-point of the dataset.

Imputing with Mean/Median/Mode vs. -1

Model 1 imputed all missing values with -1. It yielded an RMSE of 1736552.25 Model 2 included imputed values. It yielded an RMSE of 1755174.8

Because the grid search process uses a randomized approach, we feel that these numbers are not significantly different. However, the lack of difference is significant conclusion. We expected to see a greater improvement in RMSE when values were imputed using mean, median and mode, and in the final result the Model 1 using -1 was similarly effective.

This project taught us that thoughtful imputation of missing values may have little effect on the accuracy of a model. We see little difference in RMSE between Model 1 and Model 2. This may be due to our choice of modeling algorithm. We know that XGBoost calculates gain and it is less sensitive to the importance of one or more specific values.

In the future it may be wise to run models using -1 or a placeholder value, then see how well the model performs without investing time to impute in a detailed approach. Another consideration may be the type of model chosen. Other algorigthms may be more sensitive to imputation. Different datasets may also likely be more affected by mean/median/mode than this dataset. Certainly smaller datasets will rely on each value more than we saw in this large dataset. There is no conclusion we can draw from this exercise other than the fact that for this model and with this dataset we did not see a significant impact from detailed imputation.

References

CJason Brownlee (2016), machine learning mastery - A Gentle Introduction to XGBoost for Applied Machine Learning.

Appendix

Code

In [1]:
```
import pandas as pd
import numpy as np
import os
import pickle
```

Appendix

Code

In [1]:

```
import pandas as pd
import numpy as np
import os
import pickle
import math
import missingno as msnofrom
import statistics as stats
from matplotlib import pyplot as plt
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
import numpy as np
from sklearn.metrics import mean_squared_error as MSE
```

In [2]:

```
# Set Directory for image files - Comment out if you are not LL or MM
#os.chdir("C://Users/18322/OneDrive - Southern Methodist University/Deskto
p/QOW/Case Study 10")
os.chdir('C:\\SMU_Local')
```

In [4]:

```
# import csv to data
data = pd.read_csv('./data/qtw/CS10/train.csv')
# create target (y)
target = data.price_doc
# drop target from data
data.drop(['price_doc'],inplace=True,axis=1)
# convert timestamp to int YYYYMMDD
data.timestamp = pd.to_datetime(data.timestamp).apply(lambda x: x.strftime
('%Y%m%d')).astype(int)
# create separate copy for -1 imputation
data_neg_one = data.copy()
id=data["id"]
# Shape of dataframe
data.shape
```

Out[4]: (30471, 291)

In [19]:

```
# Dataframe description
data.describe()
```

Out[19]:

	id	timestamp	full_sq	life_sq	floor	max_floor	ma
count	30471.000000	3.047100e+04	30471.000000	24088.000000	30304.000000	20899.000000	20899.00
mean	15237.917397	2.013522e+07	54.214269	34.403271	7.670803	12.558974	1.82
std	8796.501536	9.530639e+03	38.031487	52.285733	5.319989	6.756550	1.48
min	1.000000	2.011082e+07	0.000000	0.000000	0.000000	0.000000	1.00

25%	7620.500000	2.013042e+07	38.000000	20.000000	3.000000	9.000000	1.00
50%	15238.000000	2.014022e+07	49.000000	30.000000	6.500000	12.000000	1.00
75%	22855.500000	2.014092e+07	63.000000	43.000000	11.000000	17.000000	2.00
max	30473.000000	2.015063e+07	5326.000000	7478.000000	77.000000	117.000000	6.00

8 rows × 276 columns

In [20]:
```
# Count null values by column
data.isnull().sum(axis = 0)
```

Out[20]:
```
id                          0
timestamp                   0
full_sq                     0
life_sq                  6383
floor                     167
                         ...
church_count_5000           0
mosque_count_5000           0
leisure_count_5000          0
sport_count_5000            0
market_count_5000           0
Length: 291, dtype: int64
```

In [21]:
```
# summarize the number of rows with missing values for each column
def my_function(NAS):
    for i in data.columns:
        if data.loc[data[i].isna(),i].shape[0]>0:
            print(i,data.loc[data[i].isna(),i].shape)
```

In [22]:
```
##########################################################
####    Team 1 Column Analysis for Missing Data       ####
##########################################################
```

In [8]:
```
# Call function to list all columns with count of nulls greater that zero
and count them up
my_function(data)
```

```
life_sq (6383,)
floor (167,)
max_floor (9572,)
material (9572,)
build_year (13605,)
num_room (9572,)
kitch_sq (9572,)
state (13559,)
preschool_quota (6688,)
school_quota (6685,)
hospital_beds_raion (14441,)
raion_build_count_with_material_info (4991,)
build_count_block (4991,)
build_count_wood (4991,)
build_count_frame (4991,)
build_count_brick (4991,)
```

```
build_count_monolith (4991,)
build_count_panel (4991,)
build_count_foam (4991,)
build_count_slag (4991,)
build_count_mix (4991,)
raion_build_count_with_builddate_info (4991,)
build_count_before_1920 (4991,)
build_count_1921-1945 (4991,)
build_count_1946-1970 (4991,)
build_count_1971-1995 (4991,)
build_count_after_1995 (4991,)
metro_min_walk (25,)
metro_km_walk (25,)
railroad_station_walk_km (25,)
railroad_station_walk_min (25,)
ID_railroad_station_walk (25,)
cafe_sum_500_min_price_avg (13281,)
cafe_sum_500_max_price_avg (13281,)
cafe_avg_price_500 (13281,)
cafe_sum_1000_min_price_avg (6524,)
cafe_sum_1000_max_price_avg (6524,)
cafe_avg_price_1000 (6524,)
cafe_sum_1500_min_price_avg (4199,)
cafe_sum_1500_max_price_avg (4199,)
cafe_avg_price_1500 (4199,)
cafe_sum_2000_min_price_avg (1725,)
cafe_sum_2000_max_price_avg (1725,)
cafe_avg_price_2000 (1725,)
cafe_sum_3000_min_price_avg (991,)
cafe_sum_3000_max_price_avg (991,)
cafe_avg_price_3000 (991,)
prom_part_5000 (178,)
cafe_sum_5000_min_price_avg (297,)
cafe_sum_5000_max_price_avg (297,)
cafe_avg_price_5000 (297,)
```

In [10]:

```
# Analysis of time and distance to railroad and metro stations. Also, extr
acting the number of industrial part areas within 5000 km
# metro_min_walk                >>>>>Time to metro by foot
# metro_km_walk             >>>>>Distance to the metro, km
# railroad_station_walk_km      >>>>Distance to the railroad station (walk
)
# railroad_station_walk_min     >>>>Time to the railroad station (walk)
# ID_railroad_station_walk      >>>>Nearest railroad station id (walk)
# prom_part_5000                >>>>The share of industrial zones in 5000
meters zone
team1_columns_all = ['metro_min_walk','metro_km_walk','railroad_station_wa
lk_km','railroad_station_walk_min','ID_railroad_station_walk','prom_part_5
000']
df=data[team1_columns_all]
```

