Libraries

In [ ]:
```
import warnings as wrn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as GS
from matplotlib.colors import LinearSegmentedColormap as LSC
import seaborn as sns
import random as rnd
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor as RFR, GradientBoostingRegressor as GBR
from sklearn.model_selection import GridSearchCV as GSCV, cross_val_score as CVS

# color mapping for further use
cmap_colors = ["#EDEDED", "#E4E4E4", "#DCDCDC", # grays
               "#72C96F", # green
               "#85C169", "#8FBD67", "#98BA64", "#A3B664", "#ABB25E", "#B5AE5B", "#BDAA58", "#C7A756", "#D0A353",# gradual transitions to orange
               "#E39B4D", "#E39B4D", "#E39B4D", "#E39B4D", "#E39B4D", # orange
               "#D68844", "#CA753A", "#BD6131", "#B04E27", "#A33B1E", "#972814", "#8A140B", # gradual transitions to red
               "#7D0101", "#7D0101", "#7D0101"] # red
custom_cmap = LSC.from_list('custom', cmap_colors)
```

1. Baseline Model Development

a. Baseline Model (from Task 2)

Getting data from train.csv & test.csv

In [ ]:
```
# for organization purposes and potential reproducibility
trainCols = ['full_sq', 'life_sq', 'floor', 'max_floor', 'material', 'build_year', 'num_room',
             'kitch_sq', 'state', 'product_type', 'sub_area', 'price_doc', 'timestamp']
testCols = ['id', 'full_sq', 'life_sq', 'floor', 'max_floor', 'material', 'build_year',
            'num_room', 'kitch_sq', 'state', 'product_type', 'sub_area', 'timestamp']
# getting all relevant features of train & test datasets
SbrBkTrain = pd.read_csv('train.csv', parse_dates=['timestamp']).filter(trainCols)
SbrBkTest = pd.read_csv('test.csv', parse_dates=['timestamp']).filter(testCols)
```

Original data cleaning

In [ ]:
```
wrn.filterwarnings('ignore')

SbrBkTrain.full_sq.loc[SbrBkTrain.full_sq < 7] = np.nan
SbrBkTrain.full_sq.loc[SbrBkTrain.full_sq > 500] = np.nan
SbrBkTrain.life_sq.loc[SbrBkTrain.life_sq < 7] = np.nan
SbrBkTrain.life_sq.loc[SbrBkTrain.life_sq > 300] = np.nan
SbrBkTrain.floor.loc[SbrBkTrain.floor > 40] = np.nan
SbrBkTrain.max_floor.loc[SbrBkTrain.max_floor > 50] = np.nan
SbrBkTrain.max_floor.loc[SbrBkTrain.max_floor < SbrBkTrain.floor] = np.nan
SbrBkTrain.build_year.loc[SbrBkTrain.build_year < 1800] = np.nan
SbrBkTrain.build_year.loc[SbrBkTrain.build_year > 2020] = np.nan
SbrBkTrain.num_room.loc[SbrBkTrain.num_room < 1] = np.nan
SbrBkTrain.num_room.loc[SbrBkTrain.num_room > 12] = np.nan
SbrBkTrain.kitch_sq.loc[SbrBkTrain.kitch_sq < 2] = np.nan
SbrBkTrain.kitch_sq.loc[SbrBkTrain.kitch_sq > 250] = np.nan
SbrBkTrain.life_sq.loc[SbrBkTrain.life_sq > SbrBkTrain.full_sq] = np.nan
SbrBkTrain.kitch_sq.loc[SbrBkTrain.full_sq < SbrBkTrain.kitch_sq] = np.nan
SbrBkTrain.kitch_sq.loc[(SbrBkTrain.kitch_sq / SbrBkTrain.full_sq > SbrBkTrain.life_sq / SbrBkTrain.full_sq) & (SbrBkTrain.full_sq >= SbrBkTrain.life_sq + SbrBkTrain.kitch
SbrBkTrain.life_sq.loc[SbrBkTrain.full_sq < SbrBkTrain.life_sq + SbrBkTrain.kitch_sq] = SbrBkTrain.full_sq.loc[SbrBkTrain.full_sq < SbrBkTrain.life_sq + SbrBkTrain.kitch_
SbrBkTrain.state.loc[SbrBkTrain.state > 4] = SbrBkTrain.state.loc[SbrBkTrain.state > 4] // 10

SbrBkTest.full_sq.loc[SbrBkTest.full_sq < 7] = np.nan
SbrBkTest.full_sq.loc[SbrBkTest.full_sq > 500] = np.nan
SbrBkTest.life_sq.loc[SbrBkTest.life_sq < 7] = np.nan
SbrBkTest.life_sq.loc[SbrBkTest.life_sq > 300] = np.nan
SbrBkTest.floor.loc[SbrBkTest.floor > 40] = np.nan
SbrBkTest.max_floor.loc[SbrBkTest.max_floor > 50] = np.nan
SbrBkTest.max_floor.loc[SbrBkTest.max_floor < SbrBkTest.floor] = np.nan
SbrBkTest.build_year.loc[SbrBkTest.build_year < 1800] = np.nan
SbrBkTest.build_year.loc[SbrBkTest.build_year > 2020] = np.nan
SbrBkTest.num_room.loc[SbrBkTest.num_room < 1] = np.nan
SbrBkTest.num_room.loc[SbrBkTest.num_room > 12] = np.nan
SbrBkTest.life_sq.loc[SbrBkTest.life_sq > SbrBkTest.full_sq] = np.nan
SbrBkTest.kitch_sq.loc[SbrBkTest.full_sq < SbrBkTest.kitch_sq] = np.nan
SbrBkTest.kitch_sq.loc[(SbrBkTest.kitch_sq / SbrBkTest.full_sq > SbrBkTest.life_sq / SbrBkTest.full_sq) & (SbrBkTest.full_sq >= SbrBkTest.life_sq + SbrBkTest.kitch_sq)] =
SbrBkTest.life_sq.loc[SbrBkTest.full_sq < SbrBkTest.life_sq + SbrBkTest.kitch_sq] = SbrBkTest.full_sq.loc[SbrBkTest.full_sq < SbrBkTest.life_sq + SbrBkTest.kitch_sq] - Sbr
SbrBkTest.state.loc[SbrBkTest.state > 4] = SbrBkTest.state.loc[SbrBkTest.state > 4] // 10
```

Original Handling of missing values

Viewing precentages of missing values

In [ ]:
```
print(f'Precentages of missing values in Train:\n{(SbrBkTrain.isna().sum() / SbrBkTrain.shape[0] * 100).round(2).sort_values(ascending = False)}')
```

```
Precentages of missing values in Train:
kitch_sq        51.97
build_year      47.62
state           44.50
max_floor       36.33
num_room        31.47
material        31.41
life_sq         22.49
floor            0.56
full_sq          0.11
product_type     0.00
sub_area         0.00
price_doc        0.00
timestamp        0.00
dtype: float64
```

Removing unacceptable columns and rows

In [ ]:
```
wrn.filterwarnings('ignore')

SbrBkTrain.drop(columns = ['kitch_sq'], inplace = True)
SbrBkTest.drop(columns = ['kitch_sq'], inplace = True)

SbrBkTrain['toDrop'] = (((SbrBkTrain.build_year.isna()) | (SbrBkTrain.state.isna())) & (SbrBkTrain.isna().sum(axis = 1) > 5))
# marking rows that are randomly selected to be dropped
SbrBkTrain.toDrop[rnd.sample(list((((SbrBkTrain.build_year.isna()) | (SbrBkTrain.state.isna())) & (SbrBkTrain.isna().sum(axis = 1) == 5))[(((SbrBkTrain.build_year.isna())
# dropping all marked rows
SbrBkTrain = SbrBkTrain[~SbrBkTrain.toDrop]
SbrBkTrain.drop(columns = ['toDrop'], inplace = True)

# viewing the precentages again
print(f'Precentages of missing values in Train:\n{(SbrBkTrain.isna().sum() / SbrBkTrain.shape[0] * 100).round(4).sort_values(ascending = False)}\n')
print(f'Precentages of missing values in Test:\n{(SbrBkTest.isna().sum() / SbrBkTest.shape[0] * 100).round(4).sort_values(ascending = False)}')
```

```
Precentages of missing values in Train:
build_year      35.5516
state           31.7073
max_floor       21.6524
life_sq         18.5309
num_room        15.6719
material        15.6073
full_sq          0.1171
floor            0.0121
product_type     0.0000
sub_area         0.0000
price_doc        0.0000
timestamp        0.0000
dtype: float64

Precentages of missing values in Test:
build_year      20.9736
life_sq         19.7860
state            9.0577
max_floor        8.3921
product_type     0.4307
full_sq          0.0392
floor            0.0131
num_room         0.0131
id               0.0000
material         0.0000
sub_area         0.0000
timestamp        0.0000
dtype: float64
```

Imputing missing values

In [ ]:
```
wrn.filterwarnings('ignore')

SbrBkTrain['product_typeCode'] = pd.Categorical(SbrBkTrain.product_type).codes
SbrBkTest['product_typeCode'] = pd.Categorical(SbrBkTest.product_type).codes
SbrBkTest.product_typeCode.loc[SbrBkTest.product_typeCode == -1] = np.nan
SbrBkTrain['sub_areaCode'] = pd.Categorical(SbrBkTrain.sub_area).codes
SbrBkTest['sub_areaCode'] = pd.Categorical(SbrBkTest.sub_area).codes

imputedTrainData = KNNImputer(n_neighbors = 20).fit_transform(SbrBkTrain[['full_sq', 'life_sq', 'floor', 'max_floor', 'material', 'build_year', 'num_room', 'state', 'produ
imputedTestData = KNNImputer(n_neighbors = 20).fit_transform(SbrBkTest[['full_sq', 'life_sq', 'floor', 'max_floor', 'material', 'build_year', 'num_room', 'state', 'product
imputedTrain = pd.DataFrame(imputedTrainData, columns = ['full_sq', 'life_sq', 'floor', 'max_floor', 'material', 'build_year', 'num_room', 'state', 'product_typeCode', 'su
imputedTest = pd.DataFrame(imputedTestData, columns = ['full_sq', 'life_sq', 'floor', 'max_floor', 'material', 'build_year', 'num_room', 'state', 'product_typeCode', 'sub_
imputedTrain.drop(columns = ['product_typeCode', 'sub_areaCode'], inplace = True)
imputedTrain = imputedTrain.round()
imputedTest = imputedTest.round()
imputedTrain = imputedTrain.set_index(SbrBkTrain.index)
imputedTest = imputedTest.set_index(SbrBkTest.index)
imputedTrain['product_type'] = SbrBkTrain.product_type
imputedTest['product_type'] = SbrBkTest.product_type
imputedTrain['sub_area'] = SbrBkTrain.sub_area
imputedTest['sub_area'] = SbrBkTest.sub_area
imputedTrain['price_doc'] = SbrBkTrain.price_doc
imputedTest['id'] = SbrBkTest.id
imputedTrain['timestamp'] = SbrBkTrain.timestamp
imputedTest['timestamp'] = SbrBkTest.timestamp
imputedTest.product_type.loc[imputedTest.product_type.isna()] = np.where(imputedTest.product_typeCode.loc[imputedTest.product_type.isna()] == 0, 'Investment', 'OwnerOccupi
imputedTest.drop(columns = ['product_typeCode', 'sub_areaCode'], inplace = True)
imputedTrain['logPrice'] = np.log(imputedTrain.price_doc)
```

In [ ]:
```
print(f'Precentages of missing values in Train:\n{(imputedTrain.isna().sum() / imputedTrain.shape[0] * 100).round(4).sort_values(ascending = False)}')
print(f'Precentages of missing values in Test:\n{(imputedTest.isna().sum() / imputedTest.shape[0] * 100).round(4).sort_values(ascending = False)}')
```

```
Precentages of missing values in Train:
full_sq         0.0
life_sq         0.0
floor           0.0
max_floor       0.0
material        0.0
build_year      0.0
num_room        0.0
state           0.0
product_type    0.0
sub_area        0.0
price_doc       0.0
timestamp       0.0
logPrice        0.0
dtype: float64
Precentages of missing values in Test:
full_sq         0.0
life_sq         0.0
floor           0.0
max_floor       0.0
material        0.0
build_year      0.0
num_room        0.0
state           0.0
product_type    0.0
sub_area        0.0
id              0.0
timestamp       0.0
dtype: float64
```

Original feature engineering

In [ ]:
```
wrn.filterwarnings('ignore')

imputedTrain['buildingAge'] = imputedTrain.timestamp.dt.year - imputedTrain.build_year
imputedTrain = imputedTrain.assign(roomPerSq = imputedTrain.num_room / imputedTrain.full_sq,
                                   practAging = imputedTrain.buildingAge / imputedTrain.state,
                                   roomSize = imputedTrain.life_sq / imputedTrain.num_room,
                                   heightScore = (imputedTrain.floor - imputedTrain.max_floor * 0.37).abs() / (imputedTrain.max_floor * 0.63 / 10),
                                   lifeRatio = imputedTrain.life_sq / (imputedTrain.full_sq / 10),
                                   month = imputedTrain.timestamp.dt.month)
imputedTest['buildingAge'] = imputedTest.timestamp.dt.year - imputedTest.build_year
imputedTest = imputedTest.assign(roomPerSq = imputedTest.num_room / imputedTest.full_sq,
                                 practAging = imputedTest.buildingAge / imputedTest.state,
                                 roomSize = imputedTest.life_sq / imputedTest.num_room,
                                 heightScore = (imputedTest.floor - imputedTest.max_floor * 0.37).abs() / (imputedTest.max_floor * 0.63 / 10),
                                 lifeRatio = imputedTest.life_sq / (imputedTest.full_sq / 10),
                                 month = imputedTest.timestamp.dt.month)
imputedTrain.heightScore.loc[imputedTrain.heightScore.isna()] = 7
imputedTest.heightScore.loc[imputedTest.heightScore.isna()] = 7
```

Original Label encoding

In [ ]:
```
imputedUnion = pd.concat([imputedTrain, imputedTest], ignore_index=True)
prodTypeLbl = LabelEncoder().fit(imputedUnion.product_type)
subAreaLbl = LabelEncoder().fit(imputedUnion.sub_area)
mtrlLbl = LabelEncoder().fit(imputedUnion.material)
stateLbl = LabelEncoder().fit(imputedUnion.state)
imputedTrain.product_type = prodTypeLbl.transform(imputedTrain.product_type)
imputedTest.product_type = prodTypeLbl.transform(imputedTest.product_type)
imputedTrain.sub_area = subAreaLbl.transform(imputedTrain.sub_area)
imputedTest.sub_area = subAreaLbl.transform(imputedTest.sub_area)
imputedTrain.material = mtrlLbl.transform(imputedTrain.material)
imputedTest.material = mtrlLbl.transform(imputedTest.material)
imputedTrain.state = stateLbl.transform(imputedTrain.state)
imputedTest.state = stateLbl.transform(imputedTest.state)
```

Original Rescaling

In [ ]:
```
# Rescaling full_sq in imputedTrain and impuedTest
scaledMaxFullSq = imputedUnion.full_sq.max() / 100
imputedTrain.full_sq = imputedTrain.full_sq / scaledMaxFullSq
imputedTest.full_sq = imputedTest.full_sq / scaledMaxFullSq

# Rescaling life_sq in imputedTrain and impuedTest
scaledMaxLifeSq = imputedUnion.life_sq.max() / 100
imputedTrain.life_sq = imputedTrain.life_sq / scaledMaxLifeSq
imputedTest.life_sq = imputedTest.life_sq / scaledMaxLifeSq

# Rescaling build_year in imputedTrain and impuedTest
minBuildYear = imputedUnion.build_year.min()
imputedTrain.build_year = imputedTrain.build_year - minBuildYear
imputedTest.build_year = imputedTest.build_year - minBuildYear
```

Original Train/Validation split (without test, because the test dataset represents that)

In [ ]:
```
finalTrain = imputedTrain.copy(deep = True).reset_index(drop = True)
finalTest = imputedTest.copy(deep = True).reset_index(drop = True)

# due to it being a time series it is only appropriate to split the data in a chronological manner
TrainData = finalTrain.loc[:int(finalTrain.shape[0] * 70 / 85), :]
xTrn = TrainData.drop(columns = ['price_doc', 'logPrice', 'timestamp'])
yTrn = TrainData.logPrice
ValidationData = finalTrain.loc[int(finalTrain.shape[0] * 70 / 85):, :]
xVal = ValidationData.drop(columns = ['price_doc', 'logPrice', 'timestamp'])
yVal = ValidationData.logPrice
```

Baseline Model Model

In [ ]:
```
ftrSlctModel = RFR(n_jobs = -1, random_state = 42)
ftrSlctModel.fit(xTrn, yTrn)
imprtnc = ftrSlctModel.feature_importances_
features = pd.Series(imprtnc, index = xTrn.columns).sort_values(ascending = False)
features = list(features[features > 0.05].index)

rndFrstBaseModel = RFR()
rndFrstParams = {
    'max_depth': [i for i in range(4, 9, 2)],
    'n_estimators': [i for i in range(100, 301, 50)],
    'min_samples_leaf': [i for i in range(3, 10, 3)]
}

grdSrcRndFrst = GSCV(rndFrstBaseModel, rndFrstParams, cv = 5, scoring = 'neg_mean_squared_error', verbose = 2)
grdSrcRndFrst.fit(xTrn[features], yTrn)

RndFrstModelParams = grdSrcRndFrst.best_params_

rndFrstOptModel = RFR(**RndFrstModelParams)
rndFrstOptModel.fit(xTrn[features], yTrn)
```

backup for creating the models quickly

In [ ]:
```
ftrSlctModel = RFR(n_jobs = -1, random_state = 42)
ftrSlctModel.fit(xTrn, yTrn)
imprtnc = ftrSlctModel.feature_importances_
features = pd.Series(imprtnc, index = xTrn.columns).sort_values(ascending = False)
features = list(features[features > 0.05].index)
RndFrstModelParams = {
    'max_depth': 8,
    'min_samples_leaf': 9,
    'n_estimators': 200
}
```

```
rndFrstOptModel = RFR(**RndFrstModelParams)
rndFrstOptModel.fit(xTrn[features], yTrn)
```

Out[ ]:

RandomForestRegressor

```
RandomForestRegressor(max_depth=8, min_samples_leaf=9, n_estimators=200)
```

b. Baseline Results

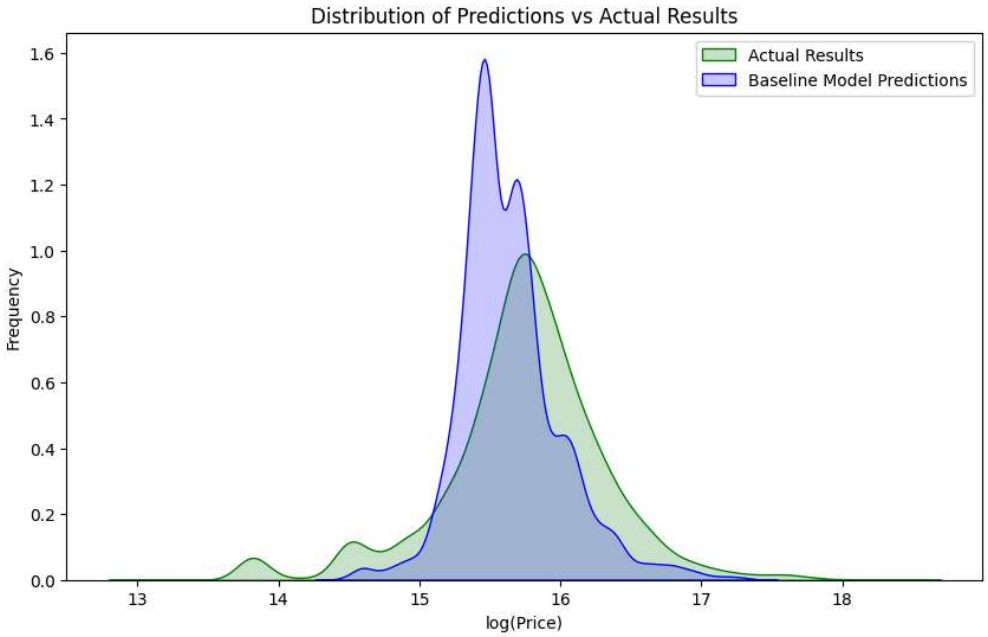
need to discuss the model's performance metrics

therefor need to calculate the model's performance metrics

In [ ]:

```
rndFrstMseScores = -1 * CVS(rndFrstOptModel, xVal[features], yVal, cv = 5, scoring = 'neg_mean_squared_error')
rndFrstPred = rndFrstOptModel.predict(xVal[features])
rndFrstRmsle = np.sqrt(np.mean((rndFrstPred - yVal) ** 2))
rndFrstR2 = rndFrstOptModel.score(xVal[features], yVal)

metrics = pd.DataFrame({'Model': ['Random Forest Model', 'temp'],
                        'MSE': [rndFrstMseScores.mean(), 1],
                        'RMSLE': [rndFrstRmsle, 1],
                        'R^2': [rndFrstR2, 1]})

metrics.drop(index = 1, inplace = True)

display(metrics)

plt.figure(figsize=(10, 6))
sns.kdeplot(yVal, color = 'green', fill = True, alpha = 0.2, label='Actual Results')
sns.kdeplot(rndFrstPred, color = 'blue', fill = True, alpha = 0.2, label='Baseline Model Predictions')
plt.xlabel('log(Price)')
plt.ylabel('Frequency')
plt.title('Distribution of Predictions vs Actual Results')
plt.legend()
plt.show()
```

	Model	MSE	RMSLE	R^2
0	Random Forest Model	0.192769	0.44912	0.409938



We can see here that there isnt to much of an overlap of the baseline model's predictions distribution with the actual distribution

Kaggle Submission

In [ ]:

```
# export baseline model
yPredRndFrst = np.round(np.exp(rndFrstOptModel.predict(finalTest[features])), decimals = -3)
rndFrstSubmission = pd.DataFrame({'id': finalTest.id, 'price_doc': yPredRndFrst})
rndFrstSubmission.to_csv('RandomForestSubmission.csv', index = False)
```

when submitting the predictions test data we get the the following results:

	Private Score	Public Score
Random Forest Model	0.38628	0.38735

2. Notebook Analysis

a. Notebook Selection

we chose the notebooks called:

- Simple Exploration Notebook - Sberbank
- Feature Engineering & Validation Strategy | Kaggle

delete later

- 'basic-time-series-analysis-feature-selection.ipynb' (correlation matrix)

b. Analysis Summary

We noticed several things that are worth mentioning

Simple Exploration Notebook - Sberbank

- **Price Transformation Logarithmic Form**

  SRK (the author of these notebooks) made an observation on the distribution of price_doc and noticed that it is very right-skewed, so he decided to transform it, and later on model on the log-Price instead on the original price.

- **Median Price Change Across Time**

  SRK attempted to identify price changes over the different year & month combinations that are present in the data using a barplot of the median price in each year & month combination.

  In this graph there seem to be some relation, but further on in the second notebook it is clearer that this is not the case.

- **Missing Values**

  SRK uses a horizontal barplot to view the amounts of missing values in each column, and he finds that there is there some grouping effect over the missing values of diffeent columns, different columns have the same amount of missing values (seemingly atleast).

- **Top 5 Highest Importance Features (HI Features)**

  SRK finds the top 5 highest importance features, which are:

  1. $full\_sq \quad importance = 1677$
  2. $life\_sq \quad importance = 708$
  3. $floor \quad importance = 670$
  4. $max\_floor \quad importance = 460$
  5. $build\_year \quad importance = 431$

- **Visual Relation of Top 5 HI Features With Price**

  To better understand the relationships of these top 5 HI features, SRK creates several kinds of graphs to see and describe the interpertation of log-Price across the changing values of these features.

  1. full_sq:
  - ***[1D]Distribution-Scatter Joinplot*** of the log-Full_sq and log_Price, and there seem to be some linear relation expressed between them.
  2. life_sq:
  - ***[1D-2D]Distribution Joinplot*** of the log-Life_sq and log_Price, and it very noticeable that there are many values in log-Life_sq that equal to 0, and are practically separate from many other values, meaning many of life_sq values equal to 1 or close to 1, are there might be some error in these samples.
  3. floor:
  - ***[1D]Countplot*** of floor, it seem to be right-skewed in jumps.
  - ***Pointplot*** of floor and median-Price, the higher the floor the higher the median-Price (generally) rises where it peeks at 33.
  4. max_floor:
  - ***[1D]Countplot*** of max_floor, it is raised for consideration that some normalization might be needed for the floor and max_floor features.
  - ***Boxplot*** of max_floor and median-Price, seem fairly expected after knowing all of the previous information.

Feature Engineering & Validation Strategy | Kaggle

- **Importing the Data with Additional Macro-Economic Features**

  Not like in his previous notebook, in this notebook SRK imports the data and adds to the train dataset and the test dataset additional macro-economic features that are in the macro.csv file, this would expand his options in picking the features to model apon.

