Machine Learning Sberbank Kaggle Competition

April 24, 2024

Authors: Lidor Erez 318661444, Daniel Mizrachi 208727230, Dvir Rehavi 207206624

1 Libraries

Install if you need:

[8]:
```
# !pip install plotly
# !pip install hyperopt
```

[2]:
```
# Data Manipulation
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.io as pio
pio.renderers.default='svg'

# Machine Learning
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.impute import KNNImputer
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error
from hyperopt import fmin, tpe, hp, STATUS_OK, Trials
from hyperopt.pyll.base import scope
```

[3]:
```
train = pd.read_csv("https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/
 ↪main/Data/train.csv")
test = pd.read_csv("https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/
 ↪main/Data/test.csv")
```

2 Baseline Models

[11]:
```python
cols = ['full_sq', 'life_sq', 'floor', 'max_floor', 'material',
'build_year', 'num_room', 'kitch_sq', 'state', 'product_type',␣
↪'sub_area','price_doc']
train_cp = train.copy(deep=True)[cols]
test_cp = test.copy(deep=True)[cols[:-1]]
```

2.1 Missing Values Handling

[12]:
```python
train_cp.isna().sum()
```

[12]:
```
full_sq             0
life_sq          6383
floor             167
max_floor        9572
material         9572
build_year      13605
num_room         9572
kitch_sq         9572
state           13559
product_type        0
sub_area            0
price_doc           0
dtype: int64
```

[13]:
```python
test_cp.isna().sum()
```

[13]:
```
full_sq            0
life_sq         1176
floor              0
max_floor          0
material           0
build_year      1049
num_room           0
kitch_sq           0
state            694
product_type      33
sub_area           0
dtype: int64
```

[14]:
```python
# Remove rows with more than 5 missing values.
train_cp = train_cp.loc[train_cp.isna().sum(axis=1) < 5]


# Handle missing values in build year
```

```python
train_cp.loc[(train_cp['build_year'] == 0) | (train_cp['build_year'] == 1),␣
↪'build_year'] = None
train_cp['build_year'].fillna(train_cp['build_year'].median(numeric_only=True),␣
↪inplace=True)

test_cp.loc[(test_cp['build_year'] == 0) | (test_cp['build_year'] == 1),␣
↪'build_year'] = None
test_cp['build_year'].fillna(test_cp['build_year'].median(numeric_only=True),␣
↪inplace=True)

# Handle Missing values in life_sq
train_cp['life_sq'].fillna(train_cp['life_sq'].mean(numeric_only=True),␣
↪inplace=True)
test_cp['life_sq'].fillna(test_cp['life_sq'].mean(numeric_only=True),␣
↪inplace=True)

# Handle Missing values in categorical
train_cp['state'].fillna(train_cp['state'].mode()[0], inplace=True)
test_cp['state'].fillna(test_cp['state'].mode()[0],inplace=True)

train_cp['material'].fillna(train_cp['material'].mode()[0], inplace=True)
test_cp['material'].fillna(test_cp['material'].mode()[0],inplace=True)

test_cp['product_type'].fillna(test_cp['product_type'].mode()[0],inplace=True)


# Check how many NAS there are:

print(f'Test data:\n {test_cp.isna().sum()}  \n\n Train Data: \n{train_cp.
↪isna().sum()}')
```

```
Test data:
 full_sq         0
life_sq         0
floor           0
max_floor       0
material        0
build_year      0
num_room        0
kitch_sq        0
state           0
product_type    0
sub_area        0
dtype: int64

 Train Data:
full_sq         0
```

```
life_sq         0
floor           0
max_floor       0
material        0
build_year      0
num_room        0
kitch_sq        0
state           0
product_type    0
sub_area        0
price_doc       0
dtype: int64
```

2.2 Data Types Handling

[15]:
```
train_cp.dtypes
```

[15]:
```
full_sq           int64
life_sq         float64
floor           float64
max_floor       float64
material        float64
build_year      float64
num_room        float64
kitch_sq        float64
state           float64
product_type     object
sub_area         object
price_doc         int64
dtype: object
```

[16]:
```
test_cp.dtypes
```

[16]:
```
full_sq         float64
life_sq         float64
floor             int64
max_floor         int64
material          int64
build_year      float64
num_room          int64
kitch_sq        float64
state           float64
product_type     object
sub_area         object
dtype: object
```