In [11]:

```
# List of then null counts per column in this subset
for i in df.columns:
    if df.loc[df[i].isna(),i].shape[0]>0:
        print(i,df.loc[df[i].isna(),i].shape)
```

```
metro_min_walk (25,)
metro_km_walk (25,)
railroad_station_walk_km (25,)
railroad_station_walk_min (25,)
ID_railroad_station_walk (25,)
prom_part_5000 (178,)
```

In [12]:
```
#Finding missing % info in entire dataset
missing_value_precentage=pd.concat([(data.isnull().sum()/len(data))*100],axis=1,keys=['Precentage_of_missing_Info_in_each_column'])
missing_value_precentage.sort_values(ascending=False,by="Precentage_of_missing_Info_in_each_column")
```

Out[12]:

	Precentage_of_missing_Info_in_each_column
hospital_beds_raion	47.392603
build_year	44.649011
state	44.498047
cafe_avg_price_500	43.585704
cafe_sum_500_max_price_avg	43.585704
...	...
ID_bus_terminal	0.000000
oil_chemistry_km	0.000000
nuclear_reactor_km	0.000000
radiation_km	0.000000
market_count_5000	0.000000

291 rows × 1 columns

In [13]:
```
#Finding missing % in selected columns for walk and share of industrial zones in 5000 meters zone
missing_value_precentage=pd.concat([(df.isnull().sum()/len(data))*100],axis=1,keys=['Precentage_missing_Info_in_each_column'])
missing_value_precentage.sort_values(ascending=False,by="Precentage_missing_Info_in_each_column")
```

Out[13]:

	Precentage_missing_Info_in_each_column
prom_part_5000	0.584162
metro_min_walk	0.082045
metro_km_walk	0.082045
railroad_station_walk_km	0.082045
railroad_station_walk_min	0.082045
ID_railroad_station_walk	0.082045

In [14]:
```
###-Rules to impute-####

# Data Normally distributed try mean
# Data Categorical try Mode ( ID_Railroad_station_walk_km)
# Data Non-Normal try median( prom, metro_min, metro_km,rail_wlk_km,rail_w
alk_min)
# Multiple variables are correlated , either drop the column with missing
data or perform a fit to predict missing data.
# flag value missing: try -1, 0, -100 could allow the model to learn how t
o deal with missing data
```

In [15]:
```
# Prepare data by first dropping all na values ant then create seperat dat
aframe files for each column
new_df=df.dropna()
prom_part_5000=new_df['prom_part_5000']
metro_min_walk=new_df['metro_min_walk']
metro_km_walk=new_df['metro_km_walk']
railroad_station_walk_km=new_df['railroad_station_walk_km']
railroad_station_walk_min=new_df['railroad_station_walk_min']
ID_railroad_station_walk=new_df['ID_railroad_station_walk']
```

In [35]:
```
# Print out the statistical values for each column
mean_prom=stats.mean(prom_part_5000)
median_prom=stats.median(prom_part_5000)
mode_prom=stats.mode(prom_part_5000)
print('mean_prom',mean_prom)
print('median_prom',median_prom)
print('mode_prom',mode_prom)
mean_metro_min_walk=stats.mean(metro_km_walk)
median_metro_min_walk=stats.median(metro_min_walk)
mode_metro_min_walk=stats.mode(metro_min_walk)
print('mean_metro_min_walk',mean_metro_min_walk)
print('median_metro_min_walk',median_metro_min_walk)
print('mode_metro_min_walk',mode_metro_min_walk)
mean_metro_km_walk=stats.mean(metro_km_walk)
median_metro_km_walk=stats.median(metro_km_walk)
mode_metro_km_walk=stats.mode(metro_km_walk)
print('mean_metro_km_walk',mean_metro_km_walk)
print('median_metro_km_walk',median_metro_km_walk)
print('mode_metro_km_walk',mode_metro_km_walk)
mean_railroad_station_walk_km=stats.mean(railroad_station_walk_km)
median_railroad_station_walk_km=stats.median(railroad_station_walk_km)
mode_railroad_station_walk_km=stats.mode(railroad_station_walk_km)
print('mean_railroad_station_walk_km',mean_railroad_station_walk_km)
print('median_railroad_station_walk_km',median_railroad_station_walk_km)
print('mode_railroad_station_walk_km',mode_railroad_station_walk_km)
mean_railroad_station_walk_min=stats.mean(railroad_station_walk_min)
median_railroad_station_walk_min=stats.median(railroad_station_walk_min)
mode_railroad_station_walk_min=stats.mode(railroad_station_walk_min)
print('mean_railroad_station_walk_min',mean_railroad_station_walk_min)
print('median_railroad_station_walk_min',median_railroad_station_walk_min)
print('mode_railroad_station_walk_min',mode_railroad_station_walk_min)
mode_ID_railroad_station_walk=stats.mode(ID_railroad_station_walk)
print('mode_ID_railroad_station_walk',mode_ID_railroad_station_walk)
```

```
mean_prom 10.34879641865997
median_prom 8.97
mode_prom 6.54
mean_metro_min_walk 3.3087766777142855
median_metro_min_walk 20.324236305
mode_metro_min_walk 45.3220315
mean_metro_km_walk 3.3087766777142855
median_metro_km_walk 1.6936863585
mode_metro_km_walk 3.7768359589999996
mean_railroad_station_walk_km 4.3850010627970795
median_railroad_station_walk_km 3.242133743
mode_railroad_station_walk_km 1.923494882
mean_railroad_station_walk_min 52.620012754666945
median_railroad_station_walk_min 38.905604915
mode_railroad_station_walk_min 23.08193859
mode_ID_railroad_station_walk 24.0
```

In [36]:

```
# this function takes a dataframe and a list of columns and outputs a 3-co
lumn grid of distributions, with mean and median identified
def distribution_grid(dataframe, columns):
    # identify # of plots to be created
    plots = len(columns)
    # define number of columns in grid
    cols = 3
    # figure out number of rows required based on number of plots and numb
er of columns
    rows = plots // cols + 1
    # create grid
    fig, axs = plt.subplots(rows,cols,figsize=(20,30))
    # loop through columns by index
    for k in range(plots):
        col = columns[k]
        # get grid row value for the histogram for this loop
        x = math.floor(k/cols)
        # get grid column value for the histogram for this loop
        y = k % cols
        # generate histogram, title, and mean/median lines
        axs[x,y].hist(dataframe[col], bins=10,edgecolor='black')
        axs[x,y].set_title(f'Distribution of {col}')
        axs[x,y].axvline(dataframe[col].mean(), color='#fc4f30',label=f'me
an {col} ({round(data[col].mean(),2)})')
        axs[x,y].axvline(dataframe[col].median(), color='#00FF00',label=f'
median {col} ({round(data[col].median(),2)})')
        axs[x,y].legend()
    plt.show()
```

In [37]:

```
# call function and pass in dataframe and my list of columns
distribution_grid(data,team1_columns_all)
```

*{stroke-linecap:butt;stroke-linejoin:round;}

In [39]:

```
##Rules that applied here##
##Data Categorical try Mode(ID_Railroad_station_walk_km)
#Data Non-Normal try median(prom, metro_min, metro_km, rail_wlk_km, rail_w
alk_min)
ndf=df.fillna({'metro_min_walk':median_metro_min_walk,'metro_km_walk': med
```

```
ian_metro_km_walk,'railroad_station_walk_km':median_railroad_station_walk_
km,'railroad_station_walk_min':median_railroad_station_walk_min,'ID_railro
ad_station_walk':mode_ID_railroad_station_walk,'prom_part_5000':median_pro
m})
```