- **Categorical Features Conversion - Label Encoding**

  SRK converts the categorical features to numerical using label encoding to be able further on to test their importance and affect on the price.

- **Missing Values Features**

  SRK knows from his previous notebook that some behaviour is displayed across the missing values, so here he makes a new feature that represents the amount of missing values in each row, where potentially this could have an affect on the model.

  right after that he fills all the missing values with -99, this value is highly unlikely, and due to the use of XGBoost and Random Forest models these values are very descernable by the trees, and further more, could be handled appropriately according to them being a missing value.

- **Timestamp Derived Features**

  The timestamp feature is of either character, datetime or (after some tranformation) int dtype, all 3 of these dont properly represent the full interpertation of the feature that could be important to the model, such as the year, year & month combination, month of the year, year & week combination, week of the year, day of the week and potentially many others,

  SRK derives several features out of the timestamp feature and plots each of their relation with price, here it is clearer that they dont have any clear relation that is usefull to the model.

- **Top 5 HI Features Derived Features**

  According to the previous notebook top 5 HI features, SRK creates and tests some features that could be derived from these features.

  - from full_sq and life_sq he tries to create some ratio features and difference features, all of which dont seem to be of any help.
  - from floor and max_floor he tries as well to create some ration features and difference features, these he doesnt seem to particulary test in any visual way like previously.
  - from build_year he creates a building age feature, also not tested.
  - out of the timestamp features he created previously he creates supply & demand features that represent aproximatly how many apartments were available and were sold in a specific year & month/week combination
  - further on he starts creating features that are generaly logical to take into acount when pricing an apartment, such as the ratio of children in school or preschool over the school and preschool quotas.

- **Model Generation & Importance Testing**

  After all of this feature engineering, SRK made an XGBoost model using the XGBoost package with some preset hyper-parameters over all of the features he has in he's data and tested it over a validation dataset he created prior to that, this model has trained on the train dataset again and again untill its rmse results havent decreased significantly for 50 rounds, and right after that he ploted the importance of the top 50 features.

  here we see that only full_sq and floor remained in the top 5 HI features, and that full_sq has a significantly high importance score compared to all other features, and he gives a warning that adding many variables will degrade the performance of the model.

c. Critique and Improvements

Simple Exploration Notebook - Sberbank

SRK gave a fairly detailed data exploration methodology and some vary compeling observations that helped him further on in the exploration and in his next notebook when he did the feature engineering and modeling.

while he gave many observations that helped in the further process he made in his next notebook, he completely missed or ignored many observations about missing values, skewing values and illogical values that are present in the features, which are fairly important to the machine learning pre-processes as they are indications for needed actions at the data cleaning stage.

all of these problematic values in the data are noticeble in hes exploration, and he only comments on tiny parts of them and does not handle them almost at all.

without f proper data cleaning prior to modeling, the resulting model will become worse and worse the 'dirtier' the data is, as there is more prominant noise for the model to catch and treat it as a behaviour represented in the data, thus becoming more and more off-course.

skewing values are noticeble at:

- log-Price's Distribution graph, where there are 2 relatively high spikes to the left of the distribution, which should potentially be considered outliers, and smoothed before modeling.
- log-Life_sq's & log-Price's [1D-2D]Distribution graph, where in the left side of the distribution we see a high and unusual spike, which heavily skewes the distribution of log-Life_sq and life_sq as well.

illogical values are noticeble at:

- log-Full_sq's & log-Price's [1D]Distribution-Scatter graph, where we see among the dots groups of dots with the exact same log-Full_sq value or the exact same log-Price value, something that is highly unlikely to happen, especially to this sizes of groups that we can see are creating almost completely continuous lines.

missing values are mentioned in its own graph but there is no observation made for the potential relationships the groups SRK mentions has and these relationships' affect on the model created later on.

Feature Engineering & Validation Strategy | Kaggle

SRK gave here as well fairly good examples, this time in the process of feature engineering.

the first noticeble (and in our eyes significant) difference, as mentioned in task 2.b., is the addition of the macro-economic features to the datasets to use them as important inforamtional features to create with other derived features.

here SRK starts with setting the values of all the prices that are higher in their value than the 99th precentile value as that 99th precentile value, and aswell for the 1st precentile for the prices that are lower in value from it, we dont agree with doing so because it creates an abnormality that affects the modeling and makes it a tad-bit 'clearer' that there is there some significant behaviour that happens in these 1st and 99th precentiles that in actuality does not exist, such a behaviour now would be easier to catch for the trees in the model, while this behaviour has been manually manifactured.

it would be better to decide whether to delete these samples or perform winsorization on them, instead of setting their value like that.

right after this, he performs label encoding, which is needed here in our opinion, but we see an issue with the performance, there is no consideration for missing values, misspelled values or any other problematic value that could be in a categorical feature, instead we think there should be done some proper data cleaning (as mentioned over at the previous notebook) so the label encoding would not be compromised with its values.

next there is a very creative, informative and inovative feature engineering in accordance to the different features and also with the amounts of missing values in a sample, after keeping such a feature SRK replaces all of the missing values with -99, this as some sort of a flag for the model of a missing value, this should be possible for the model to recognise as explained over at the previous notebook, and is a great additional way to use the information that could come out from using missing values like that, but we do not think it is helpful in our case because it is too generic in a way and not always would be recognised by the trees.

further more, in the rest of this notebook SRK does not check and give any consideration for removing 'volatile' features which have over 50% (or other any other threashold) missing values, which should be taken care of before the start of the modeling.

for us one more thing is missing before modeling and seeing the importance levels of the features and that is some form of feature filtering, we believe that the use of a correlation matrix is needed here and according to it SRK should have filtered all but one chosen arbitrarily out of each group of highly correlated features, this way we decrease (pretty sure significantly) the features that act as 'noise catchers'.

3. Model Enhancement

Prior to feature engineering, we need to perform our preprocessing after learning from the techniques used in the notebooks, and in addition some of our own ideas and opinions

in addition, please notice that across the preprocessing, data engineering, model generation and its performance comparison there are comments and explanation of the development process of the enhanced model as required in sub-task b

Pre-Processess

Data Import

this time we want to import the macro features that exist in the macro file as done in SRK's notebook '***Feature Engineering & Validation Strategy | Kaggle***', and use them in our feature engineering further on

In [ ]:
```
SbrBkEnhncTrain = pd.read_csv('train.csv', parse_dates=['timestamp']).filter(['full_sq', 'life_sq', 'floor', 'max_floor', 'material', 'build_year', 'num_room', 'kitch_sq'
SbrBkEnhncTest = pd.read_csv('test.csv', parse_dates=['timestamp']).filter(['id', 'full_sq', 'life_sq', 'floor', 'max_floor', 'material', 'build_year', 'num_room', 'kitch_
SbrBkEnhncTrain = pd.merge(SbrBkEnhncTrain, pd.read_csv('macro.csv', parse_dates=['timestamp']), how = 'left', on='timestamp')
SbrBkEnhncTest = pd.merge(SbrBkEnhncTest, pd.read_csv('macro.csv', parse_dates=['timestamp']), how = 'left', on='timestamp')
```

Data Cleaning

handling missing values

In [ ]:
```
res = (SbrBkEnhncTrain.isna().sum() / SbrBkEnhncTrain.shape[0] * 100).round(2).sort_values(ascending = False)

print(f'Precentages of missing values in Train:')
display(res.loc[res > 37])

print(f'Number of rows with missing values in Train: {SbrBkEnhncTrain.isna().any(axis = 1).sum()}')
print(f'Precentage of missing values overall in Train: {SbrBkEnhncTrain.isna().sum().sum() / (SbrBkEnhncTrain.shape[0] * SbrBkEnhncTrain.shape[1]) * 100:.3f}%')
```

```
Precentages of missing values in Train:
```

```
provision_retail_space_modern_sqm         97.53
provision_retail_space_sqm                81.65
museum_visitis_per_100_cap                55.47
theaters_viewers_per_1000_cap             55.47
load_of_teachers_preschool_per_teacher    55.47
students_reg_sports_share                 55.47
build_year                                44.65
state                                     44.50
dtype: float64
Number of rows with missing values in Train: 30471
Precentage of missing values overall in Train: 9.245%
```

We can see here that in 8 out of the 112 features we have available in the train dataset, over 37% of the values in them are missing values.

to us this is an indication that these features will bring more noise to the model further on then contribute effectively, so we'll remove them

notes:

- build_year has been removed here even though it was found as one of the top 5 HI features in *Simple Exploration Notebook - Sberbank*, this is because we prefer to lose that feature than use it and forgive over the noise or unwarented skewness it could give due to the missing values being imputed in some way, we could keep it and remove all or some of the rows that contain missing values in that feature, but this aswell is a high cost in our opinion, which we do not want to pay.
- kitch_sq should be removed aswell due to an observation in our baseline model preprocessing of this project.

In [ ]:
```
colsToDrop = res.loc[res > 37].index
SbrBkEnhncTrain.drop(columns = colsToDrop, inplace = True)
SbrBkEnhncTrain.drop(columns = ['kitch_sq'], inplace = True)
SbrBkEnhncTest.drop(columns = colsToDrop, inplace = True)
SbrBkEnhncTest.drop(columns = ['kitch_sq'], inplace = True)
print(f'{SbrBkEnhncTrain.shape[1]-1} features are left in the dataset.\n')
print(f'Number of rows with missing values in Train: {SbrBkEnhncTrain.isna().any(axis = 1).sum()}')
print(f'Precentage of missing values overall in Train: {SbrBkEnhncTrain.isna().sum().sum() / (SbrBkEnhncTrain.shape[0] * SbrBkEnhncTrain.shape[1]) * 100:.3f}%')
```

```
102 features are left in the dataset.

Number of rows with missing values in Train: 16157
Precentage of missing values overall in Train: 4.989%
```

this seems better!, now we should handle any illogical values.

for example:

- full_sq is not in the range (7, 500)
- life_sq is not in the range (7, 300)
- life_sq is greater than full_sq
- the average size of a room in the apartment is smaller than 2
- the floor of the apartment is higher than the max_floor in the building
- num_room is not in the range (1, 12)

notes:

- it should be mentioned that missing values has not been handled yet, and will be handled further on, but to try and find the illogical values before that they've been temporarily set to be an absurd value only for the search for illogical values

In [ ]:
```
temp_df = SbrBkEnhncTrain.copy(deep = True)
temp_df.fillna(-99, inplace = True)

illogicalTrain_df = pd.DataFrame({
    'illogicalFull_sq': (temp_df['full_sq'] < 7) | (temp_df['full_sq'] > 500),
    'illogicalLife_sq': (temp_df['life_sq'] < 7) | (temp_df['life_sq'] > 300),
    'illogicalLifeGreatFull': (temp_df['life_sq'] > temp_df['full_sq']) & (temp_df['full_sq'] != -99) & (temp_df['life_sq'] != -99),
    'illogicalLifeSize': (temp_df['life_sq'] / temp_df['num_room'] < 2) & (temp_df['life_sq'] != -99) & (temp_df['num_room'] != -99),
    'illogicalMax_floor': (temp_df['floor'] > temp_df['max_floor']) & (temp_df['max_floor'] != -99) & (temp_df['floor'] != -99),
    'illogicalNum_room': ((temp_df['num_room'] < 1) | (temp_df['num_room'] > 12)) & (temp_df['num_room'] != -99)
})

print(f'Precentages of illogical values in Train:\n{(illogicalTrain_df.sum() / illogicalTrain_df.shape[0] * 100).sort_values(ascending = False)}\n')
print(f'Number of rows with illogical values in Train: {illogicalTrain_df.any(axis = 1).sum()}')
print(f'Precentage of missing values overall in Train: {SbrBkEnhncTrain.isna().sum().sum() / (SbrBkEnhncTrain.shape[0] * SbrBkEnhncTrain.shape[1]) * 100:.3f}%')
```

```
Precentages of illogical values in Train:
illogicalLife_sq          22.408191
illogicalMax_floor         4.899741
illogicalLifeSize          1.401332
illogicalLifeGreatFull     0.121427
illogicalFull_sq           0.111582
illogicalNum_room          0.052509
dtype: float64

Number of rows with illogical values in Train: 7286
Precentage of missing values overall in Train: 4.989%
```

Out of all of the train data there are 7286 rows that have any illogical values in them, we can afford to remove these rows and sacrifice the contribution they could potentially hold.

we should check the same in the test!

In [ ]:
```
wrn.filterwarnings('ignore')

temp_df = SbrBkEnhncTest.copy(deep = True)
temp_df.fillna(-99, inplace = True)

illogicalTest_df = pd.DataFrame({
    'illogicalFull_sq': (temp_df['full_sq'] < 7) | (temp_df['full_sq'] > 500),
    'illogicalLife_sq': (temp_df['life_sq'] < 7) | (temp_df['life_sq'] > 300),
    'illogicalLifeGreatFull': (temp_df['life_sq'] > temp_df['full_sq']) & (temp_df['full_sq'] != -99) & (temp_df['life_sq'] != -99),
    'illogicalLifeSize': (temp_df['life_sq'] / temp_df['num_room'] < 2) & (temp_df['life_sq'] != -99) & (temp_df['num_room'] != -99),
    'illogicalMax_floor': (temp_df['floor'] > temp_df['max_floor']) & (temp_df['max_floor'] != -99) & (temp_df['floor'] != -99),
    'illogicalNum_room': ((temp_df['num_room'] < 1) | (temp_df['num_room'] > 12)) & (temp_df['num_room'] != -99)
})

print(f'Precentages of illogical values in Test:\n{(illogicalTest_df.sum() / illogicalTest_df.shape[0] * 100).sort_values(ascending = False)}\n')
print(f'Number of rows with illogical values in Test: {illogicalTest_df.any(axis = 1).sum()}')
print(f'Precentage of missing values overall in Test: {SbrBkEnhncTest.isna().sum().sum() / (SbrBkEnhncTest.shape[0] * SbrBkEnhncTest.shape[1]) * 100:.3f}%')
```

```
Precentages of illogical values in Test:
illogicalLife_sq          19.694597
illogicalMax_floor         8.392065
illogicalLifeSize          4.320021
illogicalLifeGreatFull     0.156617
illogicalFull_sq           0.039154
illogicalNum_room          0.013051
dtype: float64

Number of rows with illogical values in Test: 1651
Precentage of missing values overall in Test: 45.780%
```

In [ ]:
```python
# handling illogical values - train dataset
SbrBkEnhncTrain = SbrBkEnhncTrain.loc[illogicalTrain_df.any(axis = 1) != True,:]
res = (SbrBkEnhncTrain.isna().sum() / SbrBkEnhncTrain.shape[0] * 100).round(2).sort_values(ascending = False)
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(f'Precentages of missing values in Train:\n{res}\n')
print(f'Number of rows with missing values in Train: {SbrBkEnhncTrain.isna().any(axis = 1).sum()}')
print(f'Precentage of missing values overall in Train: {SbrBkEnhncTrain.isna().sum().sum() / (SbrBkEnhncTrain.shape[0] * SbrBkEnhncTrain.shape[1]) * 100:.3f}%')
# handling illogical values - test dataset
SbrBkEnhncTest.full_sq.loc[illogicalTest_df['illogicalFull_sq']] = np.nan
SbrBkEnhncTest.num_room.loc[illogicalTest_df['illogicalNum_room']] = np.nan
SbrBkEnhncTest.max_floor.loc[illogicalTest_df['illogicalMax_floor']] = np.nan
SbrBkEnhncTest.life_sq.loc[(illogicalTest_df['illogicalLife_sq']) | ((illogicalTest_df['illogicalLifeGreatFull']) & (~illogicalTest_df['illogicalFull_sq'])) | ((illogicalT
```

```
Precentages of missing values in Train:
max_floor                                    31.51
material                                     31.51
num_room                                     31.51
modern_education_share                       20.07
old_education_build_share                    20.07
profitable_enterpr_share                     10.05
baths_share                                  10.05
old_house_share                              10.05
heating_share                                10.05
electric_stove_share                         10.05
grp                                          10.05
grp_growth                                   10.05
hot_water_share                              10.05
real_dispos_income_per_cap_growth            10.05
gas_share                                    10.05
sewerage_share                               10.05
lodging_sqm_per_cap                          10.05
water_pipes_share                            10.05
perinatal_mort_per_1000_cap                  10.05
housing_fund_sqm                             10.05
pop_total_inc                                10.05
pop_migration                                10.05
invest_fixed_assets_phys                     10.05
unprofitable_enterpr_share                   10.05
share_own_revenues                           10.05
overdue_wages_per_cap                        10.05
fin_res_per_cap                              10.05
marriages_per_1000_cap                       10.05
infant_mortarity_per_1000_cap                10.05
incidence_population                         10.05
construction_value                           10.05
child_on_acc_pre_school                      10.05
apartment_build                              10.05
population_reg_sports_share                  10.05
hospital_bed_occupancy_per_year              10.05
hospital_beds_available_per_cap              10.05
power_clinics                                10.05
provision_doctors                            10.05
divorce_rate                                 10.05
floor                                         0.60
rent_price_1room_eco                          0.00
rent_price_2room_bus                          0.00
average_life_exp                              0.00
rent_price_3room_bus                          0.00
bandwidth_sports                              0.00
seats_theather_rfmin_per_100000_cap           0.00
turnover_catering_per_cap                     0.00
pop_natural_increase                          0.00
rent_price_1room_bus                          0.00
load_of_teachers_school_per_teacher           0.00
childbirth                                    0.00
mortality                                     0.00
rent_price_3room_eco                          0.00
rent_price_4+room_bus                         0.00
provision_nurse                               0.00
rent_price_2room_eco                          0.00
students_state_oneshift                       0.00
load_on_doctors                               0.00
full_sq                                       0.00
life_sq                                       0.00
invest_fixed_assets                           0.00
average_provision_of_build_contract           0.00
gdp_annual_growth                             0.00
gdp_annual                                    0.00
net_capital_export                            0.00
brent                                         0.00
eurrub                                        0.00
usdrub                                        0.00
balance_trade_growth                          0.00
balance_trade                                 0.00
gdp_deflator                                  0.00
ppi                                           0.00
cpi                                           0.00
gdp_quart_growth                              0.00
gdp_quart                                     0.00
oil_urals                                     0.00
timestamp                                     0.00
price_doc                                     0.00
sub_area                                      0.00
product_type                                  0.00
average_provision_of_build_contract_moscow    0.00
rts                                           0.00
micex                                         0.00
salary_growth                                 0.00
invest_fixed_capital_per_cap                  0.00
employment                                    0.00
unemployment                                  0.00
labor_force                                   0.00
retail_trade_turnover_growth                  0.00
retail_trade_turnover_per_cap                 0.00
retail_trade_turnover                         0.00
fixed_basket                                  0.00
salary                                        0.00
micex_rgbi_tr                                 0.00
income_per_cap                                0.00
mortgage_rate                                 0.00
mortgage_growth                               0.00
mortgage_value                                0.00
deposits_rate                                 0.00
deposits_growth                               0.00
deposits_value                                0.00
micex_cbi_tr                                  0.00
apartment_fund_sqm                            0.00
dtype: float64

Number of rows with missing values in Train: 9634
Precentage of missing values overall in Train: 4.629%
```

Out of all of the test data there are 1651 rows that have any illogical values in them, because this is the test dataset, we cannot afford to remove them, so instead we'll set them to be missing values in respect to the reason they are illogical.

There are 9634 rows with any missing values in the train dataset which is above 40% of the rows left, but overall these missing values make up only 4.629% of all the data we have left so far, and the features most affected by this have only 31.51% of their values missing.

therefor we can alow ourself to impute these missing values using KNNImputer like in our baseline model's preprocessing, we'll do so later on, we need to check first the test dataset for missing values as well, and handle it accordingly.

but just before that we want to handle the existing categorical features.

In [ ]:
```
for f in SbrBkEnhncTrain.columns:
    if SbrBkEnhncTrain[f].dtype=='object':
        print(f)
```

```
product_type
sub_area
child_on_acc_pre_school
modern_education_share
old_education_build_share
```

lets take a look at the features that are new to us and are categorical since we know how to handle the ones that were in the baseline model.

the features are:

- *child_on_acc_pre_school*
- *modern_education_share*
- *old_education_build_share*

In [ ]:
```
viewCategoricals = SbrBkEnhncTrain.columns[SbrBkEnhncTrain.dtypes == 'object']
for col in viewCategoricals[2:]:
    print(f'{SbrBkEnhncTrain[col].unique()}')
```

```
['#!' '7,311' '3,013' '16,765' nan]
[nan '90,92' '93,08' '95,4918']
[nan '23,14' '25,47' '8,2517']
```

these columns seem to be miss-understood when they were read from the csv to be written as categorical instead of numerical.

we would rather avoid handling them, since they do not seem to be mentioned as important features in the notebooks we analised, so we'll remove them.

In [ ]:
```
SbrBkEnhncTrain.drop(columns = viewCategoricals[2:], inplace = True)
SbrBkEnhncTest.drop(columns = viewCategoricals[2:], inplace = True)
```

now lets handle the features and their missing values in the test dataset

In [ ]:
```
res = (SbrBkEnhncTest.isna().sum() / SbrBkEnhncTest.shape[0] * 100).round(2).sort_values(ascending = False)

print(f'Precentages of missing values in Train:')
display(res.loc[res > 37])

print(f'Number of rows with missing values in Test: {SbrBkEnhncTest.isna().any(axis = 1).sum()}')
print(f'Precentage of missing values overall in Test: {SbrBkEnhncTest.isna().sum().sum() / (SbrBkEnhncTest.shape[0] * SbrBkEnhncTest.shape[1]) * 100:.3f}%')
```

```
Precentages of missing values in Train:
gas_share                              100.00
infant_mortarity_per_1000_cap          100.00
pop_total_inc                          100.00
construction_value                     100.00
grp                                    100.00
housing_fund_sqm                       100.00
lodging_sqm_per_cap                    100.00
water_pipes_share                      100.00
baths_share                            100.00
sewerage_share                         100.00
grp_growth                             100.00
hot_water_share                        100.00
electric_stove_share                   100.00
heating_share                          100.00
old_house_share                        100.00
divorce_rate                           100.00
perinatal_mort_per_1000_cap            100.00
invest_fixed_assets_phys               100.00
incidence_population                   100.00
real_dispos_income_per_cap_growth      100.00
marriages_per_1000_cap                 100.00
provision_doctors                      100.00
fin_res_per_cap                        100.00
overdue_wages_per_cap                  100.00
power_clinics                          100.00
hospital_beds_available_per_cap        100.00
hospital_bed_occupancy_per_year        100.00
share_own_revenues                     100.00
unprofitable_enterpr_share             100.00
profitable_enterpr_share               100.00
population_reg_sports_share            100.00
apartment_build                        100.00
pop_migration                          100.00
invest_fixed_assets                     51.98
pop_natural_increase                    51.98
childbirth                              51.98
mortality                               51.98
employment                              51.98
average_life_exp                        51.98
load_of_teachers_school_per_teacher     51.98
students_state_oneshift                 51.98
provision_nurse                         51.98
load_on_doctors                         51.98
turnover_catering_per_cap               51.98
seats_theather_rfmin_per_100000_cap     51.98
bandwidth_sports                        51.98
invest_fixed_capital_per_cap            51.98
apartment_fund_sqm                      51.98
unemployment                            51.98
retail_trade_turnover_growth            51.98
retail_trade_turnover_per_cap           51.98
retail_trade_turnover                   51.98
salary_growth                           51.98
salary                                  51.98
income_per_cap                          51.98
labor_force                             51.98
dtype: float64
```

```
Number of rows with missing values in Test: 7662
Precentage of missing values overall in Test: 45.243%
```

it seems like there are many features in the test dataset that have over 37% of their values missing values, all these features should be removed aswell.

In [ ]:
```
colsToDrop = res.loc[res > 37].index
SbrBkEnhncTrain.drop(columns = colsToDrop, inplace = True)
SbrBkEnhncTest.drop(columns = colsToDrop, inplace = True)
res = (SbrBkEnhncTest.isna().sum() / SbrBkEnhncTest.shape[0] * 100).round(2).sort_values(ascending = False)

print(f'Precentages of missing values in Test:')
display(res.loc[res > 37])

print(f'Number of rows with missing values in Test: {SbrBkEnhncTest.isna().any(axis = 1).sum()}')
print(f'Precentage of missing values overall in Test: {SbrBkEnhncTest.isna().sum().sum() / (SbrBkEnhncTest.shape[0] * SbrBkEnhncTest.shape[1]) * 100:.3f}%')
```

```
Precentages of missing values in Test:
Series([], dtype: float64)
Number of rows with missing values in Test: 1677
Precentage of missing values overall in Test: 0.651%
```

now this looks much better

lets impute the missing values now.