[17]:
```python
categorical_cols = ['state','material','sub_area','product_type']
to_int_cols = ['num_room','floor','max_floor','build_year']
to_float_cols = ['full_sq','price_doc']


# Changing some of the float columns to int.
train_cp[to_int_cols] = train_cp[to_int_cols].astype('int64')
test_cp[to_int_cols] = test_cp[to_int_cols].astype('int64')

# From int to float

train_cp[to_float_cols] = train_cp[to_float_cols].astype('float64')
test_cp[to_float_cols[:-1]] = test_cp[to_float_cols[:-1]].astype('float64')


# Categorical columns to int.
train_cp[categorical_cols[:2]] = train_cp[categorical_cols[:2]].astype('int64')
test_cp[categorical_cols[:2]] = test_cp[categorical_cols[:2]].astype('int64')


numeric_columns = list(train_cp.select_dtypes(['float64']).columns)
```

2.3 Outliers Handling

[18]:
```python
def categoricalOutlierHandler(df, cat_cols):
    vals_to_delete = {}

    for col in cat_cols:
        # Check for relative frequenecies
        freq = df[col].value_counts()
        # calculate the 10th quantile of the frequencies distribution
        q5 = freq.quantile(0.05)
        # Get rid of values that return less than the 10th quantile.
        vals_to_delete[col] = list(freq[freq < q5].index)

    for key in vals_to_delete:
         df = df[~df[key].isin(vals_to_delete[key])]

    return df


# categorical columns without product type (product type is binary).

train_cp = categoricalOutlierHandler(train_cp, ['state','material']).
 ↪reset_index(drop=True)
```

[19]:
```
def numericOutlierHandler(df,numeric_cols):
    for col in numeric_cols:
        # Calculate iqr
        q25 = np.quantile(df[col], 0.25)
        q75 = np.quantile(df[col], 0.75)
        iqr = q75 - q25
        lower_val = q25 - 1.5*iqr
        upper_val = q75 + 1.5*iqr

        # Cap the outliers using the upper val and lower val.
        df.loc[df[col] > upper_val, col] = upper_val
        df.loc[df[col] < lower_val, col] = lower_val
    return df

# Get all numeric columns
train_cp = numericOutlierHandler(train_cp, numeric_columns)
```

2.4 Categorical Features Handling

[20]:
```
# Product Type has only 2 values we can change it to binary
train_cp['product_type'] = train_cp['product_type'].apply(lambda x: (x ==␣
 ↪'Investment')*1)
test_cp['product_type'] = test_cp['product_type'].apply(lambda x:␣
 ↪(x=='Investment')*1)

# Dummies for sub area
train_cp = train_cp.drop(columns = 'sub_area').join(pd.
 ↪get_dummies(train_cp['sub_area'])*1).reset_index(drop=True)
test_cp = test_cp.drop(columns = 'sub_area').join(pd.
 ↪get_dummies(test_cp['sub_area'])*1).reset_index(drop=True)
```

2.5 Feature Engineering

[21]:
```
train_cp['room_sq'] = train_cp['num_room']/(train_cp['full_sq'] + 1e-10)
test_cp['room_sq'] = test_cp['num_room']/(test_cp['full_sq'] + 1e-10)
```

2.6 Train - Validation - Test Split

[22]:
```
X = train_cp.drop(columns = 'price_doc').reset_index(drop=True)
y = train_cp['price_doc']

x_train, x_testval, y_train, y_testval = train_test_split(X,y, test_size=0.3)
x_val, x_test, y_val, y_test = train_test_split(x_testval, y_testval,␣
 ↪test_size=0.15)
```

[23]:
```python
# Get all columns that are in X and not in Test Data Frame

X.columns.difference(test_cp.columns)
```

[23]: Index(['Poselenie Klenovskoe'], dtype='object')

[24]:
```python
test_cp['Poselenie Klenovskoe'] = 0
```

[25]:
```python
X.columns.difference(test_cp.columns)
```

[25]: Index([], dtype='object')

2.7 Random Forest Regressor

[26]:
```python
rf = RandomForestRegressor()
rf.fit(x_train, y_train)
# Get the most important features:
importance = rf.feature_importances_
```

[27]:
```python
# Get the most important features
important_features = list(x_train.columns[np.where(importance > np.
 ↪median(importance))])
x_train_subset = x_train[important_features]
rf.fit(x_train_subset,y_train)
```

[27]: RandomForestRegressor()