In [40]:
```
#Make sure there are no more na
ndf.isnull().sum()
```

Out[40]:
```
metro_min_walk              0
metro_km_walk               0
railroad_station_walk_km    0
railroad_station_walk_min   0
ID_railroad_station_walk    0
prom_part_5000              0
dtype: int64
```

In [ ]:
```
##########################################################
####    Team 2 Column Analysis for Missing Data       ####
##########################################################
```

In [41]:
```
# Beginning of analysis
# create lists of build columns, non-build columns, and a combined list
team2_columns_build = ['raion_build_count_with_material_info','build_count
_block','build_count_wood','build_count_frame','build_count_brick','build_
count_monolith','build_count_panel','build_count_foam','build_count_slag',
'build_count_mix','raion_build_count_with_builddate_info','build_count_bef
ore_1920','build_count_1921-1945','build_count_1946-1970','build_count_197
1-1995','build_count_after_1995']

team2_columns_non_build = ['hospital_beds_raion']

team2_columns_all = team2_columns_build + team2_columns_non_build
```

In [42]:
```
# loop through each of the build columns and identify if the other build c
olumns are all null at the same time, a zero output confirms this to be tr
ue
build_column_status = {}
for col in team2_columns_build:
    build_column_status[col] = data[data[col].isnull()][team2_columns_buil
d].dropna(how='all').shape[0]
print(build_column_status)
```

```
{'raion_build_count_with_material_info': 0, 'build_count_block': 0, 'build
_count_wood': 0, 'build_count_frame': 0, 'build_count_brick': 0, 'build_co
unt_monolith': 0, 'build_count_panel': 0, 'build_count_foam': 0, 'build_co
unt_slag': 0, 'build_count_mix': 0, 'raion_build_count_with_builddate_info
': 0, 'build_count_before_1920': 0, 'build_count_1921-1945': 0, 'build_cou
nt_1946-1970': 0, 'build_count_1971-1995': 0, 'build_count_after_1995': 0}
```

In [43]:
```
data_build_na_true = data[data.build_count_after_1995.isnull() == True]
data_build_na_false = data[data.build_count_after_1995.isnull() == True]
build_cols_na_nunique = data_build_na_true.nunique()
print(build_cols_na_nunique[build_cols_na_nunique == 1])
```

```
preschool_education_centers_raion        1
school_education_centers_raion           1
```

```
school_education_centers_top_20_raion    1
hospital_beds_raion                      1
healthcare_centers_raion                 1
university_top_20_raion                  1
culture_objects_top_25                   1
culture_objects_top_25_raion             1
thermal_power_plant_raion                1
incineration_raion                       1
oil_chemistry_raion                      1
radiation_raion                          1
railroad_terminal_raion                  1
nuclear_reactor_raion                    1
detention_facility_raion                 1
ecology                                  1
cafe_count_500_na_price                  1
cafe_count_500_price_high                1
mosque_count_500                         1
leisure_count_500                        1
market_count_500                         1
cafe_count_1000_price_high               1
mosque_count_1000                        1
leisure_count_1000                       1
market_count_1000                        1
cafe_count_1500_price_high               1
mosque_count_1500                        1
leisure_count_1500                       1
market_count_1500                        1
cafe_count_2000_price_high               1
mosque_count_2000                        1
leisure_count_2000                       1
cafe_count_3000_price_high               1
leisure_count_3000                       1
cafe_count_5000_price_high               1
dtype: int64
```

In [ ]:

```
# This function moved to team 1 content
""" # this function takes a dataframe and a list of columns and outputs a 
3-column grid of distributions, with mean and median identified
def distribution_grid(dataframe, columns):
    # identify # of plots to be created
    plots = len(columns)
    # define number of columns in grid
    cols = 3
    # figure out number of rows required based on number of plots and numb
er of columns
    rows = plots // cols + 1
    # create grid
    fig, axs = plt.subplots(rows,cols,figsize=(20,30))
    # loop through columns by index
    for k in range(plots):
        col = columns[k]
        # get grid row value for the histogram for this loop
        x = math.floor(k/cols)
        # get grid column value for the histogram for this loop
        y = k % cols
        # generate histogram, title, and mean/median lines
        axs[x,y].hist(dataframe[col], bins=10,edgecolor='black')
```

```
        axs[x,y].set_title(f'Distribution of {col}')
        axs[x,y].axvline(dataframe[col].mean(), color='#fc4f30',label=f'me
an {col} ({round(data[col].mean(),2)})')
        axs[x,y].axvline(dataframe[col].median(), color='#00FF00',label=f'
median {col} ({round(data[col].median(),2)})')
        axs[x,y].legend()
    plt.show() """
```

In [44]:
```
# call function and pass in dataframe and my list of columns
distribution_grid(data,team2_columns_all)
```

*{stroke-linecap:butt;stroke-linejoin:round;}

In [45]:
```
'''
Because all are missing at random and not normally distributed, generate d
ictionary of NA fill values by looping through columns as dictionary key a
nd inserting median as the value.  If not all of your columns are this sam
e situation, the dictionary will need to be build manually.
'''
team2_fillna_dict = {}
for col in team2_columns_all:
    team2_fillna_dict[col] = data[col].median()
print(team2_fillna_dict)
```

```
{'raion_build_count_with_material_info': 273.0, 'build_count_block': 42.0,
 'build_count_wood': 0.0, 'build_count_frame': 0.0, 'build_count_brick': 6
7.0, 'build_count_monolith': 6.0, 'build_count_panel': 92.0, 'build_count_
foam': 0.0, 'build_count_slag': 0.0, 'build_count_mix': 0.0, 'raion_build_
count_with_builddate_info': 271.0, 'build_count_before_1920': 0.0, 'build_
count_1921-1945': 2.0, 'build_count_1946-1970': 135.0, 'build_count_1971-1
995': 71.0, 'build_count_after_1995': 24.0, 'hospital_beds_raion': 990.0}
```

In [46]:
```
# code below tests dictionary and confirms no nulls
new_data = data.fillna(team2_fillna_dict)
new_data[team2_columns_all].isnull().sum()
```

Out[46]:
```
raion_build_count_with_material_info     0
build_count_block                        0
build_count_wood                         0
build_count_frame                        0
build_count_brick                        0
build_count_monolith                     0
build_count_panel                        0
build_count_foam                         0
build_count_slag                         0
build_count_mix                          0
raion_build_count_with_builddate_info    0
build_count_before_1920                  0
build_count_1921-1945                    0
build_count_1946-1970                    0
build_count_1971-1995                    0
build_count_after_1995                   0
hospital_beds_raion                      0
dtype: int64
```

In [ ]:

In [47]:
```
############################################################
####    Team 3 Column Analysis for Missing Data       ####
############################################################

# Beginning of analysis
# create lists of build columns, non-build columns, and a combined list
team3_columns_build = ['life_sq','floor','max_floor','material','build_yea
r','num_room','kitch_sq','state','preschool_quota','school_quota']


# loop through each of the build columns and identify if the other build c
olumns are all null at the same time, a zero output confirms this to be tr
ue
build_column_status = {}
for col in team3_columns_build:
    build_column_status[col] = data[data[col].isnull()][team3_columns_buil
d].dropna(how='all').shape[0]
print(build_column_status)


# this function takes a dataframe and a list of columns and outputs a 3-co
lumn grid of distributions, with mean and median identified
def distribution_grid(dataframe, columns):
    # identify # of plots to be created
    plots = len(columns)
    # define number of columns in grid
    cols = 3
    # figure out number of rows required based on number of plots and numb
er of columns
    rows = plots // cols + 1
    # create grid
    fig, axs = plt.subplots(rows,cols,figsize=(20,30))
    # loop through columns by index
    for k in range(plots):
        col = columns[k]
        # get grid row value for the histogram for this loop
        x = math.floor(k/cols)
        # get grid column value for the histogram for this loop
        y = k % cols
        # generate histogram, title, and mean/median lines
        axs[x,y].hist(dataframe[col], bins=10,edgecolor='black')
        axs[x,y].set_title(f'Distribution of {col}')
        axs[x,y].axvline(dataframe[col].mean(), color='#fc4f30',label=f'me
an {col} ({round(data[col].mean(),2)})')
        axs[x,y].axvline(dataframe[col].median(), color='#00FF00',label=f'
median {col} ({round(data[col].median(),2)})')
        axs[x,y].legend()
    plt.show()

# Call function defined in team 2 and pass in dataframe and my list of col
umns
distribution_grid(data,team3_columns_build)

'''
Because all are missing at random and not normally distributed, generate d
ictionary of NA fill values by looping through columns as dictionary key a
```

```
nd inserting median as the value.  If not all of your columns are this sam
e situation, the dictionary will need to be build manually.
'''
team3_fillna_dict = {}
for col in team3_columns_build:
    team3_fillna_dict[col] = data[col].median()
print(team3_fillna_dict)

# code below tests dictionary and confirms no nulls
new_data = data.fillna(team3_fillna_dict)
new_data[team3_columns_build].isnull().sum()
```

```
{'life_sq': 6360, 'floor': 144, 'max_floor': 9549, 'material': 9549, 'buil
d_year': 13582, 'num_room': 9549, 'kitch_sq': 9549, 'state': 13536, 'presc
hool_quota': 6665, 'school_quota': 6662}
```

*{stroke-linecap:butt;stroke-linejoin:round;}

```
{'life_sq': 30.0, 'floor': 6.5, 'max_floor': 12.0, 'material': 1.0, 'build
_year': 1979.0, 'num_room': 2.0, 'kitch_sq': 6.0, 'state': 2.0, 'preschool
_quota': 2854.0, 'school_quota': 7377.0}
```

Out[47]:
```
life_sq            0
floor              0
max_floor          0
material           0
build_year         0
num_room           0
kitch_sq           0
state              0
preschool_quota    0
school_quota       0
dtype: int64
```

In [18]:
```
# LL NEW
#  The factor 'material' is categorical because each numerical value repre
sents a type of building material.
#  Calculate the mode of material for our imputation dictionary

mode_material=stats.mode(data['material'])
mode_material
```

Out[18]:
```
1.0
```

In [48]:
```
###########################################################
####    Team 4 Column Analysis for Missing Data       ####
###########################################################

# Beginning of analysis
# create lists of build columns, non-build columns, and a combined list

team4_columns_build = ['cafe_sum_500_min_price_avg','cafe_sum_500_min_pric
e_avg','cafe_avg_price_500','cafe_sum_1000_min_price_avg','cafe_sum_1000_m
in_price_avg','cafe_avg_price_1000','cafe_sum_1500_min_price_avg','cafe_su
m_1500_min_price_avg','cafe_avg_price_1500','cafe_sum_2000_min_price_avg',
'cafe_sum_2000_min_price_avg','cafe_avg_price_2000','cafe_sum_3000_min_pri
ce_avg','cafe_sum_3000_min_price_avg','cafe_avg_price_3000','cafe_sum_5000
```

```
_min_price_avg','cafe_sum_5000_min_price_avg','cafe_avg_price_5000']

# loop through each of the build columns and identify if the other build c
olumns are all null at the same time, a zero output confirms this to be tr
ue
build_column_status = {}
for col in team4_columns_build:
    build_column_status[col] = data[data[col].isnull()][team4_columns_buil
d].dropna(how='all').shape[0]
print(build_column_status)



# this function takes a dataframe and a list of columns and outputs a 3-co
lumn grid of distributions, with mean and median identified
def distribution_grid(dataframe, columns):
    # identify # of plots to be created
    plots = len(columns)
    # define number of columns in grid
    cols = 3
    # figure out number of rows required based on number of plots and numb
er of columns
    rows = plots // cols + 1
    # create grid
    fig, axs = plt.subplots(rows,cols,figsize=(20,30))
    # loop through columns by index
    for k in range(plots):
        col = columns[k]
        # get grid row value for the histogram for this loop
        x = math.floor(k/cols)
        # get grid column value for the histogram for this loop
        y = k % cols
        # generate histogram, title, and mean/median lines
        axs[x,y].hist(dataframe[col], bins=10,edgecolor='black')
        axs[x,y].set_title(f'Distribution of {col}')
        axs[x,y].axvline(dataframe[col].mean(), color='#fc4f30',label=f'me
an {col} ({round(data[col].mean(),2)})')
        axs[x,y].axvline(dataframe[col].median(), color='#00FF00',label=f'
median {col} ({round(data[col].median(),2)})')
        axs[x,y].legend()
    plt.show()

# Call function defined in team 2 and pass in dataframe and my list of col
umns
distribution_grid(data,team4_columns_build)

'''
Because all are missing at random and not normally distributed, generate d
ictionary of NA fill values by looping through columns as dictionary key a
nd inserting median as the value.  If not all of your columns are this sam
e situation, the dictionary will need to be build manually.
'''
team4_fillna_dict = {}
for col in team4_columns_build:
    team4_fillna_dict[col] = data[col].median()
print(team4_fillna_dict)
```

```
# code below tests dictionary and confirms no nulls
new_data = data.fillna(team4_fillna_dict)
new_data[team4_columns_build].isnull().sum()
```

```
{'cafe_sum_500_min_price_avg': 12984, 'cafe_avg_price_500': 12984, 'cafe_s
um_1000_min_price_avg': 6227, 'cafe_avg_price_1000': 6227, 'cafe_sum_1500_
min_price_avg': 3902, 'cafe_avg_price_1500': 3902, 'cafe_sum_2000_min_pric
e_avg': 1428, 'cafe_avg_price_2000': 1428, 'cafe_sum_3000_min_price_avg':
694, 'cafe_avg_price_3000': 694, 'cafe_sum_5000_min_price_avg': 0, 'cafe_a
vg_price_5000': 0}
```

*{stroke-linecap:butt;stroke-linejoin:round;}

```
{'cafe_sum_500_min_price_avg': 666.67, 'cafe_avg_price_500': 916.67, 'cafe
_sum_1000_min_price_avg': 669.23, 'cafe_avg_price_1000': 912.5, 'cafe_sum_
1500_min_price_avg': 692.31, 'cafe_avg_price_1500': 926.32, 'cafe_sum_2000
_min_price_avg': 683.33, 'cafe_avg_price_2000': 919.23, 'cafe_sum_3000_min
_price_avg': 711.11, 'cafe_avg_price_3000': 961.11, 'cafe_sum_5000_min_pri
ce_avg': 721.74, 'cafe_avg_price_5000': 966.67}
```