In [ ]:
```
wrn.filterwarnings('ignore')

SbrBkEnhncTrain['product_typeCode'] = pd.Categorical(SbrBkEnhncTrain.product_type).codes
SbrBkEnhncTest['product_typeCode'] = pd.Categorical(SbrBkEnhncTest.product_type).codes
SbrBkEnhncTest.product_typeCode.loc[SbrBkEnhncTest.product_typeCode == -1] = np.nan
SbrBkEnhncTrain['sub_areaCode'] = pd.Categorical(SbrBkEnhncTrain.sub_area).codes
SbrBkEnhncTest['sub_areaCode'] = pd.Categorical(SbrBkEnhncTest.sub_area).codes

tempTrain = SbrBkEnhncTrain.copy(deep = True).reset_index(drop = True)
tempTest = SbrBkEnhncTest.copy(deep = True).reset_index(drop = True)
tempTrain.drop(columns = ['product_type', 'sub_area', 'price_doc', 'timestamp'], inplace = True)
tempTest.drop(columns = ['product_type', 'sub_area', 'id', 'timestamp'], inplace = True)
cols = list(tempTrain.columns)

colsToRoundTrain = tempTrain.columns[tempTrain.isna().any(axis = 0)]
colsToRoundTest = tempTest.columns[tempTest.isna().any(axis = 0)]

imputedEnhncTrainData = KNNImputer(n_neighbors = 20).fit_transform(tempTrain)
imputedEnhncTestData = KNNImputer(n_neighbors = 20).fit_transform(tempTest)
imputedEnhncTrain = pd.DataFrame(imputedEnhncTrainData, columns = cols)
imputedEnhncTest = pd.DataFrame(imputedEnhncTestData, columns = cols)
imputedEnhncTrain.drop(columns = ['product_typeCode', 'sub_areaCode'], inplace = True)
imputedEnhncTrain[colsToRoundTrain] = imputedEnhncTrain[colsToRoundTrain].round()
imputedEnhncTest[colsToRoundTest] = imputedEnhncTest[colsToRoundTest].round()
imputedEnhncTrain = imputedEnhncTrain.set_index(SbrBkEnhncTrain.index)
imputedEnhncTest = imputedEnhncTest.set_index(SbrBkEnhncTest.index)
imputedEnhncTrain['product_type'] = SbrBkEnhncTrain.product_type
imputedEnhncTest['product_type'] = SbrBkEnhncTest.product_type
imputedEnhncTrain['sub_area'] = SbrBkEnhncTrain.sub_area
imputedEnhncTest['sub_area'] = SbrBkEnhncTest.sub_area
imputedEnhncTrain['price_doc'] = SbrBkEnhncTrain.price_doc
imputedEnhncTest['id'] = SbrBkEnhncTest.id
imputedEnhncTrain['timestamp'] = SbrBkEnhncTrain.timestamp
imputedEnhncTest['timestamp'] = SbrBkEnhncTest.timestamp
imputedEnhncTest.product_type.loc[imputedEnhncTest.product_type.isna()] = np.where(imputedEnhncTest.product_typeCode.loc[imputedEnhncTest.product_type.isna()] == 0, 'Inves
imputedEnhncTest.drop(columns = ['product_typeCode', 'sub_areaCode'], inplace = True)
imputedEnhncTrain['logPrice'] = np.log(imputedEnhncTrain.price_doc)
```

we should make sure the imputation has worked properly!

In [ ]:
```
res = (imputedEnhncTrain.isna().sum() / imputedEnhncTrain.shape[0] * 100).round(2).sort_values(ascending = False)

print(f'Precentages of missing values in Train:')
display(res.loc[res > 37])

print(f'Number of rows with missing values in Train: {imputedEnhncTrain.isna().any(axis = 1).sum()}')
print(f'Precentage of missing values overall in Train: {imputedEnhncTrain.isna().sum().sum() / (imputedEnhncTrain.shape[0] * imputedEnhncTrain.shape[1]) * 100:.3f}%')

print('\n')
res = (imputedEnhncTest.isna().sum() / imputedEnhncTest.shape[0] * 100).round(2).sort_values(ascending = False)

print(f'Precentages of missing values in Test:')
display(res.loc[res > 37])

print(f'Number of rows with missing values in Test: {imputedEnhncTest.isna().any(axis = 1).sum()}')
print(f'Precentage of missing values overall in Test: {imputedEnhncTest.isna().sum().sum() / (imputedEnhncTest.shape[0] * imputedEnhncTest.shape[1]) * 100:.3f}%')
```

```
Precentages of missing values in Train:
Series([], dtype: float64)
Number of rows with missing values in Train: 0
Precentage of missing values overall in Train: 0.000%


Precentages of missing values in Test:
Series([], dtype: float64)
Number of rows with missing values in Test: 0
Precentage of missing values overall in Test: 0.000%
```

Perfect!

finally we can label encode the categorical features we found earlier and after that start with feature engineering.

In [ ]:
```
imputedEnhncTrain['originalIndex'] = imputedEnhncTrain.index*10+1
imputedEnhncTest['originalIndex'] = imputedEnhncTest.index*10
imputedEnhncUnion = pd.concat([imputedEnhncTrain, imputedEnhncTest], ignore_index=True)
prodTypeLbl = LabelEncoder().fit(imputedEnhncUnion.product_type)
subAreaLbl = LabelEncoder().fit(imputedEnhncUnion.sub_area)
imputedEnhncTrain.product_type = prodTypeLbl.transform(imputedEnhncTrain.product_type)
imputedEnhncTest.product_type = prodTypeLbl.transform(imputedEnhncTest.product_type)
imputedEnhncTrain.sub_area = subAreaLbl.transform(imputedEnhncTrain.sub_area)
imputedEnhncTest.sub_area = subAreaLbl.transform(imputedEnhncTest.sub_area)
```

a. Feature Engineering

as we mentioned in our answers for task 2.b., SRK made several features, all under logical framework, so we did aswell!

we made the following features:

- **year**, **month**, **dayOfWeek** & **season**

  Resoning: to take into acount the changes dependent on time itself as a numerical value rather than as a characterised value.

- **fullLifeRatio**

  Resoning: the ratio between full_sq and life_sq could give many implications on the apartment and in accordance to other features could give a significant indication to the price of the house

- **usdeur**: USD/EUR exchange rate

  Resoning: this feature should indicate about the affect of changes in the US Dollar and European Euro exchange rate in accordance to oil dependent features like brent and oil_urals on the overall state of the economy in russia and by proxy on the state of the housing market.

- **avgRoomSize**: average room size

  Resoning: the bigger the rooms the more comfortable they are, therefor, the higher the price of the apartment.

- **interesetSpread**: mortgage interest rate & deposits interest rate difference

  Resoning: the intrest rate of a mortgage has somewhat of a 'repeling force' towards the population, as well as the deposits interest rate but in the opposit way, meaning the farther apart they are, the more affect on the population to take a mortgage and/or put a deposit and make a transaction in the housing market, and by proxy affect the prices in the housing market.

- **depostMortggRatio**: deposit_value & mortgage_value Ratio

  Resoning: this represents the balance between the savings and the loans of the population, when its higher than 1 there are more deposits than mortgages, which means there are more savings than loans in the market, this is true on the opposite way, and by proxy affects the housing market and the prices in it.

- **brentUralsDiff**, **brentUralsDiff14**: oil_urals & brent difference and it MA over 14 days

  Resoning: these features come to answer the question: are the oil prices in russia more attractive than the current world standard prices?, this by showing a positive or negative difference between them, the more atractive the oil prices, the better the state of the russian economy, and by proxy the state of the housing market and its prices.

- **soldAmnt30day**: apartments sold in current area in the last 30 days, demand in the area across current month

  Resoning: demand of apartments in the area and time of the current sample in the recent past, the higher the demand the higher the price should be.

- **rtsVolatility**, **micexVolatility**: market volatility: sd of the Index RTS and of the MICEX Index over 90 days

  Resoning: how volatile, unpredictable and unsafe has the russian stock market has been over the last 90 days (roughly a quarter-year), the more volatile the russian stock market is, the more unstable the russian economy becomes, and by proxy the housing market and its prices.

- **mortgageGrowthMA3**, **depositGrowthMA3**: MA per month for 3 months periods of mortgage_growth and of deposits_growth

  Resoning: a moving average of recent quarter of the mortgage_growth and the deposits_growth would show a partially smoothed out value representing the quarter-volatility of the mortgages and deposits, when it stays the same for longer, the housing market should stay more predictable and safe to rely on as information used by the population in the housing market.

- **basketInflationMA3**: MA per month for 3 months periods of the fixed basket inflation.

  Resoning: The higher the inflation of the fixed basket is, the harder it is for the russian population to estimate their prefrences over using their income on necessities or in the housing market.

In [ ]:
```
imputedEnhncTrain = imputedEnhncTrain.assign(year = imputedEnhncTrain.timestamp.dt.year,
                                             month = imputedEnhncTrain.timestamp.dt.month,
                                             dayOfWeek = imputedEnhncTrain.timestamp.dt.dayofweek,
                                             season = imputedEnhncTrain.timestamp.dt.month % 12 // 3 + 1, # 1 = Winter, 2 = Spring, 3 = Summer, 4 = Autumn
                                             fullLifeRatio = imputedEnhncTrain.life_sq / imputedEnhncTrain.full_sq,
                                             usdeur = imputedEnhncTrain.usdrub / imputedEnhncTrain.eurrub,
                                             avgRoomSize = imputedEnhncTrain.life_sq / imputedEnhncTrain.num_room,
                                             interestSpread = imputedEnhncTrain.mortgage_rate - imputedEnhncTrain.deposits_rate,
                                             depostMortggRatio = imputedEnhncTrain.deposits_value / imputedEnhncTrain.mortgage_value)
imputedEnhncTest = imputedEnhncTest.assign(year = imputedEnhncTest.timestamp.dt.year,
                                           month = imputedEnhncTest.timestamp.dt.month,
                                           dayOfWeek = imputedEnhncTest.timestamp.dt.dayofweek,
                                           season = imputedEnhncTest.timestamp.dt.month % 12 // 3 + 1, # 1 = Winter, 2 = Spring, 3 = Summer, 4 = Autumn
                                           fullLifeRatio = imputedEnhncTest.life_sq / imputedEnhncTest.full_sq,
                                           usdeur = imputedEnhncTest.usdrub / imputedEnhncTest.eurrub,
                                           avgRoomSize = imputedEnhncTest.life_sq / imputedEnhncTest.num_room,
                                           interestSpread = imputedEnhncTest.mortgage_rate - imputedEnhncTest.deposits_rate,
                                           depostMortggRatio = imputedEnhncTest.deposits_value / imputedEnhncTest.mortgage_value)

imputedEnhncUnion["numOday"] = (imputedEnhncUnion.timestamp - imputedEnhncUnion.timestamp.min()).dt.days
imputedEnhncUnion["soldAmnt30day"] = imputedEnhncUnion[["numOday", "sub_area"]].apply(lambda x: ((x.numOday - imputedEnhncUnion.numOday < 31) & (x.numOday - imputedEnhncUn
imputedEnhncTrain = pd.merge(imputedEnhncTrain, imputedEnhncUnion[['originalIndex', 'soldAmnt30day']], how = 'left', on = 'originalIndex')
imputedEnhncTest = pd.merge(imputedEnhncTest, imputedEnhncUnion[['originalIndex', 'soldAmnt30day']], how = 'left', on = 'originalIndex')
imputedEnhncTrain.drop(columns = ['originalIndex'], inplace = True)
imputedEnhncTest.drop(columns = ['originalIndex'], inplace = True)

macroDf = pd.read_csv('macro.csv', parse_dates=['timestamp'])

macroDf["rtsVolatility"] = macroDf.rts.rolling(window = 90).std()
macroDf["micexVolatility"] = macroDf.micex.rolling(window = 90).std()
macroDf["uralBrentDiff"] = macroDf.brent - macroDf.oil_urals
macroDf["uralBrentDiff14"] = macroDf.uralBrentDiff.rolling(window = 14, min_periods = 1).mean()
macroDf["yearNmonth"] = macroDf.timestamp.dt.year * 100 + macroDf.timestamp.dt.month

monthMacroDf = macroDf.drop_duplicates(subset = ['yearNmonth'])
monthMacroDf["basketInflation"] = monthMacroDf.fixed_basket / monthMacroDf.fixed_basket.shift(1) - 1
monthMacroDf["basketInflationMA3"] = monthMacroDf.basketInflation.rolling(window = 3, min_periods = 1).mean()
monthMacroDf["mortgageGrowthMA3"] = monthMacroDf.mortgage_growth.rolling(window = 3, min_periods = 1).mean()
monthMacroDf["depositGrowthMA3"] = monthMacroDf.deposits_growth.rolling(window = 3, min_periods = 1).mean()

macroDf = pd.merge(macroDf, monthMacroDf[['yearNmonth', 'basketInflationMA3', 'mortgageGrowthMA3', 'depositGrowthMA3']], how = 'left', on = 'yearNmonth')

imputedEnhncTrain = pd.merge(imputedEnhncTrain, macroDf[['timestamp', 'rtsVolatility', 'micexVolatility', 'uralBrentDiff', 'uralBrentDiff14', 'basketInflationMA3', 'mortga
imputedEnhncTest = pd.merge(imputedEnhncTest, macroDf[['timestamp', 'rtsVolatility', 'micexVolatility', 'uralBrentDiff', 'uralBrentDiff14', 'basketInflationMA3', 'mortgage
```

now lets make sure there are no issues in these new features

In [ ]:
```
display(imputedEnhncTrain.columns[imputedEnhncTrain.isna().sum()>0])
display(imputedEnhncTest.columns[imputedEnhncTest.isna().sum()>0])
```

```
Index([], dtype='object')
Index([], dtype='object')
```

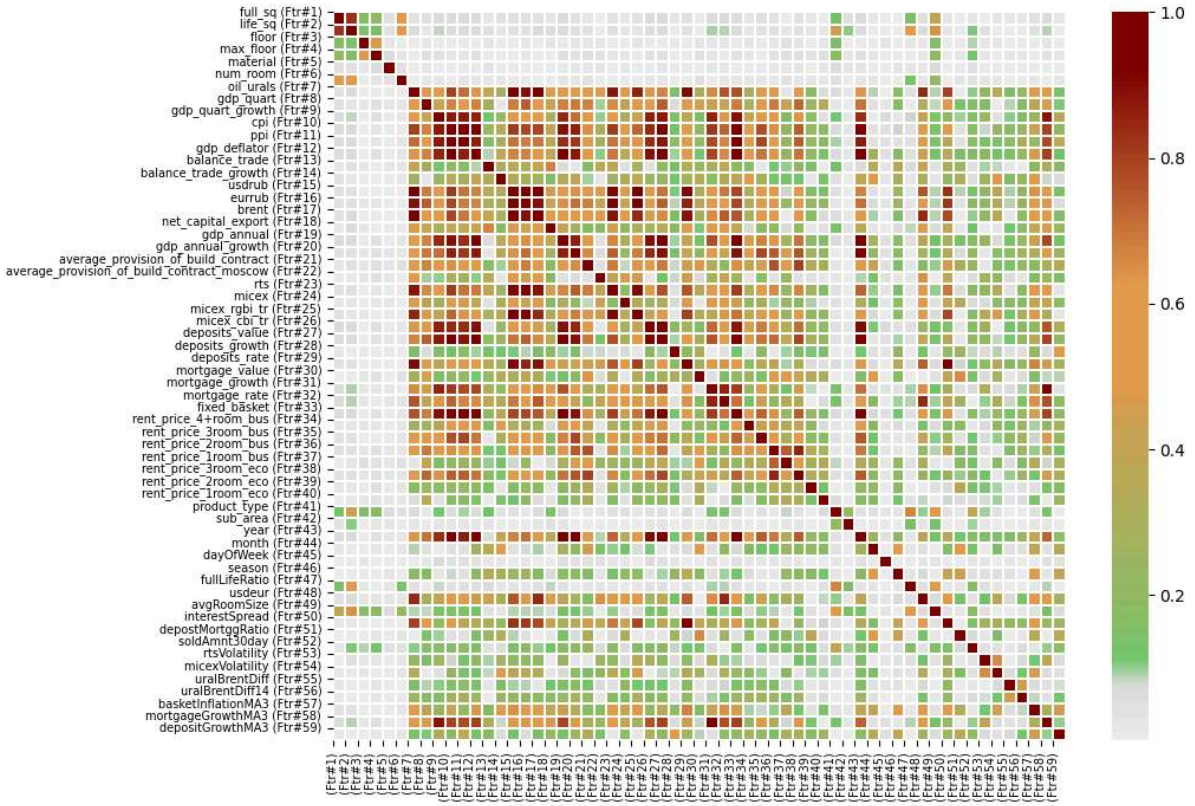
great!

we need now to filter the best features to model upon later on, we'll do so by taking into acount their multicolinearity and their importance.

but just before that we need to separte them to train and validation sets.

In [ ]:
```
xTrn = imputedEnhncTrain.loc[:int(imputedEnhncTrain.shape[0] * 70 / 85), :].drop(columns = ['price_doc', 'logPrice', 'timestamp'])
yTrn = imputedEnhncTrain.loc[:int(imputedEnhncTrain.shape[0] * 70 / 85), 'logPrice']
xVal = imputedEnhncTrain.loc[int(imputedEnhncTrain.shape[0] * 70 / 85):, :].drop(columns = ['price_doc', 'logPrice', 'timestamp'])
yVal = imputedEnhncTrain.loc[int(imputedEnhncTrain.shape[0] * 70 / 85):, 'logPrice']
```

In [ ]:
```
corrMatrix = imputedEnhncTrain.drop(columns = ['timestamp', 'price_doc', 'logPrice']).corr().abs()
fig, axs = plt.subplots(figsize = (10, 8))
ylabels = [col + f' (Ftr#{i+1})' for i, col in enumerate(corrMatrix.columns)]
xlabels = [f' (Ftr#{i+1})' for i in range(corrMatrix.shape[0])]
sns.heatmap(corrMatrix, cmap = custom_cmap, linewidths = 0.1)
plt.xticks(ticks = range(corrMatrix.shape[1]), labels = xlabels, fontsize = 7)
plt.yticks(ticks = range(corrMatrix.shape[0]), labels = ylabels, fontsize = 7)
plt.show()
```



it is clear when looking at this correlation metrix that there are groups that correlate with one another in various levels.

we'll take into acount only the sevearly correlated features and filter out some of the features to lower the multicolinearity the features in the data have, doing so while taking into acount the importance level they have to not remove features that are important and helpful.

the importance level we'll get using the hyper-parameters we got in the baseline model just to have some relatively stable and apropriate footing.

In [ ]:
```
importanceRndFrstModel = RFR(n_jobs=-1, random_state = 42, **RndFrstModelParams)
importanceRndFrstModel.fit(xTrn, yTrn)
importance = pd.Series(importanceRndFrstModel.feature_importances_, index = xTrn.columns).sort_values(ascending = True)
importance = importance/importance.max()*100

groups = []
corr_threshold = 0.75
notInGroup = pd.Series([True for i in range(corrMatrix.shape[1])], index = corrMatrix.columns)

for feature in corrMatrix.columns:
    if not groups:
        groups.append([feature])
        notInGroup[feature] = False
    else:
        if notInGroup[feature]:
            currGroup = [feature]
            notInGroup[feature] = False
            for f in corrMatrix.columns:
                if (corrMatrix.loc[f, feature] >= corr_threshold) & (notInGroup[f]):
                    currGroup.append(f)
                    notInGroup[f] = False
            groups.append(currGroup)

# before filtering the features, we make a copy of the dataframes for further use
rndFrst_xTrn = xTrn.copy(deep = True)
rndFrst_xVal = xVal.copy(deep = True)
rndFrst_imputedEnhncTrain = imputedEnhncTrain.copy(deep = True)
rndFrst_imputedEnhncTest = imputedEnhncTest.copy(deep = True)
rndFrst_corrMatrix = corrMatrix.copy(deep = True)

print(f'Feature Filteration Log - Enhanced Random Forest Model\n=======================================================\n    threshold = {corr_threshold}\n')

removed = []

for i, group in enumerate(groups):
```

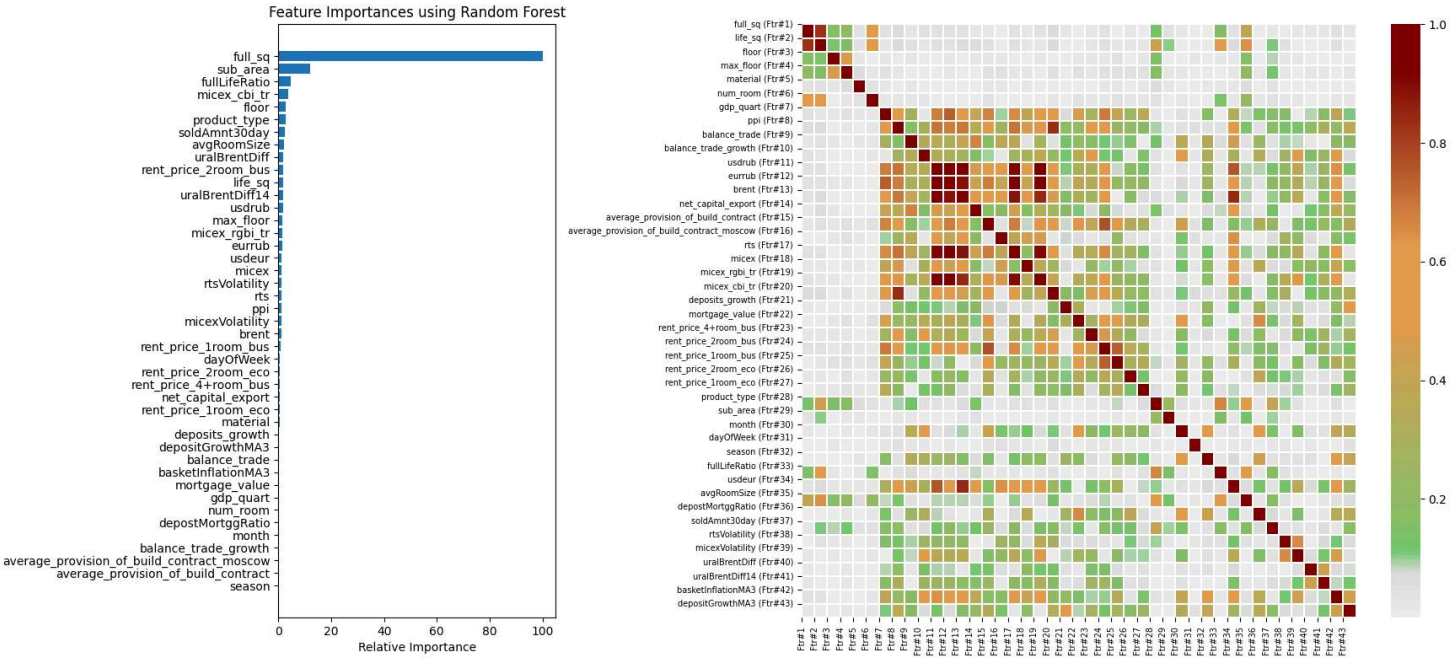
```
    if len(group) == 1:
        print(f"Group {i+1}:\n    - the feature ['{group[0]}'] is not correlated enough to any other feature.")
    else:
        avgImportance = importance.loc[group].mean()
        col2Keep = importance.loc[group][importance.loc[group] >= avgImportance].index
        removed += [col for col in group if col not in col2Keep]
        rndFrst_imputedEnhncTrain = rndFrst_imputedEnhncTrain[[col for col in rndFrst_imputedEnhncTrain.columns if col not in [c for c in group if c not in col2Keep]]]
        rndFrst_imputedEnhncTest = rndFrst_imputedEnhncTest[[col for col in rndFrst_imputedEnhncTest.columns if col not in [c for c in group if c not in col2Keep]]]
        rndFrst_xTrn = rndFrst_xTrn[[col for col in rndFrst_xTrn.columns if col not in [c for c in group if c not in col2Keep]]]
        rndFrst_xVal = rndFrst_xVal[[col for col in rndFrst_xVal.columns if col not in [c for c in group if c not in col2Keep]]]
        rndFrst_corrMatrix = rndFrst_corrMatrix.loc[[col for col in rndFrst_corrMatrix.columns if col not in [c for c in group if c not in col2Keep]], [col for col in rndF
        print(f"Group {i+1}:\n    - features kept- {list(col2Keep)}\n    - features removed- {[col for col in group if col not in col2Keep]}")

print(f'removed {len(removed)} features:\n    - {removed}')
```

```
Feature Filteration Log - Enhanced Random Forest Model
======================================================
    threshold = 0.75

Group 1:
    - the feature ['full_sq'] is not correlated enough to any other feature.
Group 2:
    - the feature ['life_sq'] is not correlated enough to any other feature.
Group 3:
    - the feature ['floor'] is not correlated enough to any other feature.
Group 4:
    - the feature ['max_floor'] is not correlated enough to any other feature.
Group 5:
    - the feature ['material'] is not correlated enough to any other feature.
Group 6:
    - the feature ['num_room'] is not correlated enough to any other feature.
Group 7:
    - features kept- ['usdrub', 'eurrub', 'brent', 'rts', 'micex_rgbi_tr', 'usdeur']
    - features removed- ['oil_urals', 'cpi', 'deposits_rate', 'mortgage_rate', 'fixed_basket', 'interestSpread']
Group 8:
    - the feature ['gdp_quart'] is not correlated enough to any other feature.
Group 9:
    - features kept- ['ppi', 'micex_cbi_tr']
    - features removed- ['gdp_quart_growth', 'gdp_deflator', 'gdp_annual', 'deposits_value', 'mortgage_growth', 'year', 'mortgageGrowthMA3']
Group 10:
    - the feature ['balance_trade'] is not correlated enough to any other feature.
Group 11:
    - the feature ['balance_trade_growth'] is not correlated enough to any other feature.
Group 12:
    - the feature ['net_capital_export'] is not correlated enough to any other feature.
Group 13:
    - features kept- ['rent_price_2room_bus']
    - features removed- ['gdp_annual_growth', 'rent_price_3room_bus', 'rent_price_3room_eco']
Group 14:
    - the feature ['average_provision_of_build_contract'] is not correlated enough to any other feature.
Group 15:
    - the feature ['average_provision_of_build_contract_moscow'] is not correlated enough to any other feature.
Group 16:
    - the feature ['micex'] is not correlated enough to any other feature.
Group 17:
    - the feature ['deposits_growth'] is not correlated enough to any other feature.
Group 18:
    - the feature ['mortgage_value'] is not correlated enough to any other feature.
Group 19:
    - the feature ['rent_price_4+room_bus'] is not correlated enough to any other feature.
Group 20:
    - the feature ['rent_price_1room_bus'] is not correlated enough to any other feature.
Group 21:
    - the feature ['rent_price_2room_eco'] is not correlated enough to any other feature.
Group 22:
    - the feature ['rent_price_1room_eco'] is not correlated enough to any other feature.
Group 23:
    - the feature ['product_type'] is not correlated enough to any other feature.
Group 24:
    - the feature ['sub_area'] is not correlated enough to any other feature.
Group 25:
    - the feature ['month'] is not correlated enough to any other feature.
Group 26:
    - the feature ['dayOfWeek'] is not correlated enough to any other feature.
Group 27:
    - the feature ['season'] is not correlated enough to any other feature.
Group 28:
    - the feature ['fullLifeRatio'] is not correlated enough to any other feature.
Group 29:
    - the feature ['avgRoomSize'] is not correlated enough to any other feature.
Group 30:
    - the feature ['depostMortggRatio'] is not correlated enough to any other feature.
Group 31:
    - the feature ['soldAmnt30day'] is not correlated enough to any other feature.
Group 32:
    - the feature ['rtsVolatility'] is not correlated enough to any other feature.
Group 33:
    - the feature ['micexVolatility'] is not correlated enough to any other feature.
Group 34:
    - the feature ['uralBrentDiff'] is not correlated enough to any other feature.
Group 35:
    - the feature ['uralBrentDiff14'] is not correlated enough to any other feature.
Group 36:
    - the feature ['basketInflationMA3'] is not correlated enough to any other feature.
Group 37:
    - the feature ['depositGrowthMA3'] is not correlated enough to any other feature.
removed features:
    - ['oil_urals', 'cpi', 'deposits_rate', 'mortgage_rate', 'fixed_basket', 'interestSpread', 'gdp_quart_growth', 'gdp_deflator', 'gdp_annual', 'deposits_value', 'mortgag
e_growth', 'year', 'mortgageGrowthMA3', 'gdp_annual_growth', 'rent_price_3room_bus', 'rent_price_3room_eco']
```

we should take a look at the importance levels and the updated correlation matrix to determin which features to keep for the model itself, since this is to many features which might act as noise catchers.