[28]:
```python
# Cross Validation
cv_mse =␣
 ↪cross_val_score(rf,x_val[important_features],y_val,cv=5,scoring='neg_mean_squared_error')
print(f'Random Forest MSE Score {np.mean(-cv_mse)}')
```

Random Forest MSE Score 5134281376847.779

[29]:
```python
mean_squared_error(rf.predict(x_test[important_features]),y_test)
```

[29]: 4673306534227.724

2.8 XGBoost Regressor

[30]:
```python
xgb = XGBRegressor()
params = {
    'learning_rate':np.arange(0.01,0.2),
    'max_depth' : np.arange(3,10),
    'gamma' :[0.0,0.1,0.2,0.3,0.4],
    'n_estimators': [500,600,700,800,900]
}
```

```
rs_model = RandomizedSearchCV(xgb, param_distributions=params, n_iter=5,␣
 ↪scoring='neg_mean_squared_error',cv=5,n_jobs=-1, verbose=3)
rs_model.fit(x_train, y_train)
rs_model.best_params_
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

[30]:
```
{'n_estimators': 600, 'max_depth': 8, 'learning_rate': 0.01, 'gamma': 0.4}
```

[31]:
```
new_xgb = XGBRegressor(**rs_model.best_params_)
new_xgb.fit(x_train, y_train)
cv_mse_xgb = cross_val_score(new_xgb,x_val, y_val, cv=5,␣
 ↪scoring='neg_mean_squared_error')
f'XGBoost Regressor MSE: {np.mean(-cv_mse_xgb)}'
```

[31]:
```
'XGBoost Regressor MSE: 4796309610025.951'
```

[32]:
```
mean_squared_error(new_xgb.predict(x_test),y_test)
```

[32]:
```
4521142820948.662
```

2.9 Kaggle Submission Files

[33]:
```
# predict test csv column and upload to kaggle.
xgb_preds = new_xgb.predict(test_cp[X.columns])
rf_preds = rf.predict(test_cp[important_features])

xgb_submit = pd.DataFrame({'id':test['id'],'price_doc':xgb_preds})
rf_submit = pd.DataFrame({'id':test['id'],'price_doc':rf_preds})
```

[34]:
```
# create csv files for submissions:

xgb_submit.to_csv('xgb_baseline_submission.csv',index=False)
rf_submit.to_csv('rf_baseline_submission.csv',index=False)
```

3 Notebooks Summary

3.1 Basic Time Series Analysis & Feature Selection

3.1.1 Methodology:

1. **Data Preprocessing:**
   a. Few functions for data transformation were created (log transformation).
   b. Columns that hold more 20% missing values and above were removed from the data set.
   c. Duplicated features were removed.
2. **Exploratory Data Analysis:**

a. The distribution of log price was examined.

b. The trend of price by months was examined.

3. **Feature Engineering:**

   a. Time frame columns: time frames columns such as year, month, and date were created.

   b. Log Price: log transformation to the response variable (price_doc)

4. **Feature Selection:**

   a. Correlation Test between every feature in the data set

   b. Correlation Test between every feature in the data set with log price

   c. Using XGBoost for feature selection.

   d. Choosing features according both correlation test and XGBoost importance.

5. **Data Preprocessing (Before deploying second XGBoost):**

   a. Missing values imputation using most frequent value.

   b. Normalizing the data using ‘l2’.

   c. Label Encoder was used to transform category features.

6. **XGBoost Model:**

   a. Training a model using K-Fold Cross Validation method with a rmse metric.

   b. Get the top 10 important features and examine their relationship with log price.

3.2 Results:

1. Some of the features are highly correlated with each other.
2. The highest correlation with the log price is less than 0.040.
3. Total area of the apartment is the most important feature to the first XGBoost model.
4. Time frame features created on the feature engineering phase have high importance.
5. Model’s RMSE is 0.46%

3.3 Conclusions:

1. Reducing the number of features is a crucial step because of the high correlation.
2. No strong linear relation with log price was detected.
3. Time features seem to contribute a lot to the model even though there’s no meaning to the order of the values (the values weren’t sorted).
4. After capping outliers for each of the top 10 features they seem to have a strong relationship with log price.

3.4 Critics:

1. Training XGBoost using cross validation is a smart decision, however when using a time series data, it's important to consider the time which the event occurred while training the model. Thus, we suggest using a time series version of the cross-validation method.
2. Removing correlated features from the data set might overcome this problem however one can also use PCA to reduce the dimensionality in the data. Thus, we suggest also trying using other methods but removing.

3.5 A Very Extensive Exploratory Analysis in Python

3.5.1 Methodology:

1. **Data Preprocessing:** Corrects data quality issues, ensuring the dataset's integrity and reliability.
2. **Exploratory Data Analysis:**
   a. Missing Values: Visualize the number of missing values in each column.
   b. Housing Internal Characteristics: Investigates features like floor area, number of rooms and building material to understand their influence on apartment prices.
   c. Time Series Analysis: The author Investigated how does price changes in several time frames.
   d. School Characteristics: Explores variables related to school facilities and their association with housing prices.
   e. Cultural Characteristics: Analyzes the impact of proximity to cultural landmarks and recreational facilities on property prices.