Out[48]:

```
cafe_sum_500_min_price_avg     0
cafe_sum_500_min_price_avg     0
cafe_avg_price_500             0
cafe_sum_1000_min_price_avg    0
cafe_sum_1000_min_price_avg    0
cafe_avg_price_1000            0
cafe_sum_1500_min_price_avg    0
cafe_sum_1500_min_price_avg    0
cafe_avg_price_1500            0
cafe_sum_2000_min_price_avg    0
cafe_sum_2000_min_price_avg    0
cafe_avg_price_2000            0
cafe_sum_3000_min_price_avg    0
cafe_sum_3000_min_price_avg    0
cafe_avg_price_3000            0
cafe_sum_5000_min_price_avg    0
cafe_sum_5000_min_price_avg    0
cafe_avg_price_5000            0
dtype: int64
```

In [ ]:

```
##########################################################
####     XGBoost Configuration and Execution.         ####
##########################################################
```

In [49]:

```
# Define a parameter grid to pass to RandomizedSearchCV
param_grid = {
        'silent': [False],
        'max_depth': [6, 10, 15, 20],
        'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.3],
        'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'colsample_bytree': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'colsample_bylevel': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'min_child_weight': [0.5, 1.0, 3.0, 5.0, 7.0, 10.0],
        'gamma': [0, 0.25, 0.5, 1.0],
        'reg_lambda': [0.1, 1.0, 5.0, 10.0, 50.0, 100.0],
        'n_estimators': [100, 200]}
```

In [50]:
```
# XGBoost wrapped into a function so it can be used on both datasets
# define function to do randomsearch
def xgb_randomsearch(xgb_df, xgb_y, n_iter, param_grid):
    # intiate XGBoost Regressor
    xgbreg = xgb.XGBRegressor()
    # define RandomSearchCV with input # of iterations, all available CPUs
, verbose output, scoring based on negative RMSE, 5 fold CV, and refitting
 with best parameters
    xgb_rs_reg = RandomizedSearchCV(xgbreg, param_grid, n_iter=n_iter, n_j
obs=-1, verbose=2, scoring='neg_root_mean_squared_error',cv=5, refit=True,
 random_state=123)
    # return fit model
    return xgb_rs_reg.fit(xgb_df, xgb_y)
```

In [ ]:
```
###########################################################
####     Model XGBoost Null Values set -1.             ####
###########################################################
```