In [ ]:
```
importanceRndFrstModel = RFR(n_jobs=-1, random_state = 42, **RndFrstModelParams)
importanceRndFrstModel.fit(rndFrst_xTrn, yTrn)
importance = pd.Series(importanceRndFrstModel.feature_importances_, index = rndFrst_xTrn.columns).sort_values(ascending = True)
importance = importance/importance.max()*100

fig = plt.figure(figsize = (18, 8))
gs = GS.GridSpec(1, 2, width_ratios=[2, 5])
```

```
ax0 = plt.subplot(gs[0])
ax1 = plt.subplot(gs[1])

ax0.set_title('Feature Importances using Random Forest')
ax0.barh(importance.index, importance.values, align='center')
ax0.set_xlabel('Relative Importance')

corrMatrixTicks = range(rndFrst_corrMatrix.shape[0])
ylabels = [col + f' (Ftr#{i+1})' for i, col in enumerate(rndFrst_corrMatrix.columns)]
xlabels = [f'Ftr#{i+1}' for i in corrMatrixTicks]
sns.heatmap(rndFrst_corrMatrix, cmap = custom_cmap, linewidths = 0.1, ax=ax1)
ax1.set_xticks(ticks = corrMatrixTicks)
ax1.set_xticklabels(labels = xlabels, fontsize = 7)
ax1.set_yticks(ticks = corrMatrixTicks)
ax1.set_yticklabels(labels = ylabels, fontsize = 7)

plt.tight_layout()
plt.show()
```



it is very clear that full_sq is the most important feature, and that sub_area and fullLifeRatio are the second and third most important features respectively.

the rest of the features, even though they have an order of their usefulness to the model, determined by their importance, it is hard to say if they have any significance to the model.

but keeping only 3 features seem to be lacking, so to be sure we decided to also keep the top 7 features after full_sq, sub_area and fullLifeRatio, leaving us with 10 features to model upon.

In [ ]:
```
rndFrst_features2drop = importance.head(importance.shape[0] - 10).index

rndFrst_xTrn.drop(columns = rndFrst_features2drop, inplace = True)
rndFrst_xVal.drop(columns = rndFrst_features2drop, inplace = True)
rndFrst_imputedEnhncTrain.drop(columns = rndFrst_features2drop, inplace = True)
rndFrst_imputedEnhncTest.drop(columns = rndFrst_features2drop, inplace = True)
```

since the baseline model we made had almost all of its hyper-parameters picked as the maximum value available to the cross validation process, we'll perform the cross validation on a higher and wider range of values.

In [ ]:
```
enhncRndFrstParams = {
    'max_depth': [i for i in range(6, 17, 2)], # 6, 8, 10, 12, 14, 16
    'min_samples_leaf': [i for i in range(8, 44, 7)], # 8, 15, 22, 29, 36, 43
    'n_estimators': [i for i in range(175, 426, 50)] # 175, 225, 275, 325, 375, 425
}
enhncRndFrstModel = RFR()
enhncRndFrstGrdSrch = GSCV(enhncRndFrstModel, enhncRndFrstParams, cv = 5, scoring = 'neg_mean_squared_error', verbose = 2)
enhncRndFrstGrdSrch.fit(rndFrst_xTrn, yTrn)

enhncRndFrstModelParams = enhncRndFrstGrdSrch.best_params_

enhncRndFrstOptModel = RFR(**enhncRndFrstModelParams)
enhncRndFrstOptModel.fit(rndFrst_xTrn, yTrn)
```

In [ ]:
```
shortcutParams = {
    'max_depth': 12,
    'min_samples_leaf': 8,
    'n_estimators': 375
}
enhncRndFrstOptModel = RFR(**shortcutParams)
enhncRndFrstOptModel.fit(rndFrst_xTrn, yTrn)
```

Out[ ]:
```
RandomForestRegressor
RandomForestRegressor(max_depth=12, min_samples_leaf=8, n_estimators=375)
```

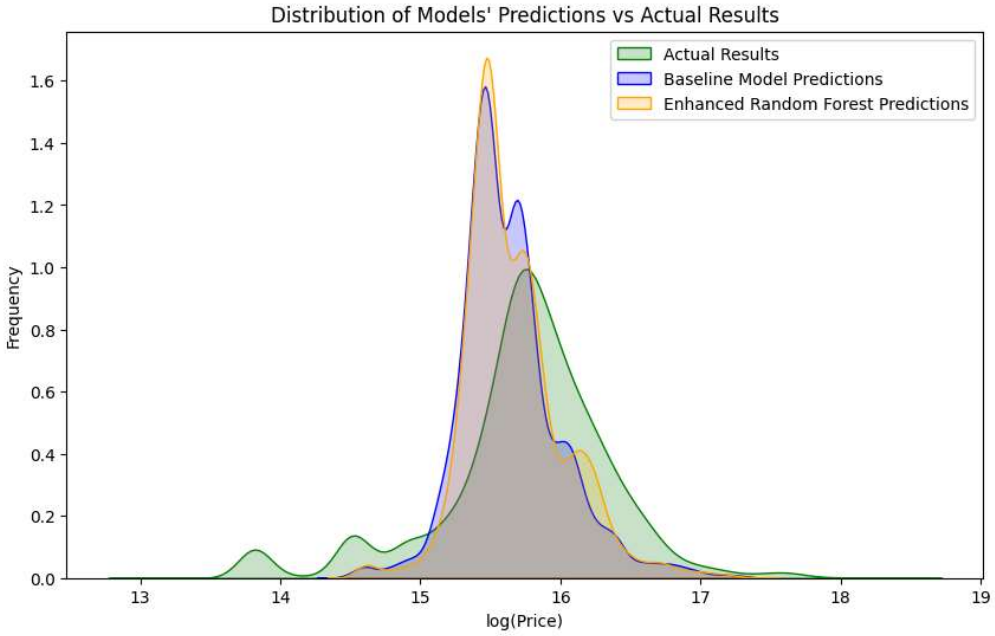
In [ ]:
```
enhncRndFrstMseScores = -1 * CVS(enhncRndFrstOptModel, rndFrst_xVal, yVal, cv = 5, scoring = 'neg_mean_squared_error')
enhncRndFrstPred = enhncRndFrstOptModel.predict(rndFrst_xVal)
enhncRndFrstRmsle = np.sqrt(np.mean((enhncRndFrstPred - yVal) ** 2))
enhncRndFrstR2 = enhncRndFrstOptModel.score(rndFrst_xVal, yVal)

metrics.loc[metrics.shape[0]] = {'Model': 'Enhanced Random Forest Model',
                                 'MSE': enhncRndFrstMseScores.mean(),
                                 'RMSLE': enhncRndFrstRmsle,
                                 'R^2': enhncRndFrstR2}

display(metrics)
```

	Model	MSE	RMSLE	R^2
0	Random Forest Model	0.192769	0.449120	0.409938
1	Enhanced Random Forest Model	0.255073	0.505923	0.326398

this does not seem to improve under the MSE & RMSLE metrics, but does under the R^2 which gives a reason to be optimistic.

lets look at the distribution

In [ ]:
```
# random forest comparison
plt.figure(figsize=(10, 6))
sns.kdeplot(yVal, color = 'green', fill = True, alpha = 0.2, label='Actual Results')
sns.kdeplot(rndFrstPred, color = 'blue', fill = True, alpha = 0.2, label='Baseline Model Predictions')
sns.kdeplot(enhncRndFrstPred, color = 'orange', fill = True, alpha = 0.2, label='Enhanced Random Forest Predictions')
plt.xlabel('log(Price)')
plt.ylabel('Frequency')
plt.title("Distribution of Models' Predictions vs Actual Results")
plt.legend()
plt.show()
```



this distribution seem to be more concetrated than its predecessor, making us more pasimistic on its performance

In [ ]:
```
yPredEnhncRndFrst = np.round(np.exp(enhncRndFrstOptModel.predict(rndFrst_imputedEnhncTest.drop(columns = ['id', 'timestamp']))), decimals = -3)
enhncRndFrstSubmission = pd.DataFrame({'id': rndFrst_imputedEnhncTest.id, 'price_doc': yPredEnhncRndFrst})
enhncRndFrstSubmission.to_csv('EnhancedRandomForestSubmission.csv', index = False)
```

when submitting the predictions test data we get the following results in comparison to previous models:

	Private Score	Public Score
Random Forest Model	0.38628	0.38735
Enhanced Random Forest Model	0.37493	0.37117

we can see that in comparison to the baseline random forest model our enhanced model has improved!

4. Advanced Modeling

a. Model Implementation

now we should implement the model under XGBoost framework, this is because rather than building a Random Forest model which relys on Bagging, XGBoost (as its name suggests) relys on Boosting.

the main differences between Boosting and Bagging are as follows:

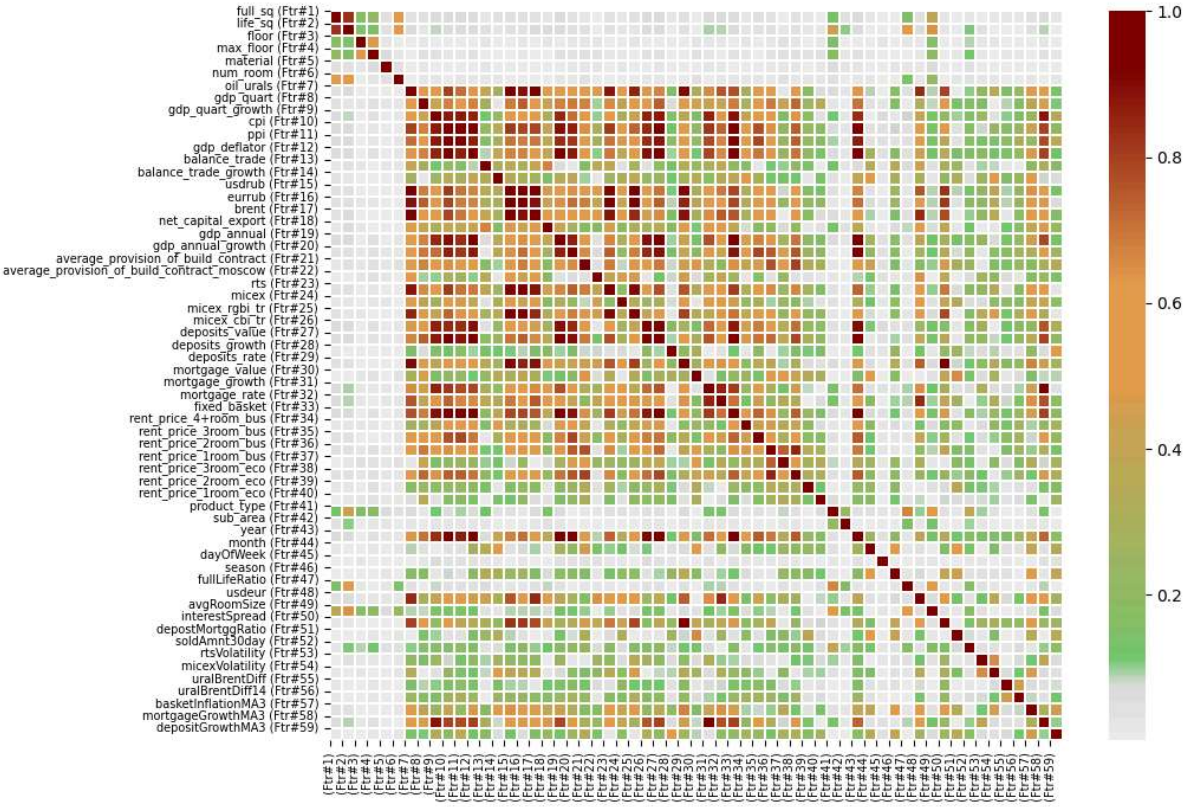
Bagging is a method that builds multiple decision trees without any concern of what other decision trees concluded, each decision tree is trained on a random set of the training data and features, and the result is an aggregation of the results of each tree (in this case an average due to it being a regression problem).

Boosting on the other hand builds multiple decision trees which rely on each-other and try to correct the precious tree's prediction, the result at the end is the result of the last tree which gives out a corrected result of the precious tree.

the main reason to not start before hand with XGBoost is it being very time consuming compared to Random Forest, that because Random Forest can make its decision trees in parallel without having any concern on the results of other trees, its trees do not have an order of some sort, but in XGBoost each tree relys on the previous one and can start only when the previous tree finished.

to start we should view again the correlation matrix to understand the process.

In [ ]:
```
fig, axs = plt.subplots(figsize = (10, 8))
ylabels = [col + f' (Ftr#{i+1})' for i, col in enumerate(corrMatrix.columns)]
xlabels = [f' (Ftr#{i+1})' for i in range(corrMatrix.shape[0])]
sns.heatmap(corrMatrix, cmap = custom_cmap, linewidths = 0.1)
plt.xticks(ticks = range(corrMatrix.shape[1]), labels = xlabels, fontsize = 7)
plt.yticks(ticks = range(corrMatrix.shape[0]), labels = ylabels, fontsize = 7)
plt.show()
```



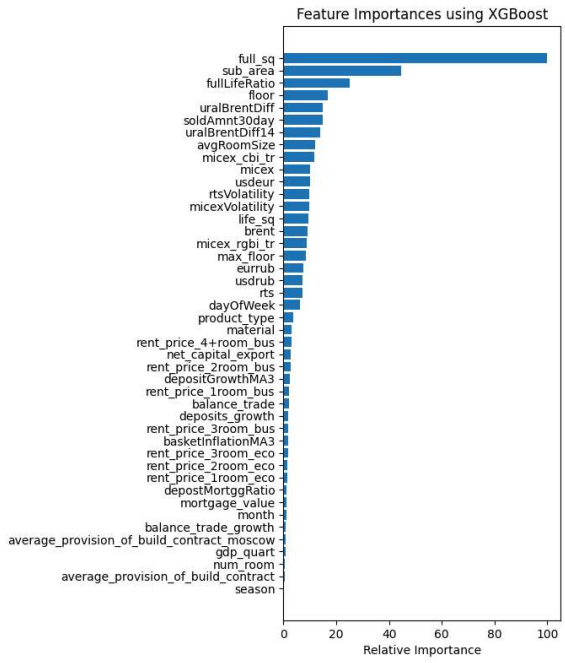
In [ ]:

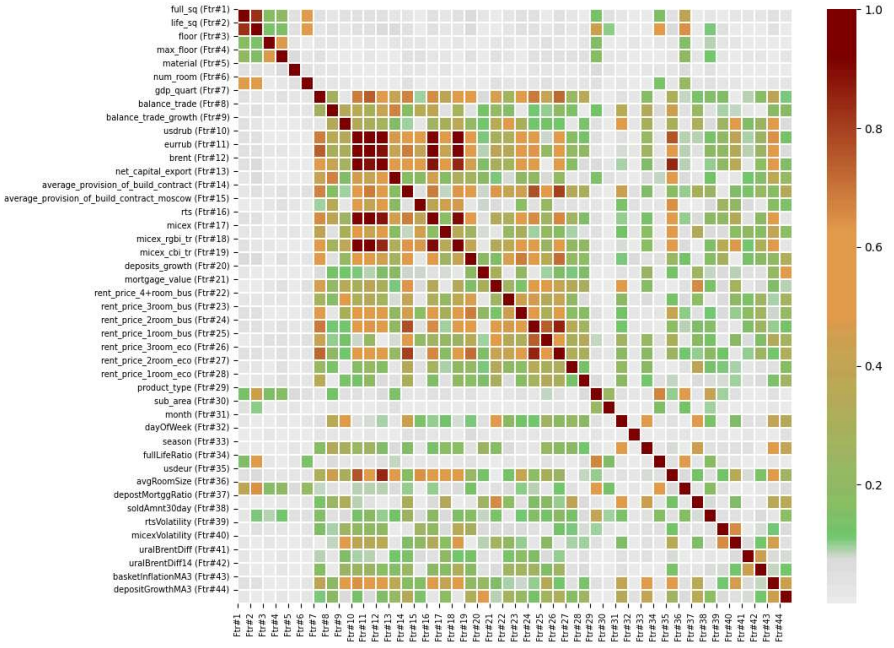
```
XGBoostParams = {
    'learning_rate': 0.014,
    'max_depth': 12,
    'min_samples_leaf': 8,
    'n_estimators': 375,
    'subsample': 0.65
}

importanceXGBModel = GBR(**XGBoostParams)
importanceXGBModel.fit(xTrn, yTrn)
importance = pd.Series(importanceXGBModel.feature_importances_, index = xTrn.columns).sort_values(ascending = True)
importance = importance/importance.max()*100

groups = []
corr_threshold = 0.75
notInGroup = pd.Series([True for i in range(corrMatrix.shape[1])], index = corrMatrix.columns)

for feature in corrMatrix.columns:
    if not groups:
        groups.append([feature])
        notInGroup[feature] = False
    else:
        if notInGroup[feature]:
            currGroup = [feature]
            notInGroup[feature] = False
            for f in corrMatrix.columns:
                if (corrMatrix.loc[f, feature] >= corr_threshold) & (notInGroup[f]):
                    currGroup.append(f)
                    notInGroup[f] = False
            groups.append(currGroup)

xgb_xTrn = xTrn.copy(deep = True)
xgb_xVal = xVal.copy(deep = True)
xgb_imputedEnhncTrain = imputedEnhncTrain.copy(deep = True)
xgb_imputedEnhncTest = imputedEnhncTest.copy(deep = True)
xgb_corrMatrix = corrMatrix.copy(deep = True)

print(f'Feature Filteration Log - Enhanced XGBoost Model\n==================================================\n    threshold = {corr_threshold}\n')

removed = []

for i, group in enumerate(groups):
    if len(group) == 1:
        print(f"Group {i+1}:\n    - the feature ['{group[0]}'] is not correlated enough to any other feature.")
    else:
        avgImportance = importance.loc[group].mean()
        col2Keep = importance.loc[group][importance.loc[group] >= avgImportance].index
        removed += [col for col in group if col not in col2Keep]
        xgb_imputedEnhncTrain = xgb_imputedEnhncTrain[[col for col in xgb_imputedEnhncTrain.columns if col not in [c for c in group if c not in col2Keep]]]
        xgb_imputedEnhncTest = xgb_imputedEnhncTest[[col for col in xgb_imputedEnhncTest.columns if col not in [c for c in group if c not in col2Keep]]]
        xgb_xTrn = xgb_xTrn[[col for col in xgb_xTrn.columns if col not in [c for c in group if c not in col2Keep]]]
        xgb_xVal = xgb_xVal[[col for col in xgb_xVal.columns if col not in [c for c in group if c not in col2Keep]]]
        xgb_corrMatrix = xgb_corrMatrix.loc[[col for col in xgb_corrMatrix.columns if col not in [c for c in group if c not in col2Keep]], [col for col in xgb_corrMatrix.c
        print(f"Group {i+1}:\n    - features kept- {list(col2Keep)}\n    - features removed- {[col for col in group if col not in col2Keep]}")

print(f'removed features:\n    - {removed}')
```

```
Feature Filteration Log - Enhanced XGBoost Model
================================================
    threshold = 0.75

Group 1:
    - the feature ['full_sq'] is not correlated enough to any other feature.
Group 2:
    - the feature ['life_sq'] is not correlated enough to any other feature.
Group 3:
    - the feature ['floor'] is not correlated enough to any other feature.
Group 4:
    - the feature ['max_floor'] is not correlated enough to any other feature.
Group 5:
    - the feature ['material'] is not correlated enough to any other feature.
Group 6:
    - the feature ['num_room'] is not correlated enough to any other feature.
Group 7:
    - features kept- ['usdrub', 'eurrub', 'brent', 'rts', 'micex_rgbi_tr', 'usdeur']
    - features removed- ['oil_urals', 'cpi', 'deposits_rate', 'mortgage_rate', 'fixed_basket', 'interestSpread']
Group 8:
    - the feature ['gdp_quart'] is not correlated enough to any other feature.
Group 9:
    - features kept- ['micex_cbi_tr']
    - features removed- ['gdp_quart_growth', 'ppi', 'gdp_deflator', 'gdp_annual', 'deposits_value', 'mortgage_growth', 'year', 'mortgageGrowthMA3']
Group 10:
    - the feature ['balance_trade'] is not correlated enough to any other feature.
Group 11:
    - the feature ['balance_trade_growth'] is not correlated enough to any other feature.
Group 12:
    - the feature ['net_capital_export'] is not correlated enough to any other feature.
Group 13:
    - features kept- ['rent_price_3room_bus', 'rent_price_2room_bus', 'rent_price_3room_eco']
    - features removed- ['gdp_annual_growth']
Group 14:
    - the feature ['average_provision_of_build_contract'] is not correlated enough to any other feature.
Group 15:
    - the feature ['average_provision_of_build_contract_moscow'] is not correlated enough to any other feature.
Group 16:
    - the feature ['micex'] is not correlated enough to any other feature.
Group 17:
    - the feature ['deposits_growth'] is not correlated enough to any other feature.
Group 18:
    - the feature ['mortgage_value'] is not correlated enough to any other feature.
Group 19:
    - the feature ['rent_price_4+room_bus'] is not correlated enough to any other feature.
Group 20:
    - the feature ['rent_price_1room_bus'] is not correlated enough to any other feature.
Group 21:
    - the feature ['rent_price_2room_eco'] is not correlated enough to any other feature.
Group 22:
    - the feature ['rent_price_1room_eco'] is not correlated enough to any other feature.
Group 23:
    - the feature ['product_type'] is not correlated enough to any other feature.
Group 24:
    - the feature ['sub_area'] is not correlated enough to any other feature.
Group 25:
    - the feature ['month'] is not correlated enough to any other feature.
Group 26:
    - the feature ['dayOfWeek'] is not correlated enough to any other feature.
Group 27:
    - the feature ['season'] is not correlated enough to any other feature.
Group 28:
    - the feature ['fullLifeRatio'] is not correlated enough to any other feature.
Group 29:
    - the feature ['avgRoomSize'] is not correlated enough to any other feature.
Group 30:
    - the feature ['depostMortggRatio'] is not correlated enough to any other feature.
Group 31:
    - the feature ['soldAmnt30day'] is not correlated enough to any other feature.
Group 32:
    - the feature ['rtsVolatility'] is not correlated enough to any other feature.
Group 33:
    - the feature ['micexVolatility'] is not correlated enough to any other feature.
Group 34:
    - the feature ['uralBrentDiff'] is not correlated enough to any other feature.
Group 35:
    - the feature ['uralBrentDiff14'] is not correlated enough to any other feature.
Group 36:
    - the feature ['basketInflationMA3'] is not correlated enough to any other feature.
Group 37:
    - the feature ['depositGrowthMA3'] is not correlated enough to any other feature.
removed features:
    - ['oil_urals', 'cpi', 'deposits_rate', 'mortgage_rate', 'fixed_basket', 'interestSpread', 'gdp_quart_growth', 'ppi', 'gdp_deflator', 'gdp_annual', 'deposits_value',
'mortgage_growth', 'year', 'mortgageGrowthMA3', 'gdp_annual_growth']
```

In [ ]:

```python
importanceXGBModel = GBR(**XGBoostParams)
importanceXGBModel.fit(xgb_xTrn, yTrn)
importance = pd.Series(importanceXGBModel.feature_importances_, index = xgb_xTrn.columns).sort_values(ascending = True)
importance = importance/importance.max()*100

fig = plt.figure(figsize = (18, 8))
gs = GS.GridSpec(1, 2, width_ratios=[2, 5])

ax0 = plt.subplot(gs[0])
ax1 = plt.subplot(gs[1])

ax0.set_title('Feature Importances using XGBoost')
ax0.barh(importance.index, importance.values, align='center')
ax0.set_xlabel('Relative Importance')

corrMatrixTicks = range(xgb_corrMatrix.shape[0])
ylabels = [col + f' (Ftr#{i+1})' for i, col in enumerate(xgb_corrMatrix.columns)]
xlabels = [f'Ftr#{i+1}' for i in corrMatrixTicks]
sns.heatmap(xgb_corrMatrix, cmap = custom_cmap, linewidths = 0.1, ax=ax1)
ax1.set_xticks(ticks = corrMatrixTicks)
ax1.set_xticklabels(labels = xlabels, fontsize = 7)
ax1.set_yticks(ticks = corrMatrixTicks)
ax1.set_yticklabels(labels = ylabels, fontsize = 7)

plt.tight_layout()
plt.show()
```





we can see that much less of the features were removed here, and its clear why, while filtering the features (using XGBoost this time) the model gave for many features higher importance levels than they had before (when using Random Forest).

and it seems that many of the features should be included, we'll keep all 21 of them, from full_sq to dayOfWeek

In [ ]:
```
xgb_features2drop = importance.head(importance.shape[0] - 21).index

xgb_xTrn.drop(columns = xgb_features2drop, inplace = True)
xgb_xVal.drop(columns = xgb_features2drop, inplace = True)
xgb_imputedEnhncTrain.drop(columns = xgb_features2drop, inplace = True)
xgb_imputedEnhncTest.drop(columns = xgb_features2drop, inplace = True)
```

we'll use the hyper-parameters of the enhanced Random Forest model as a baseline to start from, and look around those, values, and have a bit wider range for the hyper-parameters we didnt have previously.

In [ ]:
```
enhncXGBoostParams = {
    'learning_rate': [i/1000 for i in range(11, 18, 2)], # 0.011, 0.013, 0.015, 0.017
    'n_estimators': [i for i in range(350, 401, 25)], # 350, 375, 400
    'subsample': [i/20 for i in range(12, 16)], # 0.6, 0.65, 0.7, 0.75
    'max_depth': [i for i in range(9, 16, 3)], # 9, 12, 15
    'min_samples_leaf': [i for i in range(7, 11)] # 7, 8, 9, 10
}

enhncXGBoostGrdSrch = GSCV(GBR(), enhncXGBoostParams, cv = 5, scoring = 'neg_mean_squared_error', verbose = 2)
enhncXGBoostGrdSrch.fit(xgb_xTrn, yTrn)

enhncXGBoostModelParams = enhncXGBoostGrdSrch.best_params_

enhncXGBoostOptModel = GBR(**enhncXGBoostModelParams)
enhncXGBoostOptModel.fit(xgb_xTrn, yTrn)
```

In [ ]:
```
shortcutParams = {
    'learning_rate': 0.011,
    'n_estimators': 400,
    'subsample': 0.75,
    'max_depth': 9,
    'min_samples_leaf': 10
}

enhncXGBoostOptModel = GBR(**shortcutParams)
enhncXGBoostOptModel.fit(xgb_xTrn, yTrn)
```

Out[ ]:
```
GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.011, max_depth=9, min_samples_leaf=10,
                          n_estimators=400, subsample=0.75)
```

b. Evaluation and Comparison

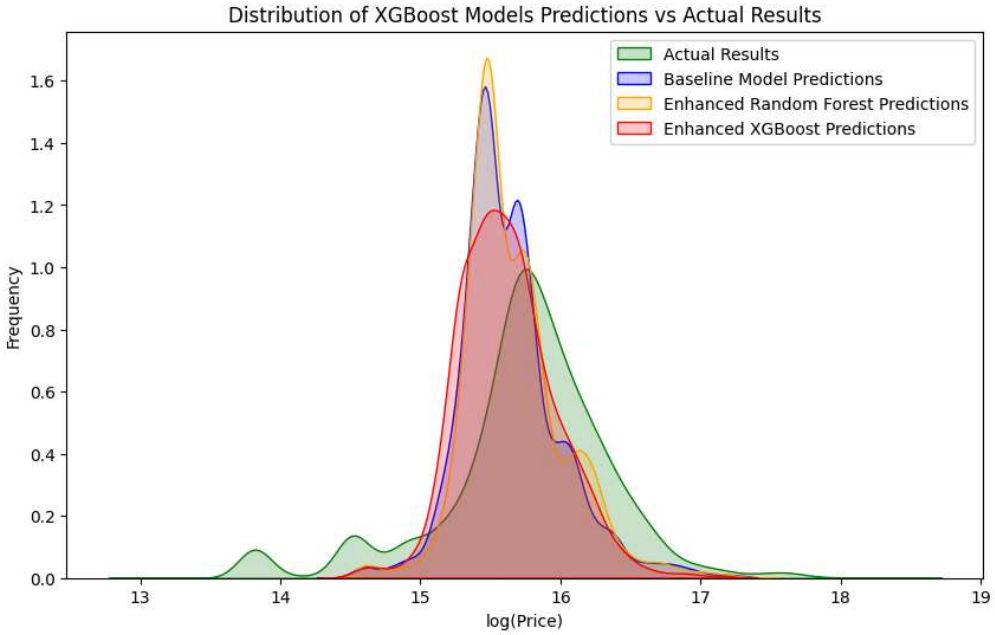
In [ ]:
```
enhncXGBoostMseScores = -1 * CVS(enhncXGBoostOptModel, xgb_xVal, yVal, cv = 5, scoring = 'neg_mean_squared_error')
enhncXGBoostPred = enhncXGBoostOptModel.predict(xgb_xVal)
enhncXGBoostRmsle = np.sqrt(np.mean((enhncXGBoostPred - yVal) ** 2))
enhncXGBoostR2 = enhncXGBoostOptModel.score(xgb_xVal, yVal)

metrics.loc[metrics.shape[0]] = {'Model': 'Enhanced XGBoost Model',
                                 'MSE': enhncXGBoostMseScores.mean(),
                                 'RMSLE': enhncXGBoostRmsle,
                                 'R^2': enhncXGBoostR2}

display(metrics)
```

	Model	MSE	RMSLE	R^2
0	Random Forest Model	0.192769	0.449120	0.409938
1	Enhanced Random Forest Model	0.255073	0.505923	0.326398
2	Enhanced XGBoost Model	0.249734	0.513840	0.305152

comparing to the directly previous model to this one, it seems to improve under the MSE & R^2 metrics but not the RMSLE, but compared to the baseline, it is worse on the MSE and RMSLE, and better on the R^2

we should see visually aswell again.

In [ ]:
```
# xgboost comparison
plt.figure(figsize=(10, 6))
sns.kdeplot(yVal, color = 'green', fill = True, alpha = 0.2, label='Actual Results')
sns.kdeplot(rndFrstPred, color = 'blue', fill = True, alpha = 0.2, label='Baseline Model Predictions')
sns.kdeplot(enhncRndFrstPred, color = 'orange', fill = True, alpha = 0.2, label='Enhanced Random Forest Predictions')
sns.kdeplot(enhncXGBoostPred, color = 'red', fill = True, alpha = 0.2, label='Enhanced XGBoost Predictions')
plt.xlabel('log(Price)')
plt.ylabel('Frequency')
plt.title('Distribution of XGBoost Models Predictions vs Actual Results')
plt.legend()
plt.show()
```



it is noticeble that the newest model so far is the closest in height to the actual results, but it is very right-squed compared to the results, its thickness may explain its downfall in its performace mentioned before

In [ ]:
```
yPredEnhncXGBoost = np.round(np.exp(enhncXGBoostOptModel.predict(xgb_imputedEnhncTest.drop(columns = ['id', 'timestamp']))), decimals = -3)
enhncXGBoostSubmission = pd.DataFrame({'id': xgb_imputedEnhncTest.id, 'price_doc': yPredEnhncXGBoost})
enhncXGBoostSubmission.to_csv('EnhancedXGBoostSubmission.csv', index = False)
```

when submitting the predictions test data we get the following results in comparison to previous models:

	Private Score	Public Score
Random Forest Model	0.38628	0.38735
Enhanced Random Forest Model	0.37493	0.37117
Enhanced XGBoost Model	0.39436	0.39266

as expected to our graphs and metrics, the new XGBoost model did not improve upon the previous models, and even more, it is the worst model yet!

5. State-Of-The-Art Techniques

a. Technique Exploration

We looked at the solution of the 1st place competitors and saw a bunch of techniques they mentioned they did that we did not think about.

1. they've split the datasets to Investment dataset and OwnerOccupier dataset for both the train and the test datasets, modeling for each train and test pair a model, and the final prediction was the results of the model appropriate to the product_type value
2. there was there different data cleaning for each model, some more and some less.
3. they said that the full_sq feature had to much importance to the models, meaning it overshadowed the importance of other features,

   so they replaced it with split features of full_sq dependent on different categorical features, calculated as: $full\_sq_{categoricalFeature} = \frac{full\_sq}{mean_{categoricalFeature}(full\_sq)}$
4. they said that the magic numbers for Investment and OwnerOccupier are different from one another, 1.05 for Investment and 0.9 for OwnerOccupier

b. Application and Performance

1. the 1st technique used shows that this difference in the performance of a 'combined' model compared to one singular model could come from the differences within the housing market, hinting on the fact that the housing market for Investment properties is seemingly similar at first glance to the housing market for OwnerOccupier propeties but actively they are different in their essence and in what they rely on.
2. the 2nd technique could also come from the 1st technique resoning, if the housing market of the 2 types of properties is different in its essence than this also implys that they atleast partially rely on different features, therefor less data cleaning would be required in one model on atleast one of the features compared to the other model.
3. the 3rd technique used solves the contradiction that is expressed within the models we made so far, where we would expect that different categorical values do affect the final results in different ways, and since they found that the full_sq is has to much importance, this could be because the affect of the categorical features on the prediction (atleast some of it) is represented within the full_sq, as full_sq is also affected by these categorical features, which corresponds with the correlation matrix viewed before, some features have correlation with the full_sq feature, and since full_sq seemingly has the most affect on the final result, its correlated part with the other features overtakes the importance of these other features, and lessens their contribution in the final model.
4. the 4th technique is the use of magic numberes the actual price, we could not understand the actual meaning of it.

lets implement these techniques to our models and see the results that would come from doing so.

to do so we need the data from the start, and the data cleaning, up untill the feature engineering.

In [ ]:
```
SbrBkFinalTrain = pd.read_csv('train.csv', parse_dates=['timestamp']).filter(['full_sq', 'life_sq', 'floor', 'max_floor', 'material', 'build_year', 'num_room', 'kitch_sq'
SbrBkFinalTest = pd.read_csv('test.csv', parse_dates=['timestamp']).filter(['id', 'full_sq', 'life_sq', 'floor', 'max_floor', 'material', 'build_year', 'num_room', 'kitch_
SbrBkFinalTrain = pd.merge(SbrBkFinalTrain, pd.read_csv('macro.csv', parse_dates=['timestamp']), how = 'left', on='timestamp')
SbrBkFinalTest = pd.merge(SbrBkFinalTest, pd.read_csv('macro.csv', parse_dates=['timestamp']), how = 'left', on='timestamp')
```

In [ ]:
```
# to include also the rows where product_type is missing
SbrBkInvstTrain = SbrBkFinalTrain[SbrBkFinalTrain.product_type != 'OwnerOccupier']
SbrBkInvstTest = SbrBkFinalTest[SbrBkFinalTest.product_type != 'OwnerOccupier']
SbrBkOwnrTrain = SbrBkFinalTrain[SbrBkFinalTrain.product_type == 'OwnerOccupier']
SbrBkOwnrTest = SbrBkFinalTest[SbrBkFinalTest.product_type == 'OwnerOccupier']
IdTest = SbrBkFinalTest.id
```

In [ ]:
```
resInvst = (SbrBkInvstTrain.isna().sum() / SbrBkInvstTrain.shape[0] * 100).round(2)
resOwnr = (SbrBkOwnrTrain.isna().sum() / SbrBkOwnrTrain.shape[0] * 100).round(2)

res = pd.DataFrame({'Investment': resInvst, 'OwnerOccupier': resOwnr}, index = resInvst.index).sort_values(by = 'Investment', ascending = False)

print('Precentages of missing values in Train:')
display(res[(res.Investment > 37) | (res.OwnerOccupier > 37)])

print(f'Number of rows with missing values in Train:\n  Investment:      {SbrBkInvstTrain.isna().any(axis = 1).sum()}\n  OwnerOccupier:   {SbrBkOwnrTrain.isna().any(axis =
print(f'Precentage of missing values overall in Train:\n  Investment:    {SbrBkInvstTrain.isna().sum().sum() / (SbrBkInvstTrain.shape[0] * SbrBkInvstTrain.shape[1]) * 100:
```

```
Precentages of missing values in Train:
```

	Investment	OwnerOccupier
provision_retail_space_modern_sqm	96.16	99.95
provision_retail_space_sqm	79.55	85.36
museum_visitis_per_100_cap	55.64	55.17
theaters_viewers_per_1000_cap	55.64	55.17
students_reg_sports_share	55.64	55.17
load_of_teachers_preschool_per_teacher	55.64	55.17
state	36.30	58.97
build_year	30.49	69.63
life_sq	0.01	57.90

```
Number of rows with missing values in Train:
  Investment:      19448
  OwnerOccupier:   11023
Precentage of missing values overall in Train:
  Investment:    8.334%
  OwnerOccupier: 10.853%
```

we see here that 3 features have less than 37% of their values missing in Investment but not in OwnerOccupier, this brings in a huge difference to the models features further on

like before the features we'll remove in each dataset and the precentages of missing values in them are:

- Investment
  - $provision\_retail\_space\_modern\_sqm \quad missing = 96.16\%$
  - $provision\_retail\_space\_sqm \quad missing = 79.55\%$
  - $museum\_visitis\_per\_100\_cap \quad missing = 55.64\%$
  - $theaters\_viewers\_per\_1000\_cap \quad missing = 55.64\%$
  - $load\_of\_teachers\_preschool\_per\_teacher \quad missing = 55.64\%$
  - $students\_reg\_sports\_share \quad missing = 55.64\%$
- OwnerOccupier
  - $provision\_retail\_space\_modern\_sqm \quad missing = 99.95\%$

- $provision\_retail\_space\_sqm$ $missing = 85.36\%$
- $museum\_visitis\_per\_100\_cap$ $missing = 55.17\%$
- $theaters\_viewers\_per\_1000\_cap$ $missing = 55.17\%$
- $load\_of\_teachers\_preschool\_per\_teacher$ $missing = 55.17\%$
- $students\_reg\_sports\_share$ $missing = 55.17\%$
- $build\_year$ $missing = 58.97\%$
- $state$ $missing = 69.63\%$
- $life\_sq$ $missing = 57.90\%$

note: here as well we'll remove kitch_sq due to the observation in our baseline model we would wlaso remove product_type, since the data is separated by it.

In [ ]:
```
colsToDropInvst = resInvst.loc[resInvst > 37].index
colsToDropOwnr = resOwnr.loc[resOwnr > 37].index
SbrBkInvstTrain.drop(columns = colsToDropInvst, inplace = True)
SbrBkOwnrTrain.drop(columns = colsToDropOwnr, inplace = True)
SbrBkInvstTest.drop(columns = colsToDropInvst, inplace = True)
SbrBkOwnrTest.drop(columns = colsToDropOwnr, inplace = True)
SbrBkInvstTrain.drop(columns = ['kitch_sq', 'product_type'], inplace = True)
SbrBkOwnrTrain.drop(columns = ['kitch_sq', 'product_type'], inplace = True)
SbrBkInvstTest.drop(columns = ['kitch_sq', 'product_type'], inplace = True)
SbrBkOwnrTest.drop(columns = ['kitch_sq', 'product_type'], inplace = True)
print(f'{SbrBkInvstTrain.shape[1]-1} features are left in the Investment dataset.\n{SbrBkOwnrTrain.shape[1]-1} features are left in the OwnerOccupier dataset.\n')
print(f'Number of rows with missing values in Investment Train: {SbrBkInvstTrain.isna().any(axis = 1).sum()}\nNumber of rows with missing values in OwnerOccupier Train: {S
print(f'Precentage of missing values overall in Investment Train: {SbrBkInvstTrain.isna().sum().sum() / (SbrBkInvstTrain.shape[0] * SbrBkInvstTrain.shape[1]) * 100:.3f}%\r
```

```
103 features are left in the Investment dataset.
100 features are left in the OwnerOccupier dataset.

Number of rows with missing values in Investment Train: 8822
Number of rows with missing values in OwnerOccupier Train: 5210

Precentage of missing values overall in Investment Train: 4.857%
Precentage of missing values overall in OwnerOccupier Train: 5.834%
```

In [ ]:
```
temp_df = SbrBkInvstTrain.copy(deep = True)
temp_df.fillna(-99, inplace = True)

illogicalInvstTrain_df = pd.DataFrame({
    'illogicalFull_sq': (temp_df['full_sq'] < 7) | (temp_df['full_sq'] > 500),
    'illogicalLife_sq': (temp_df['life_sq'] < 7) | (temp_df['life_sq'] > 300),
    'illogicalLifeGreatFull': (temp_df['life_sq'] > temp_df['full_sq']) & (temp_df['full_sq'] != -99) & (temp_df['life_sq'] != -99),
    'illogicalLifeSize': (temp_df['life_sq'] / temp_df['num_room'] < 2) & (temp_df['life_sq'] != -99) & (temp_df['num_room'] != -99),
    'illogicalMax_floor': (temp_df['floor'] > temp_df['max_floor']) & (temp_df['max_floor'] != -99) & (temp_df['floor'] != -99),
    'illogicalNum_room': ((temp_df['num_room'] < 1) | (temp_df['num_room'] > 12)) & (temp_df['num_room'] != -99),
    'illogicalBuild_year': ((temp_df['build_year'] < 1800) | (temp_df['build_year'] > 2022)) & (temp_df['build_year'] != -99),
    'illogicalState': ((temp_df['state'] < 1) | (temp_df['state'] > 4)) & (temp_df['state'] != -99)
})

temp_df = SbrBkOwnrTrain.copy(deep = True)
temp_df.fillna(-99, inplace = True)

illogicalOwnrTrain_df = pd.DataFrame({
    'illogicalFull_sq': (temp_df['full_sq'] < 7) | (temp_df['full_sq'] > 500),
    'illogicalMax_floor': (temp_df['floor'] > temp_df['max_floor']) & (temp_df['max_floor'] != -99) & (temp_df['floor'] != -99),
    'illogicalNum_room': ((temp_df['num_room'] < 1) | (temp_df['num_room'] > 12)) & (temp_df['num_room'] != -99),
})

print('Precentages of illogical values in Investment Train:')
display((illogicalInvstTrain_df.sum() / illogicalInvstTrain_df.shape[0] * 100).sort_values(ascending = False))
print('Precentages of illogical values in OwnerOccupier Train:')
display((illogicalOwnrTrain_df.sum() / illogicalOwnrTrain_df.shape[0] * 100).sort_values(ascending = False))
print(f'Number of rows with illogical values in Train:\n  Investment:    {illogicalInvstTrain_df.any(axis = 1).sum()}\n  OwnerOccupier: {illogicalOwnrTrain_df.any(axis = 1
print(f'Precentage of missing values overall in Train:\n  Investment:    {SbrBkInvstTrain.isna().sum().sum() / (SbrBkInvstTrain.shape[0] * SbrBkInvstTrain.shape[1]) * 100:
```

```
Precentages of illogical values in Investment Train:
illogicalMax_floor        0.287947
illogicalBuild_year       0.205677
illogicalLifeGreatFull    0.092555
illogicalLife_sq          0.082271
illogicalLifeSize         0.035993
illogicalNum_room         0.030852
illogicalFull_sq          0.020568
illogicalState            0.005142
dtype: float64
Precentages of illogical values in OwnerOccupier Train:
illogicalMax_floor    13.036378
illogicalFull_sq       0.272158
illogicalNum_room      0.090719
dtype: float64
Number of rows with illogical values in Train:
  Investment:    133
  OwnerOccupier: 1466

Precentage of missing values overall in Train:
  Investment:    4.857%
  OwnerOccupier: 5.834%
```

it seems now that only 1599 rows have any illogical values in them, which is better for us

lets do the same with the test datasets!