In [51]:
```
## DEFINE -1 IMPUTED DATASET

###### MAKE data_neg_one defined when loading data at the beginning of fil
e #######
data_neg_one.fillna(value=-1, inplace=True)
data_neg_one.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30471 entries, 0 to 30470
Data columns (total 291 columns):
 #    Column                                  Dtype
---   ------                                  -----
 0    id                                      int64
 1    timestamp                               int64
 2    full_sq                                 int64
 3    life_sq                                 float64
 4    floor                                   float64
 5    max_floor                               float64
 6    material                                float64
 7    build_year                              float64
 8    num_room                                float64
 9    kitch_sq                                float64
 10   state                                   float64
 11   product_type                            object
 12   sub_area                                object
 13   area_m                                  float64
 14   raion_popul                             int64
 15   green_zone_part                         float64
 16   indust_part                             float64
 17   children_preschool                      int64
 18   preschool_quota                         float64
 19   preschool_education_centers_raion       int64
 20   children_school                         int64
 21   school_quota                            float64
 22   school_education_centers_raion          int64
 23   school_education_centers_top_20_raion   int64
 24   hospital_beds_raion                     float64
 25   healthcare_centers_raion                int64
```

```
26  university_top_20_raion                int64
27  sport_objects_raion                    int64
28  additional_education_raion             int64
29  culture_objects_top_25                 object
30  culture_objects_top_25_raion           int64
31  shopping_centers_raion                 int64
32  office_raion                           int64
33  thermal_power_plant_raion              object
34  incineration_raion                     object
35  oil_chemistry_raion                    object
36  radiation_raion                        object
37  railroad_terminal_raion                object
38  big_market_raion                       object
39  nuclear_reactor_raion                  object
40  detention_facility_raion               object
41  full_all                               int64
42  male_f                                 int64
43  female_f                               int64
44  young_all                              int64
45  young_male                             int64
46  young_female                           int64
47  work_all                               int64
48  work_male                              int64
49  work_female                            int64
50  ekder_all                              int64
51  ekder_male                             int64
52  ekder_female                           int64
53  0_6_all                                int64
54  0_6_male                               int64
55  0_6_female                             int64
56  7_14_all                               int64
57  7_14_male                              int64
58  7_14_female                            int64
59  0_17_all                               int64
60  0_17_male                              int64
61  0_17_female                            int64
62  16_29_all                              int64
63  16_29_male                             int64
64  16_29_female                           int64
65  0_13_all                               int64
66  0_13_male                              int64
67  0_13_female                            int64
68  raion_build_count_with_material_info   float64
69  build_count_block                      float64
70  build_count_wood                       float64
71  build_count_frame                      float64
72  build_count_brick                      float64
73  build_count_monolith                   float64
74  build_count_panel                      float64
75  build_count_foam                       float64
76  build_count_slag                       float64
77  build_count_mix                        float64
78  raion_build_count_with_builddate_info  float64
79  build_count_before_1920                float64
80  build_count_1921-1945                  float64
81  build_count_1946-1970                  float64
82  build_count_1971-1995                  float64
```

```
83  build_count_after_1995                     float64
84  ID_metro                                   int64
85  metro_min_avto                             float64
86  metro_km_avto                              float64
87  metro_min_walk                             float64
88  metro_km_walk                              float64
89  kindergarten_km                            float64
90  school_km                                  float64
91  park_km                                    float64
92  green_zone_km                              float64
93  industrial_km                              float64
94  water_treatment_km                         float64
95  cemetery_km                                float64
96  incineration_km                            float64
97  railroad_station_walk_km                   float64
98  railroad_station_walk_min                  float64
99  ID_railroad_station_walk                   float64
100 railroad_station_avto_km                   float64
101 railroad_station_avto_min                  float64
102 ID_railroad_station_avto                   int64
103 public_transport_station_km                float64
104 public_transport_station_min_walk          float64
105 water_km                                   float64
106 water_1line                                object
107 mkad_km                                    float64
108 ttk_km                                     float64
109 sadovoe_km                                 float64
110 bulvar_ring_km                             float64
111 kremlin_km                                 float64
112 big_road1_km                               float64
113 ID_big_road1                               int64
114 big_road1_1line                            object
115 big_road2_km                               float64
116 ID_big_road2                               int64
117 railroad_km                                float64
118 railroad_1line                             object
119 zd_vokzaly_avto_km                         float64
120 ID_railroad_terminal                       int64
121 bus_terminal_avto_km                       float64
122 ID_bus_terminal                            int64
123 oil_chemistry_km                           float64
124 nuclear_reactor_km                         float64
125 radiation_km                               float64
126 power_transmission_line_km                 float64
127 thermal_power_plant_km                     float64
128 ts_km                                      float64
129 big_market_km                              float64
130 market_shop_km                             float64
131 fitness_km                                 float64
132 swim_pool_km                               float64
133 ice_rink_km                                float64
134 stadium_km                                 float64
135 basketball_km                              float64
136 hospice_morgue_km                          float64
137 detention_facility_km                      float64
138 public_healthcare_km                       float64
139 university_km                              float64
```

```
140 workplaces_km                            float64
141 shopping_centers_km                      float64
142 office_km                                float64
143 additional_education_km                  float64
144 preschool_km                             float64
145 big_church_km                            float64
146 church_synagogue_km                      float64
147 mosque_km                                float64
148 theater_km                               float64
149 museum_km                                float64
150 exhibition_km                            float64
151 catering_km                              float64
152 ecology                                  object
153 green_part_500                           float64
154 prom_part_500                            float64
155 office_count_500                         int64
156 office_sqm_500                           int64
157 trc_count_500                            int64
158 trc_sqm_500                              int64
159 cafe_count_500                           int64
160 cafe_sum_500_min_price_avg               float64
161 cafe_sum_500_max_price_avg               float64
162 cafe_avg_price_500                       float64
163 cafe_count_500_na_price                  int64
164 cafe_count_500_price_500                 int64
165 cafe_count_500_price_1000                int64
166 cafe_count_500_price_1500                int64
167 cafe_count_500_price_2500                int64
168 cafe_count_500_price_4000                int64
169 cafe_count_500_price_high                int64
170 big_church_count_500                     int64
171 church_count_500                         int64
172 mosque_count_500                         int64
173 leisure_count_500                        int64
174 sport_count_500                          int64
175 market_count_500                         int64
176 green_part_1000                          float64
177 prom_part_1000                           float64
178 office_count_1000                        int64
179 office_sqm_1000                          int64
180 trc_count_1000                           int64
181 trc_sqm_1000                             int64
182 cafe_count_1000                          int64
183 cafe_sum_1000_min_price_avg              float64
184 cafe_sum_1000_max_price_avg              float64
185 cafe_avg_price_1000                      float64
186 cafe_count_1000_na_price                 int64
187 cafe_count_1000_price_500                int64
188 cafe_count_1000_price_1000               int64
189 cafe_count_1000_price_1500               int64
190 cafe_count_1000_price_2500               int64
191 cafe_count_1000_price_4000               int64
192 cafe_count_1000_price_high               int64
193 big_church_count_1000                    int64
194 church_count_1000                        int64
195 mosque_count_1000                        int64
196 leisure_count_1000                       int64
```

```
197 sport_count_1000                      int64
198 market_count_1000                     int64
199 green_part_1500                       float64
200 prom_part_1500                        float64
201 office_count_1500                     int64
202 office_sqm_1500                       int64
203 trc_count_1500                        int64
204 trc_sqm_1500                          int64
205 cafe_count_1500                       int64
206 cafe_sum_1500_min_price_avg           float64
207 cafe_sum_1500_max_price_avg           float64
208 cafe_avg_price_1500                   float64
209 cafe_count_1500_na_price              int64
210 cafe_count_1500_price_500             int64
211 cafe_count_1500_price_1000            int64
212 cafe_count_1500_price_1500            int64
213 cafe_count_1500_price_2500            int64
214 cafe_count_1500_price_4000            int64
215 cafe_count_1500_price_high            int64
216 big_church_count_1500                 int64
217 church_count_1500                     int64
218 mosque_count_1500                     int64
219 leisure_count_1500                    int64
220 sport_count_1500                      int64
221 market_count_1500                     int64
222 green_part_2000                       float64
223 prom_part_2000                        float64
224 office_count_2000                     int64
225 office_sqm_2000                       int64
226 trc_count_2000                        int64
227 trc_sqm_2000                          int64
228 cafe_count_2000                       int64
229 cafe_sum_2000_min_price_avg           float64
230 cafe_sum_2000_max_price_avg           float64
231 cafe_avg_price_2000                   float64
232 cafe_count_2000_na_price              int64
233 cafe_count_2000_price_500             int64
234 cafe_count_2000_price_1000            int64
235 cafe_count_2000_price_1500            int64
236 cafe_count_2000_price_2500            int64
237 cafe_count_2000_price_4000            int64
238 cafe_count_2000_price_high            int64
239 big_church_count_2000                 int64
240 church_count_2000                     int64
241 mosque_count_2000                     int64
242 leisure_count_2000                    int64
243 sport_count_2000                      int64
244 market_count_2000                     int64
245 green_part_3000                       float64
246 prom_part_3000                        float64
247 office_count_3000                     int64
248 office_sqm_3000                       int64
249 trc_count_3000                        int64
250 trc_sqm_3000                          int64
251 cafe_count_3000                       int64
252 cafe_sum_3000_min_price_avg           float64
253 cafe_sum_3000_max_price_avg           float64
```

```
 254 cafe_avg_price_3000                       float64
 255 cafe_count_3000_na_price                  int64
 256 cafe_count_3000_price_500                 int64
 257 cafe_count_3000_price_1000                int64
 258 cafe_count_3000_price_1500                int64
 259 cafe_count_3000_price_2500                int64
 260 cafe_count_3000_price_4000                int64
 261 cafe_count_3000_price_high                int64
 262 big_church_count_3000                     int64
 263 church_count_3000                         int64
 264 mosque_count_3000                         int64
 265 leisure_count_3000                        int64
 266 sport_count_3000                          int64
 267 market_count_3000                         int64
 268 green_part_5000                           float64
 269 prom_part_5000                            float64
 270 office_count_5000                         int64
 271 office_sqm_5000                           int64
 272 trc_count_5000                            int64
 273 trc_sqm_5000                              int64
 274 cafe_count_5000                           int64
 275 cafe_sum_5000_min_price_avg               float64
 276 cafe_sum_5000_max_price_avg               float64
 277 cafe_avg_price_5000                       float64
 278 cafe_count_5000_na_price                  int64
 279 cafe_count_5000_price_500                 int64
 280 cafe_count_5000_price_1000                int64
 281 cafe_count_5000_price_1500                int64
 282 cafe_count_5000_price_2500                int64
 283 cafe_count_5000_price_4000                int64
 284 cafe_count_5000_price_high                int64
 285 big_church_count_5000                     int64
 286 church_count_5000                         int64
 287 mosque_count_5000                         int64
 288 leisure_count_5000                        int64
 289 sport_count_5000                          int64
 290 market_count_5000                         int64
dtypes: float64(119), int64(157), object(15)
memory usage: 67.7+ MB
```

In [52]:
```
data_neg_one_object_dtype_cols = data_neg_one.select_dtypes(include='objec
t')
object_neg_one_cols = data_neg_one_object_dtype_cols.columns
```

In [53]:
```
object_neg_one_one_hot_df = pd.get_dummies(data=data_neg_one_object_dtype_
cols)
object_neg_one_one_hot_df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30471 entries, 0 to 30470
Data columns (total 177 columns):
 #   Column                                     Dtype
---  ------                                     -----
 0   product_type_Investment                    uint8
 1   product_type_OwnerOccupier                 uint8
 2   sub_area_Ajeroport                         uint8
```

3	sub_area_Akademicheskoe	uint8
4	sub_area_Alekseevskoe	uint8
5	sub_area_Altuf'evskoe	uint8
6	sub_area_Arbat	uint8
7	sub_area_Babushkinskoe	uint8
8	sub_area_Basmannoe	uint8
9	sub_area_Begovoe	uint8
10	sub_area_Beskudnikovskoe	uint8
11	sub_area_Bibirevo	uint8
12	sub_area_Birjulevo Vostochnoe	uint8
13	sub_area_Birjulevo Zapadnoe	uint8
14	sub_area_Bogorodskoe	uint8
15	sub_area_Brateevo	uint8
16	sub_area_Butyrskoe	uint8
17	sub_area_Caricyno	uint8
18	sub_area_Cheremushki	uint8
19	sub_area_Chertanovo Central'noe	uint8
20	sub_area_Chertanovo Juzhnoe	uint8
21	sub_area_Chertanovo Severnoe	uint8
22	sub_area_Danilovskoe	uint8
23	sub_area_Dmitrovskoe	uint8
24	sub_area_Donskoe	uint8
25	sub_area_Dorogomilovo	uint8
26	sub_area_Filevskij Park	uint8
27	sub_area_Fili Davydkovo	uint8
28	sub_area_Gagarinskoe	uint8
29	sub_area_Gol'janovo	uint8
30	sub_area_Golovinskoe	uint8
31	sub_area_Hamovniki	uint8
32	sub_area_Horoshevo-Mnevniki	uint8
33	sub_area_Horoshevskoe	uint8
34	sub_area_Hovrino	uint8
35	sub_area_Ivanovskoe	uint8
36	sub_area_Izmajlovo	uint8
37	sub_area_Jakimanka	uint8
38	sub_area_Jaroslavskoe	uint8
39	sub_area_Jasenevo	uint8
40	sub_area_Juzhnoe Butovo	uint8
41	sub_area_Juzhnoe Medvedkovo	uint8
42	sub_area_Juzhnoe Tushino	uint8
43	sub_area_Juzhnoportovoe	uint8
44	sub_area_Kapotnja	uint8
45	sub_area_Kon'kovo	uint8
46	sub_area_Koptevo	uint8
47	sub_area_Kosino-Uhtomskoe	uint8
48	sub_area_Kotlovka	uint8
49	sub_area_Krasnosel'skoe	uint8
50	sub_area_Krjukovo	uint8
51	sub_area_Krylatskoe	uint8
52	sub_area_Kuncevo	uint8
53	sub_area_Kurkino	uint8
54	sub_area_Kuz'minki	uint8
55	sub_area_Lefortovo	uint8
56	sub_area_Levoberezhnoe	uint8
57	sub_area_Lianozovo	uint8
58	sub_area_Ljublino	uint8
59	sub_area_Lomonosovskoe	uint8

60	sub_area_Losinoostrovskoe	uint8
61	sub_area_Mar'ina Roshha	uint8
62	sub_area_Mar'ino	uint8
63	sub_area_Marfino	uint8
64	sub_area_Matushkino	uint8
65	sub_area_Meshhanskoe	uint8
66	sub_area_Metrogorodok	uint8
67	sub_area_Mitino	uint8
68	sub_area_Molzhaninovskoe	uint8
69	sub_area_Moskvorech'e-Saburovo	uint8
70	sub_area_Mozhajskoe	uint8
71	sub_area_Nagatino-Sadovniki	uint8
72	sub_area_Nagatinskij Zaton	uint8
73	sub_area_Nagornoe	uint8
74	sub_area_Nekrasovka	uint8
75	sub_area_Nizhegorodskoe	uint8
76	sub_area_Novo-Peredelkino	uint8
77	sub_area_Novogireevo	uint8
78	sub_area_Novokosino	uint8
79	sub_area_Obruchevskoe	uint8
80	sub_area_Ochakovo-Matveevskoe	uint8
81	sub_area_Orehovo-Borisovo Juzhnoe	uint8
82	sub_area_Orehovo-Borisovo Severnoe	uint8
83	sub_area_Ostankinskoe	uint8
84	sub_area_Otradnoe	uint8
85	sub_area_Pechatniki	uint8
86	sub_area_Perovo	uint8
87	sub_area_Pokrovskoe Streshnevo	uint8
88	sub_area_Poselenie Desjonovskoe	uint8
89	sub_area_Poselenie Filimonkovskoe	uint8
90	sub_area_Poselenie Kievskij	uint8
91	sub_area_Poselenie Klenovskoe	uint8
92	sub_area_Poselenie Kokoshkino	uint8
93	sub_area_Poselenie Krasnopahorskoe	uint8
94	sub_area_Poselenie Marushkinskoe	uint8
95	sub_area_Poselenie Mihajlovo-Jarcevskoe	uint8
96	sub_area_Poselenie Moskovskij	uint8
97	sub_area_Poselenie Mosrentgen	uint8
98	sub_area_Poselenie Novofedorovskoe	uint8
99	sub_area_Poselenie Pervomajskoe	uint8
100	sub_area_Poselenie Rjazanovskoe	uint8
101	sub_area_Poselenie Rogovskoe	uint8
102	sub_area_Poselenie Shhapovskoe	uint8
103	sub_area_Poselenie Shherbinka	uint8
104	sub_area_Poselenie Sosenskoe	uint8
105	sub_area_Poselenie Vnukovskoe	uint8
106	sub_area_Poselenie Voronovskoe	uint8
107	sub_area_Poselenie Voskresenskoe	uint8
108	sub_area_Preobrazhenskoe	uint8
109	sub_area_Presnenskoe	uint8
110	sub_area_Prospekt Vernadskogo	uint8
111	sub_area_Ramenki	uint8
112	sub_area_Rjazanskij	uint8
113	sub_area_Rostokino	uint8
114	sub_area_Savelki	uint8
115	sub_area_Savelovskoe	uint8
116	sub_area_Severnoe	uint8

```
117 sub_area_Severnoe Butovo                         uint8
118 sub_area_Severnoe Izmajlovo                      uint8
119 sub_area_Severnoe Medvedkovo                     uint8
120 sub_area_Severnoe Tushino                        uint8
121 sub_area_Shhukino                                uint8
122 sub_area_Silino                                  uint8
123 sub_area_Sokol                                   uint8
124 sub_area_Sokol'niki                              uint8
125 sub_area_Sokolinaja Gora                         uint8
126 sub_area_Solncevo                                uint8
127 sub_area_Staroe Krjukovo                         uint8
128 sub_area_Strogino                                uint8
129 sub_area_Sviblovo                                uint8
130 sub_area_Taganskoe                               uint8
131 sub_area_Tekstil'shhiki                          uint8
132 sub_area_Teplyj Stan                             uint8
133 sub_area_Timirjazevskoe                          uint8
134 sub_area_Troickij okrug                          uint8
135 sub_area_Troparevo-Nikulino                      uint8
136 sub_area_Tverskoe                                uint8
137 sub_area_Veshnjaki                               uint8
138 sub_area_Vnukovo                                 uint8
139 sub_area_Vojkovskoe                              uint8
140 sub_area_Vostochnoe                              uint8
141 sub_area_Vostochnoe Degunino                     uint8
142 sub_area_Vostochnoe Izmajlovo                    uint8
143 sub_area_Vyhino-Zhulebino                        uint8
144 sub_area_Zamoskvorech'e                          uint8
145 sub_area_Zapadnoe Degunino                       uint8
146 sub_area_Zjablikovo                              uint8
147 sub_area_Zjuzino                                 uint8
148 culture_objects_top_25_no                        uint8
149 culture_objects_top_25_yes                       uint8
150 thermal_power_plant_raion_no                     uint8
151 thermal_power_plant_raion_yes                    uint8
152 incineration_raion_no                            uint8
153 incineration_raion_yes                           uint8
154 oil_chemistry_raion_no                           uint8
155 oil_chemistry_raion_yes                          uint8
156 radiation_raion_no                               uint8
157 radiation_raion_yes                              uint8
158 railroad_terminal_raion_no                       uint8
159 railroad_terminal_raion_yes                      uint8
160 big_market_raion_no                              uint8
161 big_market_raion_yes                             uint8
162 nuclear_reactor_raion_no                         uint8
163 nuclear_reactor_raion_yes                        uint8
164 detention_facility_raion_no                      uint8
165 detention_facility_raion_yes                     uint8
166 water_1line_no                                   uint8
167 water_1line_yes                                  uint8
168 big_road1_1line_no                               uint8
169 big_road1_1line_yes                              uint8
170 railroad_1line_no                                uint8
171 railroad_1line_yes                               uint8
172 ecology_excellent                                uint8
173 ecology_good                                     uint8
```

```
 174 ecology_no data                                    uint8
 175 ecology_poor                                       uint8
 176 ecology_satisfactory                               uint8
dtypes: uint8(177)
memory usage: 5.1 MB
```