In [ ]:
```
wrn.filterwarnings('ignore')

temp_df = SbrBkInvstTest.copy(deep = True)
temp_df.fillna(-99, inplace = True)

illogicalInvstTest_df = pd.DataFrame({
    'illogicalFull_sq': (temp_df['full_sq'] < 7) | (temp_df['full_sq'] > 500),
    'illogicalLife_sq': (temp_df['life_sq'] < 7) | (temp_df['life_sq'] > 300),
    'illogicalLifeGreatFull': (temp_df['life_sq'] > temp_df['full_sq']) & (temp_df['full_sq'] != -99) & (temp_df['life_sq'] != -99),
    'illogicalLifeSize': (temp_df['life_sq'] / temp_df['num_room'] < 2) & (temp_df['life_sq'] != -99) & (temp_df['num_room'] != -99),
    'illogicalMax_floor': (temp_df['floor'] > temp_df['max_floor']) & (temp_df['max_floor'] != -99) & (temp_df['floor'] != -99),
    'illogicalNum_room': ((temp_df['num_room'] < 1) | (temp_df['num_room'] > 12)) & (temp_df['num_room'] != -99),
    'illogicalBuild_year': ((temp_df['build_year'] < 1800) | (temp_df['build_year'] > 2022)) & (temp_df['build_year'] != -99),
    'illogicalState': ((temp_df['state'] < 1) | (temp_df['state'] > 4)) & (temp_df['state'] != -99)
})

temp_df = SbrBkOwnrTest.copy(deep = True)
temp_df.fillna(-99, inplace = True)
```

```
illogicalOwnrTest_df = pd.DataFrame({
    'illogicalFull_sq': (temp_df['full_sq'] < 7) | (temp_df['full_sq'] > 500),
    'illogicalMax_floor': (temp_df['floor'] > temp_df['max_floor']) & (temp_df['max_floor'] != -99) & (temp_df['floor'] != -99),
    'illogicalNum_room': ((temp_df['num_room'] < 1) | (temp_df['num_room'] > 12)) & (temp_df['num_room'] != -99),
})

print('Precentages of illogical values in Investment Train:')
display((illogicalInvstTest_df.sum() / illogicalInvstTest_df.shape[0] * 100).sort_values(ascending = False))
print('Precentages of illogical values in OwnerOccupier Train:')
display((illogicalOwnrTest_df.sum() / illogicalOwnrTest_df.shape[0] * 100).sort_values(ascending = False))
print(f'Number of rows with illogical values in Train:\n  Investment:    {illogicalInvstTest_df.any(axis = 1).sum()}\n  OwnerOccupier: {illogicalOwnrTest_df.any(axis = 1)
print(f'Precentage of missing values overall in Train:\n  Investment:    {SbrBkInvstTest.isna().sum().sum() / (SbrBkInvstTest.shape[0] * SbrBkInvstTest.shape[1]) * 100:.3
```

```
Precentages of illogical values in Investment Train:
illogicalMax_floor        0.496919
illogicalBuild_year       0.218644
illogicalLife_sq          0.099384
illogicalLifeGreatFull    0.059630
illogicalLifeSize         0.059630
illogicalNum_room         0.019877
illogicalFull_sq          0.000000
illogicalState            0.000000
dtype: float64
Precentages of illogical values in OwnerOccupier Train:
illogicalMax_floor    23.489168
illogicalFull_sq       0.114025
illogicalNum_room      0.000000
dtype: float64
Number of rows with illogical values in Train:
  Investment:    38
  OwnerOccupier: 621

Precentage of missing values overall in Train:
  Investment:    46.467%
  OwnerOccupier: 44.059%
Precentage of missing values overall in Train:
  Investment:    46.467%
  OwnerOccupier: 44.059%
```

In [ ]:
```
# handling illogical values - train dataset
SbrBkInvstTrain = SbrBkInvstTrain.loc[illogicalInvstTrain_df.any(axis = 1) != True,:]
SbrBkOwnrTrain = SbrBkOwnrTrain.loc[illogicalOwnrTrain_df.any(axis = 1) != True,:]
resInvst = (SbrBkInvstTrain.isna().sum() / SbrBkInvstTrain.shape[0] * 100).round(2).sort_values(ascending = False)
resOwnr = (SbrBkOwnrTrain.isna().sum() / SbrBkOwnrTrain.shape[0] * 100).round(2).sort_values(ascending = False)

display(res[(res.Investment > 37) | (res.OwnerOccupier > 37)])
print(f'Number of rows with missing values in Train:\n  Investment:      {SbrBkInvstTrain.isna().any(axis = 1).sum()}\n  OwnerOccupier:   {SbrBkOwnrTrain.isna().any(axis =
print(f'Precentage of missing values overall in Train:\n  Investment:    {SbrBkInvstTrain.isna().sum().sum() / (SbrBkInvstTrain.shape[0] * SbrBkInvstTrain.shape[1]) * 100

# handling illogical values - test dataset
SbrBkInvstTest.full_sq.loc[illogicalInvstTest_df['illogicalFull_sq']] = np.nan
SbrBkInvstTest.num_room.loc[illogicalInvstTest_df['illogicalNum_room']] = np.nan
SbrBkInvstTest.max_floor.loc[illogicalInvstTest_df['illogicalMax_floor']] = np.nan
SbrBkInvstTest.life_sq.loc[(illogicalInvstTest_df['illogicalLife_sq']) | ((illogicalInvstTest_df['illogicalLifeGreatFull']) & (~illogicalInvstTest_df['illogicalFull_sq']))
SbrBkInvstTest.build_year.loc[illogicalInvstTest_df['illogicalBuild_year']] = np.nan
SbrBkInvstTest.state.loc[illogicalInvstTest_df['illogicalState']] = np.nan

SbrBkOwnrTest.full_sq.loc[illogicalOwnrTest_df['illogicalFull_sq']] = np.nan
SbrBkOwnrTest.num_room.loc[illogicalOwnrTest_df['illogicalNum_room']] = np.nan
SbrBkOwnrTest.max_floor.loc[illogicalOwnrTest_df['illogicalMax_floor']] = np.nan
```

	Investment	OwnerOccupier
provision_retail_space_modern_sqm	96.16	99.95
provision_retail_space_sqm	79.55	85.36
museum_visitis_per_100_cap	55.64	55.17
theaters_viewers_per_1000_cap	55.64	55.17
students_reg_sports_share	55.64	55.17
load_of_teachers_preschool_per_teacher	55.64	55.17
state	36.30	58.97
build_year	30.49	69.63
life_sq	0.01	57.90

```
Number of rows with missing values in Train:
  Investment:      8787
  OwnerOccupier:   4956
Precentage of missing values overall in Train:
  Investment:    4.861%
  OwnerOccupier: 5.841%
```

like before we'll remove the categorical features we have yet to know

In [ ]:
```
SbrBkInvstTrain.drop(columns = ['child_on_acc_pre_school', 'modern_education_share', 'old_education_build_share'], inplace = True)
SbrBkOwnrTrain.drop(columns = ['child_on_acc_pre_school', 'modern_education_share', 'old_education_build_share'], inplace = True)
SbrBkInvstTest.drop(columns = ['child_on_acc_pre_school', 'modern_education_share', 'old_education_build_share'], inplace = True)
SbrBkOwnrTest.drop(columns = ['child_on_acc_pre_school', 'modern_education_share', 'old_education_build_share'], inplace = True)
```

now lets handle the features and their missing values in the test dataset

In [ ]:
```
resInvst = (SbrBkInvstTest.isna().sum() / SbrBkInvstTest.shape[0] * 100).round(2)
resOwnr = (SbrBkOwnrTest.isna().sum() / SbrBkOwnrTest.shape[0] * 100).round(2)

res = pd.DataFrame({'Investment': resInvst, 'OwnerOccupier': resOwnr}, index = resInvst.index).sort_values(by = 'Investment', ascending = False)

print('Precentages of missing values in Train:')
display(res[(res.Investment > 37) | (res.OwnerOccupier > 37)])

print(f'Number of rows with missing values in Train:\n  Investment:      {SbrBkInvstTest.isna().any(axis = 1).sum()}\n  OwnerOccupier:   {SbrBkOwnrTest.isna().any(axis =
print(f'Precentage of missing values overall in Train:\n  Investment:    {SbrBkInvstTest.isna().sum().sum() / (SbrBkInvstTest.shape[0] * SbrBkInvstTest.shape[1]) * 100:.3
```

```
Precentages of missing values in Train:
```

	Investment	OwnerOccupier
sewerage_share	100.00	100.0
baths_share	100.00	100.0
old_house_share	100.00	100.0
heating_share	100.00	100.0
electric_stove_share	100.00	100.0
hot_water_share	100.00	100.0
gas_share	100.00	100.0
grp_growth	100.00	100.0
water_pipes_share	100.00	100.0
invest_fixed_assets_phys	100.00	100.0
lodging_sqm_per_cap	100.00	100.0
housing_fund_sqm	100.00	100.0
grp	100.00	100.0
divorce_rate	100.00	100.0
pop_total_inc	100.00	100.0
pop_migration	100.00	100.0
marriages_per_1000_cap	100.00	100.0
infant_mortarity_per_1000_cap	100.00	100.0
perinatal_mort_per_1000_cap	100.00	100.0
incidence_population	100.00	100.0
real_dispos_income_per_cap_growth	100.00	100.0
fin_res_per_cap	100.00	100.0
provision_doctors	100.00	100.0
overdue_wages_per_cap	100.00	100.0
share_own_revenues	100.00	100.0
power_clinics	100.00	100.0
hospital_beds_available_per_cap	100.00	100.0
hospital_bed_occupancy_per_year	100.00	100.0
unprofitable_enterpr_share	100.00	100.0
profitable_enterpr_share	100.00	100.0
population_reg_sports_share	100.00	100.0
apartment_build	100.00	100.0
construction_value	100.00	100.0
invest_fixed_assets	57.21	42.0
invest_fixed_capital_per_cap	57.21	42.0
pop_natural_increase	57.21	42.0
childbirth	57.21	42.0
mortality	57.21	42.0
unemployment	57.21	42.0
average_life_exp	57.21	42.0
load_of_teachers_school_per_teacher	57.21	42.0
students_state_oneshift	57.21	42.0
provision_nurse	57.21	42.0
load_on_doctors	57.21	42.0
turnover_catering_per_cap	57.21	42.0
seats_theather_rfmin_per_100000_cap	57.21	42.0
bandwidth_sports	57.21	42.0
employment	57.21	42.0
apartment_fund_sqm	57.21	42.0
labor_force	57.21	42.0
salary	57.21	42.0
retail_trade_turnover_growth	57.21	42.0
retail_trade_turnover_per_cap	57.21	42.0
retail_trade_turnover	57.21	42.0
income_per_cap	57.21	42.0
salary_growth	57.21	42.0

```
Number of rows with missing values in Train:
  Investment:      5031
  OwnerOccupier:   2631
Precentage of missing values overall in Train:
  Investment:    45.733%
  OwnerOccupier: 43.771%
```

like before the same features are problematic in almost the same way, we'll remove them to

In [ ]:
```
colsToDropInvst = resInvst.loc[resInvst > 37].index
colsToDropOwnr = resOwnr.loc[resOwnr > 37].index
SbrBkInvstTrain.drop(columns = colsToDropInvst, inplace = True)
SbrBkOwnrTrain.drop(columns = colsToDropOwnr, inplace = True)
SbrBkInvstTest.drop(columns = colsToDropInvst, inplace = True)
SbrBkOwnrTest.drop(columns = colsToDropOwnr, inplace = True)
print(f'{SbrBkInvstTrain.shape[1]-1} features are left in the Investment dataset.\n{SbrBkOwnrTrain.shape[1]-1} features are left in the OwnerOccupier dataset.\n')
print(f'Number of rows with missing values in Investment Train: {SbrBkInvstTest.isna().any(axis = 1).sum()}\nNumber of rows with missing values in OwnerOccupier Train: {Sb
print(f'Precentage of missing values overall in Investment Train: {SbrBkInvstTest.isna().sum().sum() / (SbrBkInvstTest.shape[0] * SbrBkInvstTest.shape[1]) * 100:.3f}%\nPre

print(f'\nInvestment Columns: \n{SbrBkInvstTrain.columns}')
print(f'\n OwnerOccupier Columns: {SbrBkOwnrTrain.columns}')
```

```
44 features are left in the Investment dataset.
41 features are left in the OwnerOccupier dataset.

Number of rows with missing values in Investment Train: 153
Number of rows with missing values in OwnerOccupier Train: 621

Precentage of missing values overall in Investment Train: 0.073%
Precentage of missing values overall in OwnerOccupier Train: 0.562%

Investment Columns:
Index(['full_sq', 'life_sq', 'floor', 'max_floor', 'material', 'build_year',
       'num_room', 'state', 'sub_area', 'price_doc', 'timestamp', 'oil_urals',
       'gdp_quart', 'gdp_quart_growth', 'cpi', 'ppi', 'gdp_deflator',
       'balance_trade', 'balance_trade_growth', 'usdrub', 'eurrub', 'brent',
       'net_capital_export', 'gdp_annual', 'gdp_annual_growth',
       'average_provision_of_build_contract',
       'average_provision_of_build_contract_moscow', 'rts', 'micex',
       'micex_rgbi_tr', 'micex_cbi_tr', 'deposits_value', 'deposits_growth',
       'deposits_rate', 'mortgage_value', 'mortgage_growth', 'mortgage_rate',
       'fixed_basket', 'rent_price_4+room_bus', 'rent_price_3room_bus',
       'rent_price_2room_bus', 'rent_price_1room_bus', 'rent_price_3room_eco',
       'rent_price_2room_eco', 'rent_price_1room_eco'],
      dtype='object')

 OwnerOccupier Columns: Index(['full_sq', 'floor', 'max_floor', 'material', 'num_room', 'sub_area',
       'price_doc', 'timestamp', 'oil_urals', 'gdp_quart', 'gdp_quart_growth',
       'cpi', 'ppi', 'gdp_deflator', 'balance_trade', 'balance_trade_growth',
       'usdrub', 'eurrub', 'brent', 'net_capital_export', 'gdp_annual',
       'gdp_annual_growth', 'average_provision_of_build_contract',
       'average_provision_of_build_contract_moscow', 'rts', 'micex',
       'micex_rgbi_tr', 'micex_cbi_tr', 'deposits_value', 'deposits_growth',
       'deposits_rate', 'mortgage_value', 'mortgage_growth', 'mortgage_rate',
       'fixed_basket', 'rent_price_4+room_bus', 'rent_price_3room_bus',
       'rent_price_2room_bus', 'rent_price_1room_bus', 'rent_price_3room_eco',
       'rent_price_2room_eco', 'rent_price_1room_eco'],
      dtype='object')
```

let impute them and label incode them

In [ ]:
```
wrn.filterwarnings('ignore')

SbrBkInvstTrain['sub_areaCode'] = pd.Categorical(SbrBkInvstTrain.sub_area).codes
SbrBkOwnrTrain['sub_areaCode'] = pd.Categorical(SbrBkOwnrTrain.sub_area).codes
SbrBkInvstTest['sub_areaCode'] = pd.Categorical(SbrBkInvstTest.sub_area).codes
SbrBkOwnrTest['sub_areaCode'] = pd.Categorical(SbrBkOwnrTest.sub_area).codes

tempInvstTrain = SbrBkInvstTrain.copy(deep = True).reset_index(drop = True)
tempOwnrTrain = SbrBkOwnrTrain.copy(deep = True).reset_index(drop = True)
tempInvstTest = SbrBkInvstTest.copy(deep = True).reset_index(drop = True)
tempOwnrTest = SbrBkOwnrTest.copy(deep = True).reset_index(drop = True)
tempInvstTrain.drop(columns = ['sub_area', 'price_doc', 'timestamp'], inplace = True)
tempOwnrTrain.drop(columns = ['sub_area', 'price_doc', 'timestamp'], inplace = True)
tempInvstTest.drop(columns = ['sub_area', 'id', 'timestamp'], inplace = True)
tempOwnrTest.drop(columns = ['sub_area', 'id', 'timestamp'], inplace = True)
colsInvst = list(tempInvstTrain.columns)
colsOwnr = list(tempOwnrTrain.columns)

colsToRoundInvstTrain = tempInvstTrain.columns[tempInvstTrain.isna().any(axis = 0)]
colsToRoundOwnrTrain = tempOwnrTrain.columns[tempOwnrTrain.isna().any(axis = 0)]
colsToRoundInvstTest = tempInvstTest.columns[tempInvstTest.isna().any(axis = 0)]
colsToRoundOwnrTest = tempOwnrTest.columns[tempOwnrTest.isna().any(axis = 0)]

imputedInvstTrainData = KNNImputer(n_neighbors = 20).fit_transform(tempInvstTrain)
imputedOwnrTrainData = KNNImputer(n_neighbors = 20).fit_transform(tempOwnrTrain)
imputedInvstTestData = KNNImputer(n_neighbors = 20).fit_transform(tempInvstTest)
imputedOwnrTestData = KNNImputer(n_neighbors = 20).fit_transform(tempOwnrTest)
imputedInvstTrain = pd.DataFrame(imputedInvstTrainData, columns = colsInvst)
imputedOwnrTrain = pd.DataFrame(imputedOwnrTrainData, columns = colsOwnr)
imputedInvstTest = pd.DataFrame(imputedInvstTestData, columns = colsInvst)
imputedOwnrTest = pd.DataFrame(imputedOwnrTestData, columns = colsOwnr)
imputedInvstTrain.drop(columns = ['sub_areaCode'], inplace = True)
imputedOwnrTrain.drop(columns = ['sub_areaCode'], inplace = True)
imputedInvstTrain[colsToRoundInvstTrain] = imputedInvstTrain[colsToRoundInvstTrain].round()
imputedOwnrTrain[colsToRoundOwnrTrain] = imputedOwnrTrain[colsToRoundOwnrTrain].round()
imputedInvstTest[colsToRoundInvstTest] = imputedInvstTest[colsToRoundInvstTest].round()
imputedOwnrTest[colsToRoundOwnrTest] = imputedOwnrTest[colsToRoundOwnrTest].round()
imputedInvstTrain = imputedInvstTrain.set_index(SbrBkInvstTrain.index)
imputedOwnrTrain = imputedOwnrTrain.set_index(SbrBkOwnrTrain.index)
imputedInvstTest = imputedInvstTest.set_index(SbrBkInvstTest.index)
imputedOwnrTest = imputedOwnrTest.set_index(SbrBkOwnrTest.index)
imputedInvstTrain['sub_area'] = SbrBkInvstTrain.sub_area
imputedOwnrTrain['sub_area'] = SbrBkOwnrTrain.sub_area
imputedInvstTest['sub_area'] = SbrBkInvstTest.sub_area
imputedOwnrTest['sub_area'] = SbrBkOwnrTest.sub_area
imputedInvstTrain['price_doc'] = SbrBkInvstTrain.price_doc
imputedOwnrTrain['price_doc'] = SbrBkOwnrTrain.price_doc
imputedInvstTest['id'] = SbrBkInvstTest.id
imputedOwnrTest['id'] = SbrBkOwnrTest.id
imputedInvstTrain['timestamp'] = SbrBkInvstTrain.timestamp
imputedOwnrTrain['timestamp'] = SbrBkOwnrTrain.timestamp
imputedInvstTest['timestamp'] = SbrBkInvstTest.timestamp
imputedOwnrTest['timestamp'] = SbrBkOwnrTest.timestamp
imputedInvstTest.drop(columns = ['sub_areaCode'], inplace = True)
imputedOwnrTest.drop(columns = ['sub_areaCode'], inplace = True)
imputedInvstTrain['logPrice'] = np.log(imputedInvstTrain.price_doc)
imputedOwnrTrain['logPrice'] = np.log(imputedOwnrTrain.price_doc)
```

In [ ]:
```
imputedInvstTrain.isna().sum().sum() + imputedInvstTest.isna().sum().sum() + imputedOwnrTrain.isna().sum().sum() + imputedOwnrTest.isna().sum().sum()
```

Out[ ]:
```
0
```

and finally label encode the categorical features

In [ ]:
```
imputedInvstTrain['originalIndex'] = imputedInvstTrain.index*10+1
imputedOwnrTrain['originalIndex'] = imputedOwnrTrain.index*10+1
imputedInvstTest['originalIndex'] = imputedInvstTest.index*10
imputedOwnrTest['originalIndex'] = imputedOwnrTest.index*10
imputedInvstUnion = pd.concat([imputedInvstTrain, imputedInvstTest], ignore_index=True)
imputedOwnrUnion = pd.concat([imputedOwnrTrain, imputedOwnrTest], ignore_index=True)
subAreaInvstLbl = LabelEncoder().fit(imputedInvstUnion.sub_area)
subAreaOwnrLbl = LabelEncoder().fit(imputedOwnrUnion.sub_area)
imputedInvstTrain.sub_area = subAreaInvstLbl.transform(imputedInvstTrain.sub_area)
imputedOwnrTrain.sub_area = subAreaOwnrLbl.transform(imputedOwnrTrain.sub_area)
imputedInvstTest.sub_area = subAreaInvstLbl.transform(imputedInvstTest.sub_area)
imputedOwnrTest.sub_area = subAreaOwnrLbl.transform(imputedOwnrTest.sub_area)
```

in the part of the feature engineering we'll make the same features as before (as long it is still possible), and add the suggested features of the winning competitors

In [ ]:
```
imputedInvstTrain = imputedInvstTrain.assign(year = imputedInvstTrain.timestamp.dt.year,
                                             month = imputedInvstTrain.timestamp.dt.month,
                                             dayOfWeek = imputedInvstTrain.timestamp.dt.dayofweek,
                                             season = imputedInvstTrain.timestamp.dt.month % 12 // 3 + 1, # 1 = Winter, 2 = Spring, 3 = Summer, 4 = Autumn
                                             fullLifeRatio = imputedInvstTrain.life_sq / imputedInvstTrain.full_sq,
                                             usdeur = imputedInvstTrain.usdrub / imputedInvstTrain.eurrub,
                                             avgRoomSize = imputedInvstTrain.life_sq / imputedInvstTrain.num_room,
                                             interestSpread = imputedInvstTrain.mortgage_rate - imputedInvstTrain.deposits_rate,
                                             depostMortggRatio = imputedInvstTrain.deposits_value / imputedInvstTrain.mortgage_value)
imputedOwnrTrain = imputedOwnrTrain.assign(year = imputedOwnrTrain.timestamp.dt.year,
                                           month = imputedOwnrTrain.timestamp.dt.month,
                                           dayOfWeek = imputedOwnrTrain.timestamp.dt.dayofweek,
                                           season = imputedOwnrTrain.timestamp.dt.month % 12 // 3 + 1, # 1 = Winter, 2 = Spring, 3 = Summer, 4 = Autumn
                                           usdeur = imputedOwnrTrain.usdrub / imputedOwnrTrain.eurrub,
                                           interestSpread = imputedOwnrTrain.mortgage_rate - imputedOwnrTrain.deposits_rate,
                                           depostMortggRatio = imputedOwnrTrain.deposits_value / imputedOwnrTrain.mortgage_value)
imputedInvstTest = imputedInvstTest.assign(year = imputedInvstTest.timestamp.dt.year,
                                           month = imputedInvstTest.timestamp.dt.month,
                                           dayOfWeek = imputedInvstTest.timestamp.dt.dayofweek,
                                           season = imputedInvstTest.timestamp.dt.month % 12 // 3 + 1, # 1 = Winter, 2 = Spring, 3 = Summer, 4 = Autumn
                                           fullLifeRatio = imputedInvstTest.life_sq / imputedInvstTest.full_sq,
                                           usdeur = imputedInvstTest.usdrub / imputedInvstTest.eurrub,
                                           avgRoomSize = imputedInvstTest.life_sq / imputedInvstTest.num_room,
                                           interestSpread = imputedInvstTest.mortgage_rate - imputedInvstTest.deposits_rate,
                                           depostMortggRatio = imputedInvstTest.deposits_value / imputedInvstTest.mortgage_value)
imputedOwnrTest = imputedOwnrTest.assign(year = imputedOwnrTest.timestamp.dt.year,
                                         month = imputedOwnrTest.timestamp.dt.month,
                                         dayOfWeek = imputedOwnrTest.timestamp.dt.dayofweek,
                                         season = imputedOwnrTest.timestamp.dt.month % 12 // 3 + 1, # 1 = Winter, 2 = Spring, 3 = Summer, 4 = Autumn
                                         usdeur = imputedOwnrTest.usdrub / imputedOwnrTest.eurrub,
                                         interestSpread = imputedOwnrTest.mortgage_rate - imputedOwnrTest.deposits_rate,
                                         depostMortggRatio = imputedOwnrTest.deposits_value / imputedOwnrTest.mortgage_value)

imputedInvstUnion["numOday"] = (imputedInvstUnion.timestamp - imputedInvstUnion.timestamp.min()).dt.days
imputedOwnrUnion["numOday"] = (imputedOwnrUnion.timestamp - imputedOwnrUnion.timestamp.min()).dt.days
imputedInvstUnion["soldAmnt30day"] = imputedInvstUnion[["numOday", "sub_area"]].apply(lambda x: ((x.numOday - imputedInvstUnion.numOday < 31) & (x.numOday - imputedInvstUn
imputedOwnrUnion["soldAmnt30day"] = imputedOwnrUnion[["numOday", "sub_area"]].apply(lambda x: ((x.numOday - imputedOwnrUnion.numOday < 31) & (x.numOday - imputedOwnrUnion.
imputedInvstTrain = pd.merge(imputedInvstTrain, imputedInvstUnion[['originalIndex', 'soldAmnt30day']], how = 'left', on = 'originalIndex')
imputedOwnrTrain = pd.merge(imputedOwnrTrain, imputedOwnrUnion[['originalIndex', 'soldAmnt30day']], how = 'left', on = 'originalIndex')
imputedInvstTest = pd.merge(imputedInvstTest, imputedInvstUnion[['originalIndex', 'soldAmnt30day']], how = 'left', on = 'originalIndex')
imputedOwnrTest = pd.merge(imputedOwnrTest, imputedOwnrUnion[['originalIndex', 'soldAmnt30day']], how = 'left', on = 'originalIndex')
imputedInvstTrain.drop(columns = ['originalIndex'], inplace = True)
imputedOwnrTrain.drop(columns = ['originalIndex'], inplace = True)
imputedInvstTest.drop(columns = ['originalIndex'], inplace = True)
imputedOwnrTest.drop(columns = ['originalIndex'], inplace = True)

macroDf = pd.read_csv('macro.csv', parse_dates=['timestamp'])

macroDf["rtsVolatility"] = macroDf.rts.rolling(window = 90).std()
macroDf["micexVolatility"] = macroDf.micex.rolling(window = 90).std()
macroDf["uralBrentDiff"] = macroDf.brent - macroDf.oil_urals
macroDf["uralBrentDiff14"] = macroDf.uralBrentDiff.rolling(window = 14, min_periods = 1).mean()
macroDf["yearNmonth"] = macroDf.timestamp.dt.year * 100 + macroDf.timestamp.dt.month

monthMacroDf = macroDf.drop_duplicates(subset = ['yearNmonth'])
monthMacroDf["basketInflation"] = monthMacroDf.fixed_basket / monthMacroDf.fixed_basket.shift(1) - 1
monthMacroDf["basketInflationMA3"] = monthMacroDf.basketInflation.rolling(window = 3, min_periods = 1).mean()
monthMacroDf["mortgageGrowthMA3"] = monthMacroDf.mortgage_growth.rolling(window = 3, min_periods = 1).mean()
monthMacroDf["depositGrowthMA3"] = monthMacroDf.deposits_growth.rolling(window = 3, min_periods = 1).mean()

macroDf = pd.merge(macroDf, monthMacroDf[['yearNmonth', 'basketInflationMA3', 'mortgageGrowthMA3', 'depositGrowthMA3']], how = 'left', on = 'yearNmonth')

imputedInvstTrain = pd.merge(imputedInvstTrain, macroDf[['timestamp', 'rtsVolatility', 'micexVolatility', 'uralBrentDiff', 'uralBrentDiff14', 'basketInflationMA3', 'mortga
imputedOwnrTrain = pd.merge(imputedOwnrTrain, macroDf[['timestamp', 'rtsVolatility', 'micexVolatility', 'uralBrentDiff', 'uralBrentDiff14', 'basketInflationMA3', 'mortgage
imputedInvstTest = pd.merge(imputedInvstTest, macroDf[['timestamp', 'rtsVolatility', 'micexVolatility', 'uralBrentDiff', 'uralBrentDiff14', 'basketInflationMA3', 'mortgage
imputedOwnrTest = pd.merge(imputedOwnrTest, macroDf[['timestamp', 'rtsVolatility', 'micexVolatility', 'uralBrentDiff', 'uralBrentDiff14', 'basketInflationMA3', 'mortgageGr

# full_sq by material
invstTrainGroupMonthYearMaterial = imputedInvstTrain.groupby(['year', 'month', 'material']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqMat
imputedInvstTrain = pd.merge(imputedInvstTrain, invstTrainGroupMonthYearMaterial, how = 'left', on = ['year', 'month', 'material'])
imputedInvstTrain['full_sq_material'] = imputedInvstTrain.full_sq / imputedInvstTrain.full_sqMaterial
ownrTrainGroupMonthYearMaterial = imputedOwnrTrain.groupby(['year', 'month', 'material']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqMater
imputedOwnrTrain = pd.merge(imputedOwnrTrain, ownrTrainGroupMonthYearMaterial, how = 'left', on = ['year', 'month', 'material'])
imputedOwnrTrain['full_sq_material'] = imputedOwnrTrain.full_sq / imputedOwnrTrain.full_sqMaterial
invstTestGroupMonthYearMaterial = imputedInvstTest.groupby(['year', 'month', 'material']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqMater
imputedInvstTest = pd.merge(imputedInvstTest, invstTestGroupMonthYearMaterial, how = 'left', on = ['year', 'month', 'material'])
imputedInvstTest['full_sq_material'] = imputedInvstTest.full_sq / imputedInvstTest.full_sqMaterial
ownrTestGroupMonthYearMaterial = imputedOwnrTest.groupby(['year', 'month', 'material']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqMateria
imputedOwnrTest = pd.merge(imputedOwnrTest, ownrTestGroupMonthYearMaterial, how = 'left', on = ['year', 'month', 'material'])
imputedOwnrTest['full_sq_material'] = imputedOwnrTest.full_sq / imputedOwnrTest.full_sqMaterial

# full_sq by num_room
invstTrainGroupMonthYearNroom = imputedInvstTrain.groupby(['year', 'month', 'num_room']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqNroom
imputedInvstTrain = pd.merge(imputedInvstTrain, invstTrainGroupMonthYearNroom, how = 'left', on = ['year', 'month', 'num_room'])
imputedInvstTrain['full_sq_num_room'] = imputedInvstTrain.full_sq / imputedInvstTrain.full_sqNroom
ownrTrainGroupMonthYearNroom = imputedOwnrTrain.groupby(['year', 'month', 'num_room']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqNroom'})
imputedOwnrTrain = pd.merge(imputedOwnrTrain, ownrTrainGroupMonthYearNroom, how = 'left', on = ['year', 'month', 'num_room'])
imputedOwnrTrain['full_sq_num_room'] = imputedOwnrTrain.full_sq / imputedOwnrTrain.full_sqNroom
invstTestGroupMonthYearNroom = imputedInvstTest.groupby(['year', 'month', 'num_room']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqNroom'})
```

```
imputedInvstTest = pd.merge(imputedInvstTest, invstTestGroupMonthYearNroom, how = 'left', on = ['year', 'month', 'num_room'])
imputedInvstTest['full_sq_num_room'] = imputedInvstTest.full_sq / imputedInvstTest.full_sqNroom
ownrTestGroupMonthYearNroom = imputedOwnrTest.groupby(['year', 'month', 'num_room']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqNroom'})
imputedOwnrTest = pd.merge(imputedOwnrTest, ownrTestGroupMonthYearNroom, how = 'left', on = ['year', 'month', 'num_room'])
imputedOwnrTest['full_sq_num_room'] = imputedOwnrTest.full_sq / imputedOwnrTest.full_sqNroom

# full_sq by floor
invstTrainGroupMonthYearFloor = imputedInvstTrain.groupby(['year', 'month', 'floor']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqFloor'})
imputedInvstTrain = pd.merge(imputedInvstTrain, invstTrainGroupMonthYearFloor, how = 'left', on = ['year', 'month', 'floor'])
imputedInvstTrain['full_sq_floor'] = imputedInvstTrain.full_sq / imputedInvstTrain.full_sqFloor
ownrTrainGroupMonthYearFloor = imputedOwnrTrain.groupby(['year', 'month', 'floor']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqFloor'})
imputedOwnrTrain = pd.merge(imputedOwnrTrain, ownrTrainGroupMonthYearFloor, how = 'left', on = ['year', 'month', 'floor'])
imputedOwnrTrain['full_sq_floor'] = imputedOwnrTrain.full_sq / imputedOwnrTrain.full_sqFloor
invstTestGroupMonthYearFloor = imputedInvstTest.groupby(['year', 'month', 'floor']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqFloor'})
imputedInvstTest = pd.merge(imputedInvstTest, invstTestGroupMonthYearFloor, how = 'left', on = ['year', 'month', 'floor'])
imputedInvstTest['full_sq_floor'] = imputedInvstTest.full_sq / imputedInvstTest.full_sqFloor
ownrTestGroupMonthYearFloor = imputedOwnrTest.groupby(['year', 'month', 'floor']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqFloor'})
imputedOwnrTest = pd.merge(imputedOwnrTest, ownrTestGroupMonthYearFloor, how = 'left', on = ['year', 'month', 'floor'])
imputedOwnrTest['full_sq_floor'] = imputedOwnrTest.full_sq / imputedOwnrTest.full_sqFloor

# full_sq by sub_area
invstTrainGroupMonthYearSubArea = imputedInvstTrain.groupby(['year', 'month', 'sub_area']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqSubA
imputedInvstTrain = pd.merge(imputedInvstTrain, invstTrainGroupMonthYearSubArea, how = 'left', on = ['year', 'month', 'sub_area'])
imputedInvstTrain['full_sq_sub_area'] = imputedInvstTrain.full_sq / imputedInvstTrain.full_sqSubArea
imputedInvstTrain.drop(columns = ['full_sqMaterial', 'full_sqNroom', 'full_sqFloor', 'full_sqSubArea'], inplace = True)
ownrTrainGroupMonthYearSubArea = imputedOwnrTrain.groupby(['year', 'month', 'sub_area']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqSubAre
imputedOwnrTrain = pd.merge(imputedOwnrTrain, ownrTrainGroupMonthYearSubArea, how = 'left', on = ['year', 'month', 'sub_area'])
imputedOwnrTrain['full_sq_sub_area'] = imputedOwnrTrain.full_sq / imputedOwnrTrain.full_sqSubArea
imputedOwnrTrain.drop(columns = ['full_sqMaterial', 'full_sqNroom', 'full_sqFloor', 'full_sqSubArea'], inplace = True)
invstTestGroupMonthYearSubArea = imputedInvstTest.groupby(['year', 'month', 'sub_area']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqSubAre
imputedInvstTest = pd.merge(imputedInvstTest, invstTestGroupMonthYearSubArea, how = 'left', on = ['year', 'month', 'sub_area'])
imputedInvstTest['full_sq_sub_area'] = imputedInvstTest.full_sq / imputedInvstTest.full_sqSubArea
imputedInvstTest.drop(columns = ['full_sqMaterial', 'full_sqNroom', 'full_sqFloor', 'full_sqSubArea'], inplace = True)
ownrTestGroupMonthYearSubArea = imputedOwnrTest.groupby(['year', 'month', 'sub_area']).agg({'full_sq': 'mean'}).reset_index().rename(columns = {'full_sq': 'full_sqSubArea
imputedOwnrTest = pd.merge(imputedOwnrTest, ownrTestGroupMonthYearSubArea, how = 'left', on = ['year', 'month', 'sub_area'])
imputedOwnrTest['full_sq_sub_area'] = imputedOwnrTest.full_sq / imputedOwnrTest.full_sqSubArea
imputedOwnrTest.drop(columns = ['full_sqMaterial', 'full_sqNroom', 'full_sqFloor', 'full_sqSubArea'], inplace = True)

imputedInvstTrain.drop(columns = ['full_sq'], inplace = True)
imputedOwnrTrain.drop(columns = ['full_sq'], inplace = True)
imputedInvstTest.drop(columns = ['full_sq'], inplace = True)
imputedOwnrTest.drop(columns = ['full_sq'], inplace = True)
```

making sure all of the features lack missing values

In [ ]:
```
display(imputedInvstTrain.columns[imputedInvstTrain.isna().sum()>0])
display(imputedOwnrTrain.columns[imputedOwnrTrain.isna().sum()>0])
display(imputedInvstTest.columns[imputedInvstTest.isna().sum()>0])
display(imputedOwnrTest.columns[imputedOwnrTest.isna().sum()>0])
```

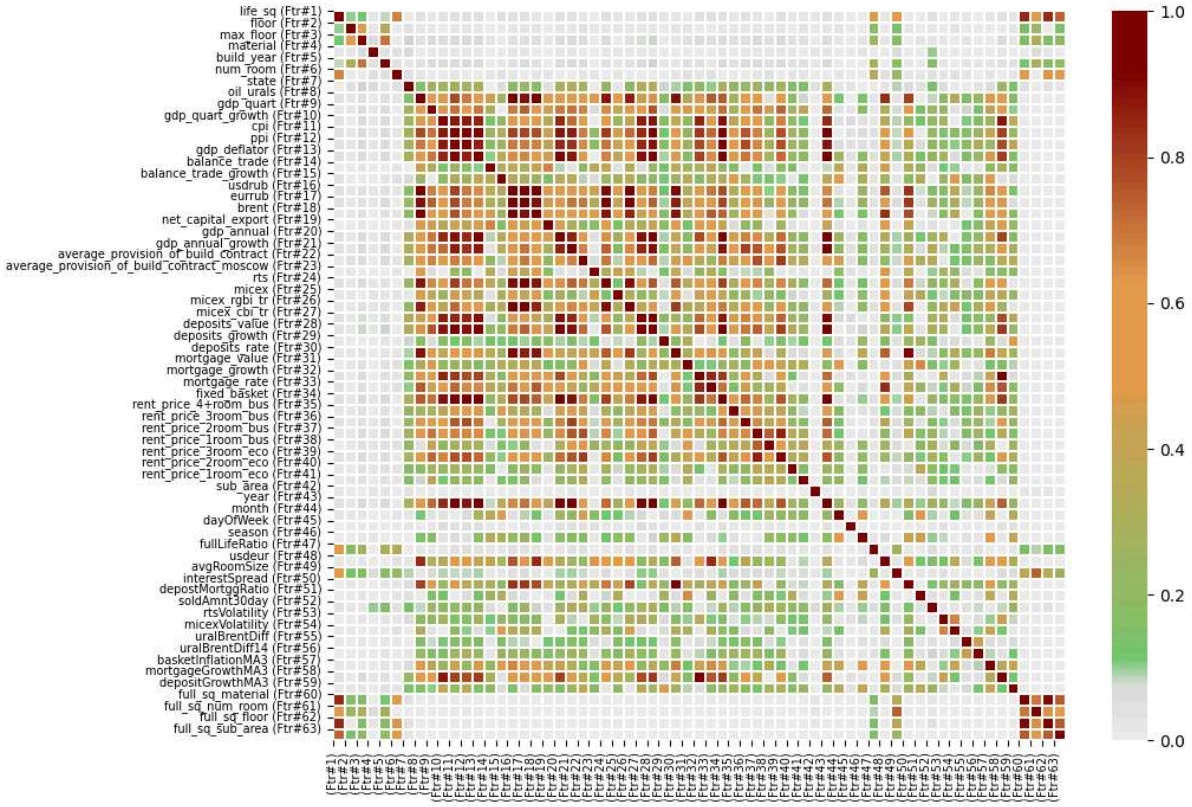
```
Index([], dtype='object')
Index([], dtype='object')
Index([], dtype='object')
Index([], dtype='object')
```

now lets separate to train and validation

In [ ]:
```
xInvstTrn = imputedInvstTrain.loc[:int(imputedInvstTrain.shape[0] * 70 / 85), :].drop(columns = ['price_doc', 'logPrice', 'timestamp'])
yInvstTrn = imputedInvstTrain.loc[:int(imputedInvstTrain.shape[0] * 70 / 85), 'logPrice'] + np.log(1.05)
xInvstVal = imputedInvstTrain.loc[int(imputedInvstTrain.shape[0] * 70 / 85):, :].drop(columns = ['price_doc', 'logPrice', 'timestamp'])
yInvstVal = imputedInvstTrain.loc[int(imputedInvstTrain.shape[0] * 70 / 85):, 'logPrice'] + np.log(1.05)
xOwnrTrn = imputedOwnrTrain.loc[:int(imputedOwnrTrain.shape[0] * 70 / 85), :].drop(columns = ['price_doc', 'logPrice', 'timestamp'])
yOwnrTrn = imputedOwnrTrain.loc[:int(imputedOwnrTrain.shape[0] * 70 / 85), 'logPrice'] + np.log(0.9)
xOwnrVal = imputedOwnrTrain.loc[int(imputedOwnrTrain.shape[0] * 70 / 85):, :].drop(columns = ['price_doc', 'logPrice', 'timestamp'])
yOwnrVal = imputedOwnrTrain.loc[int(imputedOwnrTrain.shape[0] * 70 / 85):, 'logPrice'] + np.log(0.9)
```

we'll start with the Investment model

In [ ]:
```
corrMatrixInvst = imputedInvstTrain.drop(columns = ['timestamp', 'price_doc', 'logPrice']).corr().abs()
fig, axs = plt.subplots(figsize = (10, 8))
ylabels = [col + f' (Ftr#{i+1})' for i, col in enumerate(corrMatrixInvst.columns)]
xlabels = [f' (Ftr#{i+1})' for i in range(corrMatrixInvst.shape[0])]
sns.heatmap(corrMatrixInvst, cmap = custom_cmap, linewidths = 0.1)
plt.xticks(ticks = range(corrMatrixInvst.shape[1]), labels = xlabels, fontsize = 7)
plt.yticks(ticks = range(corrMatrixInvst.shape[0]), labels = ylabels, fontsize = 7)
plt.show()
```



vary similar to the previous correlation matrices but the correlation seem weaker

even though the previous xgboost performed poorly, we'll use its parameters to perform the importance-correlation feature filtering

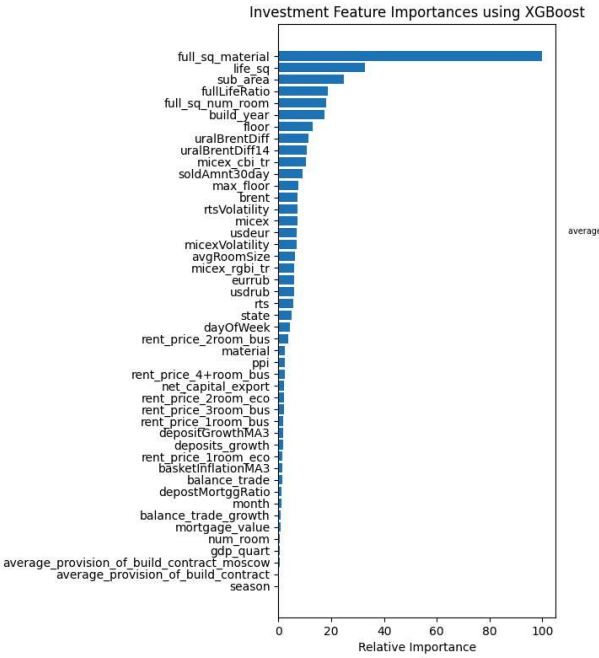
In [ ]:
```
invstImportanceXGBModel = GBR(**shortcutParams)
invstImportanceXGBModel.fit(xInvstTrn, yInvstTrn)
invstImportance = pd.Series(invstImportanceXGBModel.feature_importances_, index = xInvstTrn.columns).sort_values(ascending = True)
invstImportance = invstImportance/invstImportance.max()*100

groups = []
corr_threshold = 0.75
notInGroup = pd.Series([True for i in range(corrMatrixInvst.shape[1])], index = corrMatrixInvst.columns)

for feature in corrMatrixInvst.columns:
    if not groups:
        groups.append([feature])
        notInGroup[feature] = False
    else:
        if notInGroup[feature]:
            currGroup = [feature]
            notInGroup[feature] = False
            for f in corrMatrixInvst.columns:
                if (corrMatrixInvst.loc[f, feature] >= corr_threshold) & (notInGroup[f]):
                    currGroup.append(f)
                    notInGroup[f] = False
            groups.append(currGroup)

print(f'Feature Filteration Log - Investment XGBoost Model\n==================================================\n    threshold = {corr_threshold}\n')

removed = []

for i, group in enumerate(groups):
    if len(group) == 1:
        print(f"Group {i+1}:\n    - the feature ['{group[0]}'] is not correlated enough to any other feature.")
    else:
        avgImportance = invstImportance.loc[group].mean()
        col2Keep = invstImportance.loc[group][invstImportance.loc[group] >= avgImportance].index
        removed += [col for col in group if col not in col2Keep]
        imputedInvstTrain = imputedInvstTrain[[col for col in imputedInvstTrain.columns if col not in [c for c in group if c not in col2Keep]]]
        imputedInvstTest = imputedInvstTest[[col for col in imputedInvstTest.columns if col not in [c for c in group if c not in col2Keep]]]
        xInvstTrn = xInvstTrn[[col for col in xInvstTrn.columns if col not in [c for c in group if c not in col2Keep]]]
        xInvstVal = xInvstVal[[col for col in xInvstVal.columns if col not in [c for c in group if c not in col2Keep]]]
        corrMatrixInvst = corrMatrixInvst.loc[[col for col in corrMatrixInvst.columns if col not in [c for c in group if c not in col2Keep]], [col for col in corrMatrixInv
        print(f"Group {i+1}:\n    - features kept- {list(col2Keep)}\n    - features removed- {[col for col in group if col not in col2Keep]}")

print(f'removed {len(removed)} features:\n    - {removed}')
```

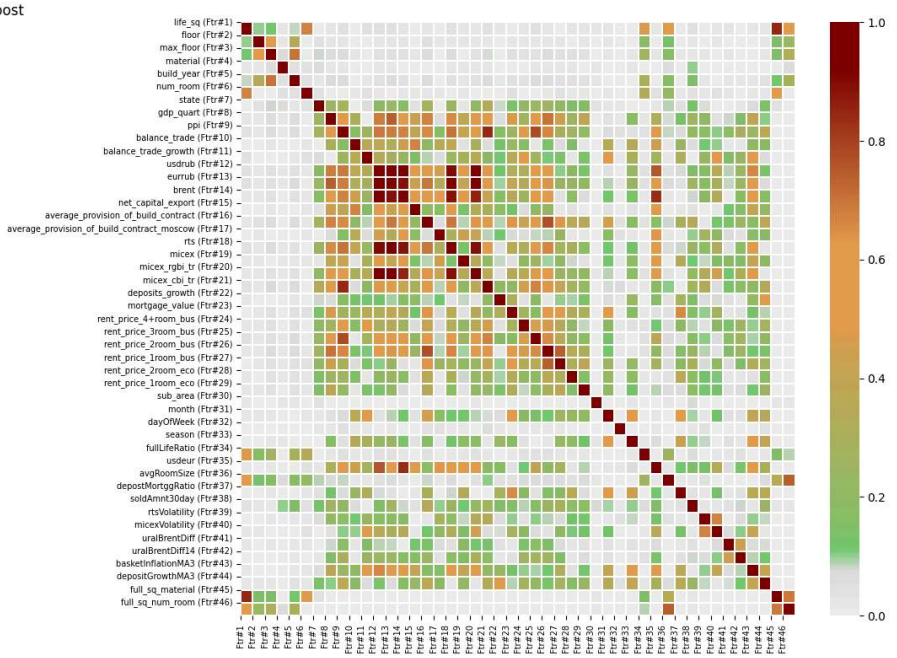
```
Feature Filteration Log - Investment XGBoost Model
==================================================
    threshold = 0.75

Group 1:
    - the feature ['life_sq'] is not correlated enough to any other feature.
Group 2:
    - the feature ['floor'] is not correlated enough to any other feature.
Group 3:
    - the feature ['max_floor'] is not correlated enough to any other feature.
Group 4:
    - the feature ['material'] is not correlated enough to any other feature.
Group 5:
    - the feature ['build_year'] is not correlated enough to any other feature.
Group 6:
    - the feature ['num_room'] is not correlated enough to any other feature.
Group 7:
    - the feature ['state'] is not correlated enough to any other feature.
Group 8:
    - features kept- ['usdrub', 'eurrub', 'brent', 'rts', 'micex_rgbi_tr', 'usdeur']
    - features removed- ['oil_urals', 'cpi', 'deposits_rate', 'mortgage_rate', 'interestSpread']
Group 9:
    - features kept- ['gdp_quart']
    - features removed- ['deposits_value']
Group 10:
    - features kept- ['ppi', 'micex_cbi_tr']
    - features removed- ['gdp_quart_growth', 'gdp_deflator', 'gdp_annual', 'mortgage_growth', 'fixed_basket', 'year', 'mortgageGrowthMA3']
Group 11:
    - the feature ['balance_trade'] is not correlated enough to any other feature.
Group 12:
    - the feature ['balance_trade_growth'] is not correlated enough to any other feature.
Group 13:
    - the feature ['net_capital_export'] is not correlated enough to any other feature.
Group 14:
    - features kept- ['rent_price_3room_bus', 'rent_price_2room_bus']
    - features removed- ['gdp_annual_growth', 'rent_price_3room_eco']
Group 15:
    - the feature ['average_provision_of_build_contract'] is not correlated enough to any other feature.
Group 16:
    - the feature ['average_provision_of_build_contract_moscow'] is not correlated enough to any other feature.
Group 17:
    - the feature ['micex'] is not correlated enough to any other feature.
Group 18:
    - the feature ['deposits_growth'] is not correlated enough to any other feature.
Group 19:
    - the feature ['mortgage_value'] is not correlated enough to any other feature.
Group 20:
    - the feature ['rent_price_4+room_bus'] is not correlated enough to any other feature.
Group 21:
    - the feature ['rent_price_1room_bus'] is not correlated enough to any other feature.
Group 22:
    - the feature ['rent_price_2room_eco'] is not correlated enough to any other feature.
Group 23:
    - the feature ['rent_price_1room_eco'] is not correlated enough to any other feature.
Group 24:
    - the feature ['sub_area'] is not correlated enough to any other feature.
Group 25:
    - the feature ['month'] is not correlated enough to any other feature.
Group 26:
    - the feature ['dayOfWeek'] is not correlated enough to any other feature.
Group 27:
    - the feature ['season'] is not correlated enough to any other feature.
Group 28:
    - the feature ['fullLifeRatio'] is not correlated enough to any other feature.
Group 29:
    - the feature ['avgRoomSize'] is not correlated enough to any other feature.
Group 30:
    - the feature ['depostMortggRatio'] is not correlated enough to any other feature.
Group 31:
    - the feature ['soldAmnt30day'] is not correlated enough to any other feature.
Group 32:
    - the feature ['rtsVolatility'] is not correlated enough to any other feature.
Group 33:
    - the feature ['micexVolatility'] is not correlated enough to any other feature.
Group 34:
    - the feature ['uralBrentDiff'] is not correlated enough to any other feature.
Group 35:
    - the feature ['uralBrentDiff14'] is not correlated enough to any other feature.
Group 36:
    - the feature ['basketInflationMA3'] is not correlated enough to any other feature.
Group 37:
    - the feature ['depositGrowthMA3'] is not correlated enough to any other feature.
Group 38:
    - features kept- ['full_sq_material']
    - features removed- ['full_sq_floor', 'full_sq_sub_area']
Group 39:
    - the feature ['full_sq_num_room'] is not correlated enough to any other feature.
removed features:
    - ['oil_urals', 'cpi', 'deposits_rate', 'mortgage_rate', 'interestSpread', 'deposits_value', 'gdp_quart_growth', 'gdp_deflator', 'gdp_annual', 'mortgage_growth', 'fixe
d_basket', 'year', 'mortgageGrowthMA3', 'gdp_annual_growth', 'rent_price_3room_eco', 'full_sq_floor', 'full_sq_sub_area']
```