In [54]:
```
data_neg_one.drop(object_neg_one_cols, inplace=True, axis=1)
frames = [data_neg_one, object_neg_one_one_hot_df]
data_neg_one = pd.concat(frames, axis=1)
```

In [ ]:
```
#######.    Pickle save data set -1 for later use
# pickle.dump(data_neg_one, open( "data_neg_one.pkl", "wb" ))
```

In [ ]:
```
# create new model using XGB randomsearch function
xgb_neg_one = xgb_randomsearch(xgb_df=data_neg_one, xgb_y=target, n_iter=2
0, param_grid=param_grid)
```

In [ ]:
```
# Pickle the model for negative 1
# pickle.dump(xgb_neg_one, open( "xgb_neg_one.pkl", "wb" ))
```

In [55]:
```
##### Load pickled model and -1 dataset if not in memory ####
data_neg_one = pickle.load(open("data_neg_one.pkl","rb"))
xgb_neg_one = pickle.load(open("xgb_neg_one.pkl", "rb"))
```

```
/Users/melschwan/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py
:315: UserWarning: Trying to unpickle estimator RandomizedSearchCV from ve
rsion 0.22.1 when using version 0.24.1. This might lead to breaking code o
r invalid results. Use at your own risk.
  UserWarning)
```

In [56]:
```
# predict value against full dataset using CV tuned model
xgb_neg_one_pred = xgb_neg_one.predict(data_neg_one)
```

In [57]:
```
# calculate RMSE
neg_one_xgb_rmse = np.sqrt(MSE(target, xgb_neg_one_pred))
# output RMSE
print(neg_one_xgb_rmse)
```

```
1739365.3840950478
```

In [ ]:
```
##########################################################
####     Model XGBoost Null Values set median         ####
##########################################################
```

In [ ]:
```
imputation_dict = {'metro_min_walk': 20.324236305,
 'metro_km_walk': 1.6936863585,
 'railroad_station_walk_km': 3.242133743,
 'railroad_station_walk_min': 38.905604915,
 'ID_railroad_station_walk': 24.0,
 'prom_part_5000': 8.97,
 'raion_build_count_with_material_info': 273.0,
 'build_count_block': 42.0,
 'build_count_wood': 0.0,
 'build_count_frame': 0.0,
```

```
 'build_count_brick': 67.0,
 'build_count_monolith': 6.0,
 'build_count_panel': 92.0,
 'build_count_foam': 0.0,
 'build_count_slag': 0.0,
 'build_count_mix': 0.0,
 'raion_build_count_with_builddate_info': 271.0,
 'build_count_before_1920': 0.0,
 'build_count_1921-1945': 2.0,
 'build_count_1946-1970': 135.0,
 'build_count_1971-1995': 71.0,
 'build_count_after_1995': 24.0,
 'hospital_beds_raion': 990.0,
 'cafe_sum_500_min_price_avg': 666.67,
 'cafe_avg_price_500': 916.67,
 'cafe_sum_1000_min_price_avg': 669.23,
 'cafe_avg_price_1000': 912.5,
 'cafe_sum_1500_min_price_avg': 692.31,
 'cafe_avg_price_1500': 926.32,
 'cafe_sum_2000_min_price_avg': 683.33,
 'cafe_avg_price_2000': 919.23,
 'cafe_sum_3000_min_price_avg': 711.11,
 'cafe_avg_price_3000': 961.11,
 'cafe_sum_5000_min_price_avg': 721.74,
 'cafe_avg_price_5000': 966.67,
 'life_sq': 30.0,
 'floor': 6.0,
 'max_floor': 12.0,
 'build_year': 1979.0,
 'num_room': 2.0,
 'kitch_sq': 6.4,
 'state': 2.0,
 'preschool_quota': 2854.0,
 'school_quota': 7377.0,
 'material': 1.0,
 'cafe_sum_500_max_price_avg': 1166.67,
 'cafe_sum_1000_max_price_avg': 1142.86,
 'cafe_sum_1500_max_price_avg': 1166.67,
 'cafe_sum_2000_max_price_avg': 1156.25,
 'cafe_sum_3000_max_price_avg': 1211.54,
 'cafe_sum_5000_max_price_avg': 1211.9450000000002
}
```

In [ ]:
```
data_impute.fillna(imputation_dict,inplace=True)
data_impute_null_count = data_impute.isnull().sum()
```

In [ ]:
```
#converting state and material to object as they are categorical items and
 will be one-hot encoded
data_impute.state = data_impute.state.astype(object)
data_impute.material = data_impute.material.astype(object)
```

In [ ]:
```
data_impute_object_dtype_cols = data_impute.select_dtypes(include='object'
)
object_impute_cols = data_neg_one_object_dtype_cols.columns
object_impute_one_hot_df = pd.get_dummies(data=data_impute_object_dtype_co
ls)
```

```
object_impute_one_hot_df.info(verbose=True)
data_impute.drop(object_impute_cols, inplace=True, axis=1)
frames_impute = [data_impute, object_impute_one_hot_df]
data_impute = pd.concat(frames_impute, axis=1)
```

In [ ]:
```
xgb_impute = xgb_randomsearch(xgb_df=data_impute, xgb_y=target, n_iter=20,
 param_grid=param_grid)
xgb_impute_pred = xgb_impute.predict(data_impute)
xgb_impute_rmse = np.sqrt(MSE(target, xgb_impute_pred))
```

In [ ]:
```
print(xgb_impute_rmse)
```

In [ ]:
```
1755174.799799309
```