In [ ]:

```
invstImportanceXGBModel = GBR(**shortcutParams)
invstImportanceXGBModel.fit(xInvstTrn, yInvstTrn)
invstImportance = pd.Series(invstImportanceXGBModel.feature_importances_, index = xInvstTrn.columns).sort_values(ascending = True)
invstImportance = invstImportance/invstImportance.max()*100

fig = plt.figure(figsize = (18, 8))
gs = GS.GridSpec(1, 2, width_ratios=[2, 5])

ax0 = plt.subplot(gs[0])
ax1 = plt.subplot(gs[1])

ax0.set_title('Investment Feature Importances using XGBoost')
ax0.barh(invstImportance.index, invstImportance.values, align='center')
ax0.set_xlabel('Relative Importance')

corrMatrixTicks = range(corrMatrixInvst.shape[0])
ylabels = [col + f' (Ftr#{i+1})' for i, col in enumerate(corrMatrixInvst.columns)]
xlabels = [f'Ftr#{i+1}' for i in corrMatrixTicks]
sns.heatmap(corrMatrixInvst, cmap = custom_cmap, linewidths = 0.1, ax=ax1)
ax1.set_xticks(ticks = corrMatrixTicks)
ax1.set_xticklabels(labels = xlabels, fontsize = 7)
```

```
ax1.set_yticks(ticks = corrMatrixTicks)
ax1.set_yticklabels(labels = ylabels, fontsize = 7)

plt.tight_layout()
plt.show()
```





here as well we see much more significance to other features rather than full_sq

we'll keep them all, meaning 25 of them

In [ ]:
```
invstFeatures2drop = invstImportance.head(invstImportance.shape[0] - 25).index

xInvstTrn.drop(columns = invstFeatures2drop, inplace = True)
xInvstVal.drop(columns = invstFeatures2drop, inplace = True)
imputedInvstTrain.drop(columns = invstFeatures2drop, inplace = True)
imputedInvstTest.drop(columns = invstFeatures2drop, inplace = True)
```

due to lack of time for a complete Cross Validation process, we'll perform a small scale process considering a narow range of hyper-parameters

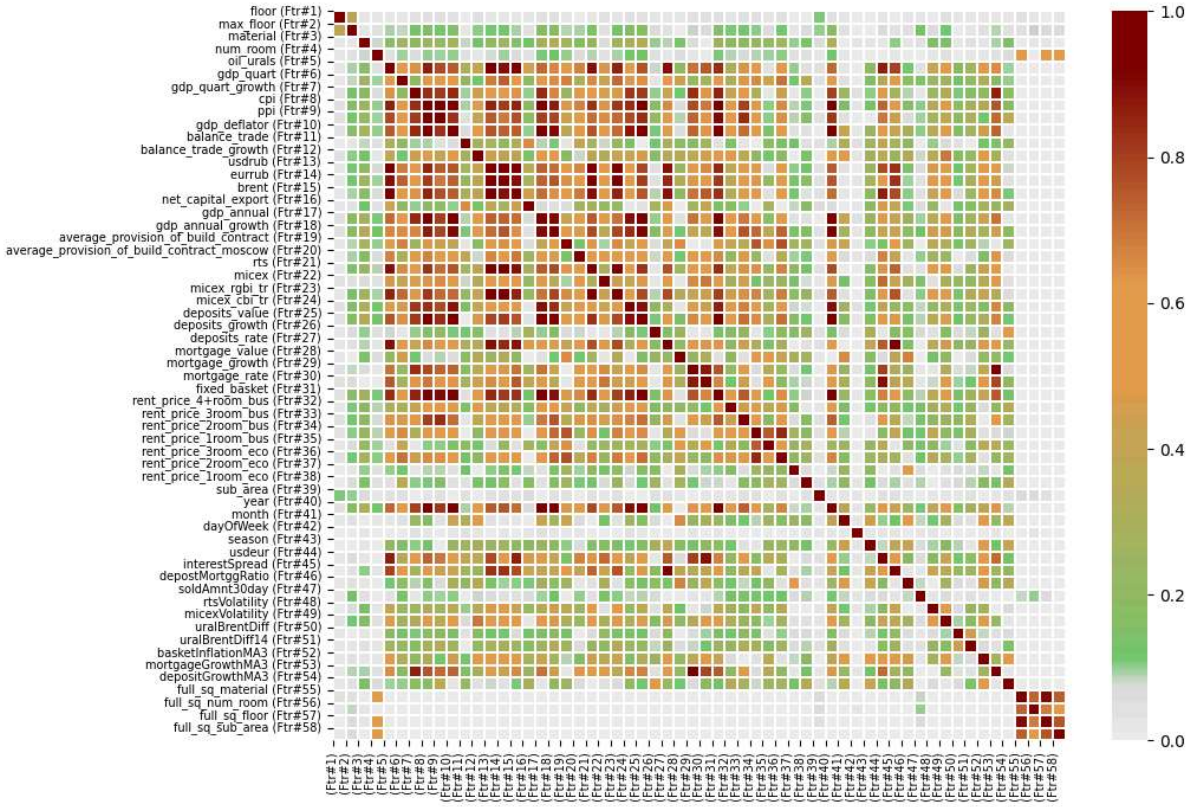
In [ ]:
```
invstXGBoostParams = {
    'learning_rate': [i/1000 for i in range(15, 30, 7)], # 0.015, 0.022, 0.029
    'n_estimators': [i for i in range(350, 426, 75)], # 350, 425
    'subsample': [i/20 for i in range(13, 16)], # 0.65, 0.7, 0.75
    'max_depth': [i for i in range(11, 15, 3)], # 11, 14
    'min_samples_leaf': [i for i in range(9, 12, 2)] # 9, 11
}

invstXGBoostGrdSrch = GSCV(GBR(), invstXGBoostParams, cv = 5, scoring = 'neg_mean_squared_error', verbose = 2)
invstXGBoostGrdSrch.fit(xInvstTrn, yInvstTrn)

invstXGBoostModelParams = invstXGBoostGrdSrch.best_params_

invstXGBoostOptModel = GBR(**invstXGBoostModelParams)
invstXGBoostOptModel.fit(xInvstTrn, yInvstTrn)
```

In [ ]:
```
shortcutInvstXGBoostParams = {
    'learning_rate': 0.015,
    'max_depth': 11,
    'min_samples_leaf': 11,
    'n_estimators': 350,
    'subsample': 0.75
}

invstXGBoostOptModel = GBR(**shortcutInvstXGBoostParams)
invstXGBoostOptModel.fit(xInvstTrn, yInvstTrn)
```

Out[ ]:
```
GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.015, max_depth=11,
                          min_samples_leaf=11, n_estimators=350,
                          subsample=0.75)
```

now lets make the OwnerOccupier model

In [ ]:
```
corrMatrixOwnr = imputedOwnrTrain.drop(columns = ['timestamp', 'price_doc', 'logPrice']).corr().abs()
fig, axs = plt.subplots(figsize = (10, 8))
ylabels = [col + f' (Ftr#{i+1})' for i, col in enumerate(corrMatrixOwnr.columns)]
xlabels = [f' (Ftr#{i+1})' for i in range(corrMatrixOwnr.shape[0])]
sns.heatmap(corrMatrixOwnr, cmap = custom_cmap, linewidths = 0.1)
plt.xticks(ticks = range(corrMatrixOwnr.shape[1]), labels = xlabels, fontsize = 7)
plt.yticks(ticks = range(corrMatrixOwnr.shape[0]), labels = ylabels, fontsize = 7)
plt.show()
```

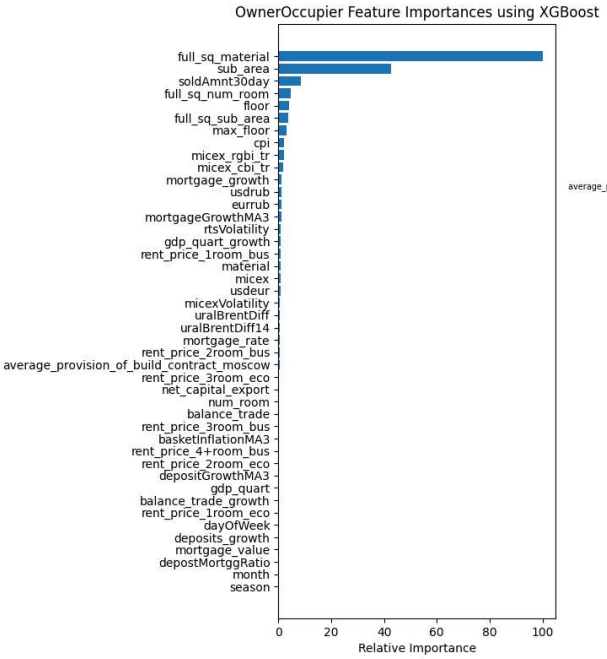


we see here much more correlation, emphasizing the difference between the Investment and the OwnerOccupier housing markets

In [ ]:
```
ownrImportanceXGBModel = GBR(**shortcutParams)
ownrImportanceXGBModel.fit(xOwnrTrn, yOwnrTrn)
ownrImportance = pd.Series(ownrImportanceXGBModel.feature_importances_, index = xOwnrTrn.columns).sort_values(ascending = True)
ownrImportance = ownrImportance/ownrImportance.max()*100

groups = []
corr_threshold = 0.75
notInGroup = pd.Series([True for i in range(corrMatrixOwnr.shape[1])], index = corrMatrixOwnr.columns)

for feature in corrMatrixOwnr.columns:
    if not groups:
        groups.append([feature])
        notInGroup[feature] = False
    else:
        if notInGroup[feature]:
            currGroup = [feature]
            notInGroup[feature] = False
            for f in corrMatrixOwnr.columns:
                if (corrMatrixOwnr.loc[f, feature] >= corr_threshold) & (notInGroup[f]):
                    currGroup.append(f)
                    notInGroup[f] = False
            groups.append(currGroup)

print(f'Feature Filteration Log - OwnerOccupier XGBoost Model\n==================================================\n    threshold = {corr_threshold}\n')

removed = []

for i, group in enumerate(groups):
    if len(group) == 1:
        print(f"Group {i+1}:\n    - the feature ['{group[0]}'] is not correlated enough to any other feature.")
    else:
        avgImportance = ownrImportance.loc[group].mean()
        col2Keep = ownrImportance.loc[group][ownrImportance.loc[group] >= avgImportance].index
        removed += [col for col in group if col not in col2Keep]
        imputedOwnrTrain = imputedOwnrTrain[[col for col in imputedOwnrTrain.columns if col not in [c for c in group if c not in col2Keep]]]
        imputedOwnrTest = imputedOwnrTest[[col for col in imputedOwnrTest.columns if col not in [c for c in group if c not in col2Keep]]]
        xOwnrTrn = xOwnrTrn[[col for col in xOwnrTrn.columns if col not in [c for c in group if c not in col2Keep]]]
        xOwnrVal = xOwnrVal[[col for col in xOwnrVal.columns if col not in [c for c in group if c not in col2Keep]]]
        corrMatrixOwnr = corrMatrixOwnr.loc[[col for col in corrMatrixOwnr.columns if col not in [c for c in group if c not in col2Keep]], [col for col in corrMatrixOwnr.c
        print(f"Group {i+1}:\n    - features kept- {list(col2Keep)}\n    - features removed- {[col for col in group if col not in col2Keep]}")

print(f'removed features:\n    - {removed}')
```

```
Feature Filteration Log - OwnerOccupier XGBoost Model
==================================================
    threshold = 0.75

Group 1:
    - the feature ['floor'] is not correlated enough to any other feature.
Group 2:
    - the feature ['max_floor'] is not correlated enough to any other feature.
Group 3:
    - the feature ['material'] is not correlated enough to any other feature.
Group 4:
    - the feature ['num_room'] is not correlated enough to any other feature.
Group 5:
    - features kept- ['cpi', 'usdrub', 'eurrub', 'micex_rgbi_tr', 'usdeur']
    - features removed- ['oil_urals', 'ppi', 'gdp_deflator', 'brent', 'rts', 'deposits_value', 'deposits_rate', 'fixed_basket', 'interestSpread']
Group 6:
    - the feature ['gdp_quart'] is not correlated enough to any other feature.
Group 7:
    - features kept- ['gdp_quart_growth', 'micex_cbi_tr', 'mortgage_growth', 'mortgageGrowthMA3']
    - features removed- ['gdp_annual', 'year']
Group 8:
    - the feature ['balance_trade'] is not correlated enough to any other feature.
Group 9:
    - the feature ['balance_trade_growth'] is not correlated enough to any other feature.
Group 10:
    - the feature ['net_capital_export'] is not correlated enough to any other feature.
Group 11:
    - features kept- ['rent_price_3room_bus']
    - features removed- ['gdp_annual_growth']
Group 12:
    - features kept- ['rent_price_2room_bus', 'rent_price_3room_eco']
    - features removed- ['average_provision_of_build_contract']
Group 13:
    - the feature ['average_provision_of_build_contract_moscow'] is not correlated enough to any other feature.
Group 14:
    - the feature ['micex'] is not correlated enough to any other feature.
Group 15:
    - the feature ['deposits_growth'] is not correlated enough to any other feature.
Group 16:
    - the feature ['mortgage_value'] is not correlated enough to any other feature.
Group 17:
    - the feature ['mortgage_rate'] is not correlated enough to any other feature.
Group 18:
    - the feature ['rent_price_4+room_bus'] is not correlated enough to any other feature.
Group 19:
    - the feature ['rent_price_1room_bus'] is not correlated enough to any other feature.
Group 20:
    - the feature ['rent_price_2room_eco'] is not correlated enough to any other feature.
Group 21:
    - the feature ['rent_price_1room_eco'] is not correlated enough to any other feature.
Group 22:
    - the feature ['sub_area'] is not correlated enough to any other feature.
Group 23:
    - the feature ['month'] is not correlated enough to any other feature.
Group 24:
    - the feature ['dayOfWeek'] is not correlated enough to any other feature.
Group 25:
    - the feature ['season'] is not correlated enough to any other feature.
Group 26:
    - the feature ['depostMortggRatio'] is not correlated enough to any other feature.
Group 27:
    - the feature ['soldAmnt30day'] is not correlated enough to any other feature.
Group 28:
    - the feature ['rtsVolatility'] is not correlated enough to any other feature.
Group 29:
    - the feature ['micexVolatility'] is not correlated enough to any other feature.
Group 30:
    - the feature ['uralBrentDiff'] is not correlated enough to any other feature.
Group 31:
    - the feature ['uralBrentDiff14'] is not correlated enough to any other feature.
Group 32:
    - the feature ['basketInflationMA3'] is not correlated enough to any other feature.
Group 33:
    - the feature ['depositGrowthMA3'] is not correlated enough to any other feature.
Group 34:
    - features kept- ['full_sq_material']
    - features removed- ['full_sq_floor']
Group 35:
    - the feature ['full_sq_num_room'] is not correlated enough to any other feature.
Group 36:
    - the feature ['full_sq_sub_area'] is not correlated enough to any other feature.
removed features:
    - ['oil_urals', 'ppi', 'gdp_deflator', 'brent', 'rts', 'deposits_value', 'deposits_rate', 'fixed_basket', 'interestSpread', 'gdp_annual', 'year', 'gdp_annual_growth',
'average_provision_of_build_contract', 'full_sq_floor']
```

In [ ]:
```
ownrImportanceXGBModel = GBR(**shortcutParams)
ownrImportanceXGBModel.fit(xOwnrTrn, yOwnrTrn)
ownrImportance = pd.Series(ownrImportanceXGBModel.feature_importances_, index = xOwnrTrn.columns).sort_values(ascending = True)
ownrImportance = ownrImportance/ownrImportance.max()*100

fig = plt.figure(figsize = (18, 8))
gs = GS.GridSpec(1, 2, width_ratios=[2, 5])

ax0 = plt.subplot(gs[0])
ax1 = plt.subplot(gs[1])

ax0.set_title('OwnerOccupier Feature Importances using XGBoost')
ax0.barh(ownrImportance.index, ownrImportance.values, align='center')
ax0.set_xlabel('Relative Importance')

corrMatrixTicks = range(corrMatrixOwnr.shape[0])
ylabels = [col + f' (Ftr#{i+1})' for i, col in enumerate(corrMatrixOwnr.columns)]
xlabels = [f'Ftr#{i+1}' for i in corrMatrixTicks]
sns.heatmap(corrMatrixOwnr, cmap = custom_cmap, linewidths = 0.1, ax=ax1)
ax1.set_xticks(ticks = corrMatrixTicks)
ax1.set_xticklabels(labels = xlabels, fontsize = 7)
ax1.set_yticks(ticks = corrMatrixTicks)
ax1.set_yticklabels(labels = ylabels, fontsize = 7)

plt.tight_layout()
plt.show()
```



it seems that only the top 7 are important enough to include, and that is a stretch aswell.

we'll keep only the seven of them

In [ ]:
```
ownrFeatures2drop = ownrImportance.head(ownrImportance.shape[0] - 7).index

xOwnrTrn.drop(columns = ownrFeatures2drop, inplace = True)
xOwnrVal.drop(columns = ownrFeatures2drop, inplace = True)
imputedOwnrTrain.drop(columns = ownrFeatures2drop, inplace = True)
imputedOwnrTest.drop(columns = ownrFeatures2drop, inplace = True)
```

here aswell, due to lack of time there will be only a short Cross Validation

In [ ]:
```
ownrXGBoostParams = {
    'learning_rate': [i/1000 for i in range(15, 30, 7)], # 0.015, 0.022, 0.029
    'n_estimators': [i for i in range(350, 426, 75)], # 350, 425
    'subsample': [i/20 for i in range(13, 16)], # 0.65, 0.7, 0.75
    'max_depth': [i for i in range(11, 15, 3)], # 11, 14
    'min_samples_leaf': [i for i in range(9, 12, 2)] # 9, 11
}

ownrXGBoostGrdSrch = GSCV(GBR(), ownrXGBoostParams, cv = 5, scoring = 'neg_mean_squared_error', verbose = 2)
ownrXGBoostGrdSrch.fit(xOwnrTrn, yOwnrTrn)

ownrXGBoostModelParams = ownrXGBoostGrdSrch.best_params_

ownrXGBoostOptModel = GBR(**ownrXGBoostModelParams)
ownrXGBoostOptModel.fit(xOwnrTrn, yOwnrTrn)
```

In [ ]:
```
shortcutOwnrXGBoostParams = {
    'learning_rate': 0.029,
    'max_depth': 11,
    'min_samples_leaf': 11,
    'n_estimators': 425,
    'subsample': 0.75
}

ownrXGBoostOptModel = GBR(**shortcutOwnrXGBoostParams)
ownrXGBoostOptModel.fit(xOwnrTrn, yOwnrTrn)
```

Out[ ]:
```
GradientBoostingRegressor(learning_rate=0.029, max_depth=11,
                          min_samples_leaf=11, n_estimators=425,
                          subsample=0.75)
```

now finally, lets make our prediction and metrics according to the Validation set

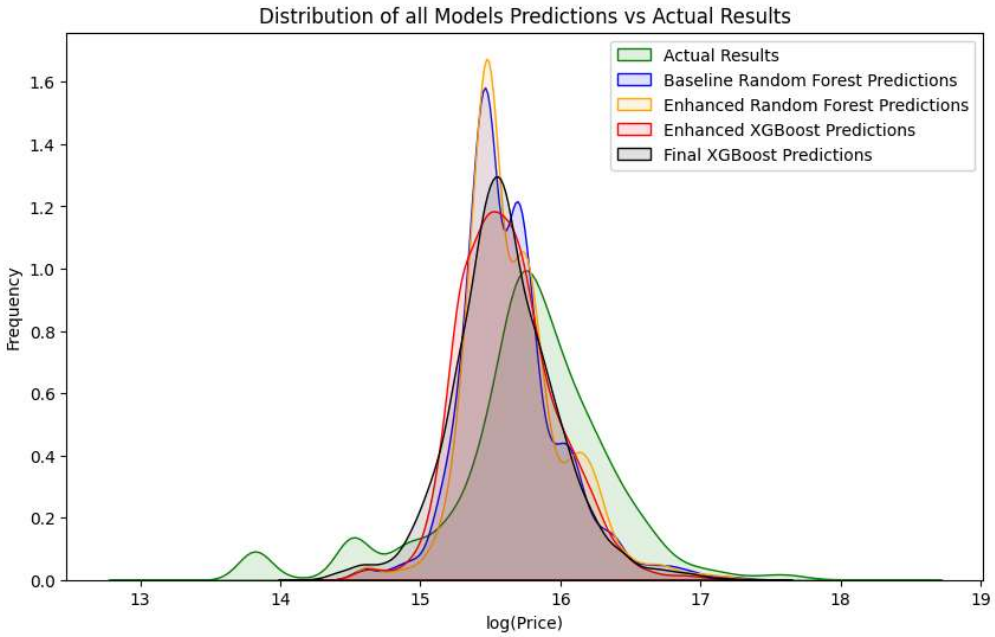
In [ ]:
```
invstXGBoostMseScores = -1 * CVS(invstXGBoostOptModel, xInvstVal, yInvstVal, cv = 5, scoring = 'neg_mean_squared_error')
ownrXGBoostMseScores = -1 * CVS(ownrXGBoostOptModel, xOwnrVal, yOwnrVal, cv = 5, scoring = 'neg_mean_squared_error')
yFinalVal = pd.Series(np.concatenate((yInvstVal, yOwnrVal)))
invstXGBoostPred = invstXGBoostOptModel.predict(xInvstVal)
ownrXGBoostPred = ownrXGBoostOptModel.predict(xOwnrVal)
finalXGBoostPred = pd.Series(np.concatenate((invstXGBoostPred, ownrXGBoostPred)))
finalXGBoostRmsle = np.sqrt(np.mean((finalXGBoostPred - yFinalVal)**2))
finalXGBoostR2 = 1 - (np.sum((finalXGBoostPred - yFinalVal)**2) / np.sum((yFinalVal - yFinalVal.mean())**2))

metrics.loc[metrics.shape[0]] = {'Model': 'Final XGBoost Model',
                                 'MSE': np.mean([invstXGBoostMseScores, ownrXGBoostMseScores]),
                                 'RMSLE': finalXGBoostRmsle,
                                 'R^2': finalXGBoostR2}

display(metrics)
```

	Model	MSE	RMSLE	R^2
0	Random Forest Model	0.192769	0.449120	0.409938
1	Enhanced Random Forest Model	0.255073	0.505923	0.326398
2	Enhanced XGBoost Model	0.249734	0.513840	0.305152
3	Final XGBoost Model	0.167448	0.477973	0.372510

already seems better than the rest by all metrics, except the RMSLE we calculated but their difference is close so this migh change in the kaggle competition

we should take a look at the distribution too

In [ ]:
```
# all comparison
plt.figure(figsize=(10, 6))
sns.kdeplot(yVal, color = 'green', fill = True, alpha = 0.1, label='Actual Results')
sns.kdeplot(rndFrstPred, color = 'blue', fill = True, alpha = 0.1, label='Baseline Random Forest Predictions')
sns.kdeplot(enhncRndFrstPred, color = 'orange', fill = True, alpha = 0.1, label='Enhanced Random Forest Predictions')
sns.kdeplot(enhncXGBoostPred, color = 'red', fill = True, alpha = 0.1, label='Enhanced XGBoost Predictions')
sns.kdeplot(finalXGBoostPred, color = 'black', fill = True, alpha = 0.1, label='Final XGBoost Predictions')
plt.xlabel('log(Price)')
plt.ylabel('Frequency')
plt.title('Distribution of all Models Predictions vs Actual Results')
plt.legend()
plt.show()
```



the distribution of the predictions from the new model has the the closest shape to the actual distribution out of all models!

In [ ]:
```
invstXGBoostOptModel = GBR(**shortcutInvstXGBoostParams)
invstXGBoostOptModel.fit(imputedInvstTrain.drop(columns = ['price_doc', 'logPrice', 'timestamp']), imputedInvstTrain.logPrice + np.log(1.05))
ownrXGBoostOptModel = GBR(**shortcutOwnrXGBoostParams)
ownrXGBoostOptModel.fit(imputedOwnrTrain.drop(columns = ['price_doc', 'logPrice', 'timestamp']), imputedOwnrTrain.logPrice + np.log(0.9))
```

Out[ ]:
```
GradientBoostingRegressor(learning_rate=0.029, max_depth=11,
                          min_samples_leaf=11, n_estimators=425,
                          subsample=0.75)
```

when submitting the predictions test data we get the following results in comparison to previous models:

	Private Score	Public Score
Random Forest Model	0.38628	0.38735
Enhanced Random Forest Model	0.37493	0.37117
Enhanced XGBoost Model	0.39436	0.39266
Final XGBoost Model	0.37355	0.36857

as expected to our graphs and metrics, this model is the best yet, but still not by much, even though it has the best scores out of all the models, it is very close to the scores of the enhanced random forest model