   f. Infrastructure Features: Examines proximity to infrastructure such as public transport, parks, and utilities in relation to housing prices.
   g. Variable Importance: Built a Random Forest Regressor model to understand which features are most important (categorical features were encoded using label encoder).
   h. Train – Test Comparison: Compared between the train and test data to understand if they're different or not.

3.5.2 Results:

1. **Housing Internal Characteristics:**
   a. Number of rooms in the apartment exhibits a high correlation with the total area of the apartment (makes sense).
   b. None of the features exhibit a strong linear relation with the price though number of rooms has a correlation score of 0.48 with the price.
   c. Housing internal features are the most important to the model.
2. **School , Cultural and Infrastructure Features:**
   a. Some of the features exhibits high correlation with each other.

b. There is no strong linear relation between the features and the price.

3.5.3 Conclusions:

1. Most important variables are housing internal features.
2. There might be redundant information in the data (high correlation between predictors).
3. The relation between price and other features is probably not linear. However, a statistical test needs to be done to support this assumption.
4. The train and test datasets are different from each other and might hold different values / number of values in each column.

3.5.4 Critic:

1. Categorical features were encoded using label encoder. However, in some cases it's better to use one hot encoding. Since label encoder might imply ordinality to a predictor which the ordinality is meaningless (such as sub area). Thus, we suggest using one hot encoding (dummies) instead.
2. Random Forest Regressor was used, and its performance wasn't tested. This could've been addressed by Cross Validation to provide a more robust assessment of its predictive capabilities and help validate the results of the model.
3. Instead of removing missing values, considering imputation or interpolation method could retain valuable information and prevent potential bias in the analysis.
4. While using Random Forest's feature importance is a smart decision, feature selection based on more than one method might lead to a better result. Thus, we suggest combining few feature selection methods to better understand which of the features contribute the most to the model.

4 Advanced Modeling

- Data Preprocessing (Integrity, Missing Values, Outliers)
- Feature Engineering
- Feature Selection (Importance, Correlation, Step-Wise regression?, LASSO)
- Hyperparameter Tuning (Bayesian Optimization, GridSearch, RandomSearch)
- Meta Learner (SVM, Linear Regression, Polynomial Regression, LASSO, Ridge)

4.1 Data Integrity (Quality)

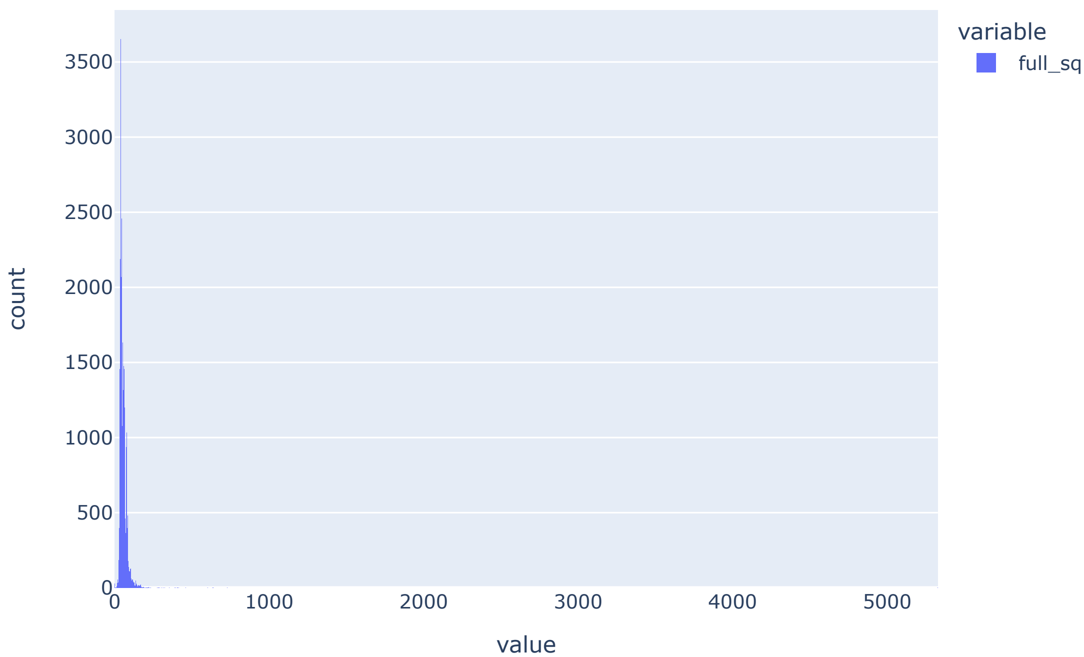
We made sure the data make sense in both train and test data frames

[35]:
```
full_data = pd.concat([train,test]).reset_index(drop=True)
```

[36]:
```
interior_cols = ['id', 'timestamp', 'full_sq', 'floor', 'num_room',␣
↪'product_type',
       'sub_area', 'price_doc',␣
↪'life_sq','max_floor','build_year','kitch_sq','state','material']
```

[37]:
```
interiors = full_data[interior_cols]
```

[38]:
```
px.histogram(interiors['full_sq'])
```

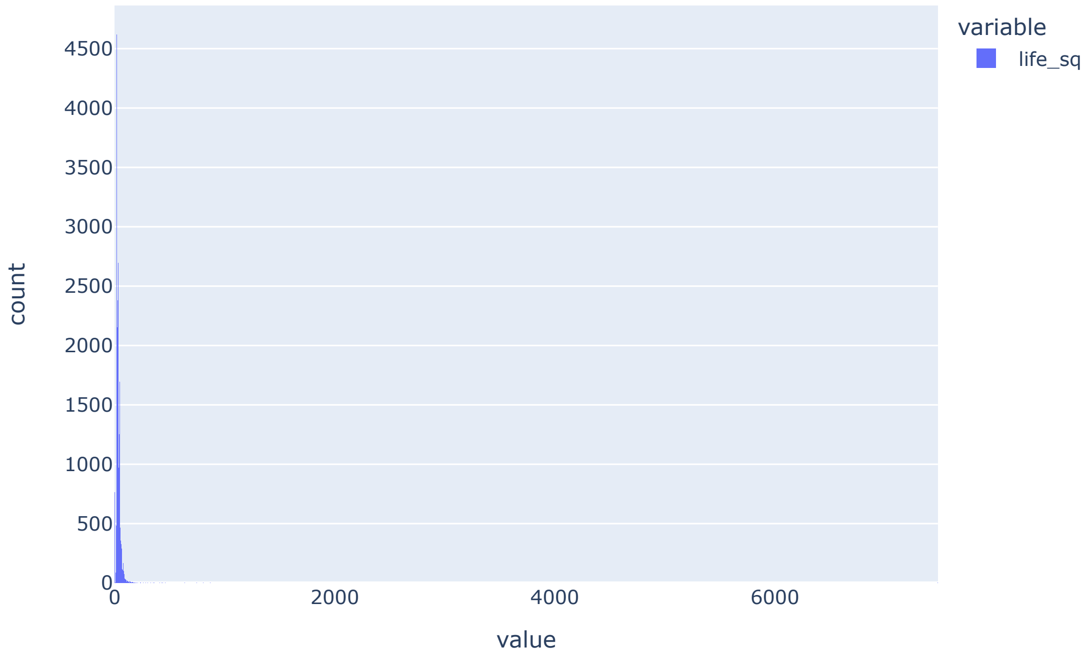


4.1.1 Full Square

[39]:
```
# Life square supposed to be at least 0.3 of the full square (conservative␣
↪value)
interiors.loc[interiors.full_sq < 10 ,'full_sq'] = np.nan # full sq is too low..
↪.
interiors.loc[(interiors.life_sq < 0.3*interiors.full_sq)*(interiors.full_sq >␣
↪210), 'full_sq'] = np.nan # full sq is probably wrong.
```

4.1.2 Life Square

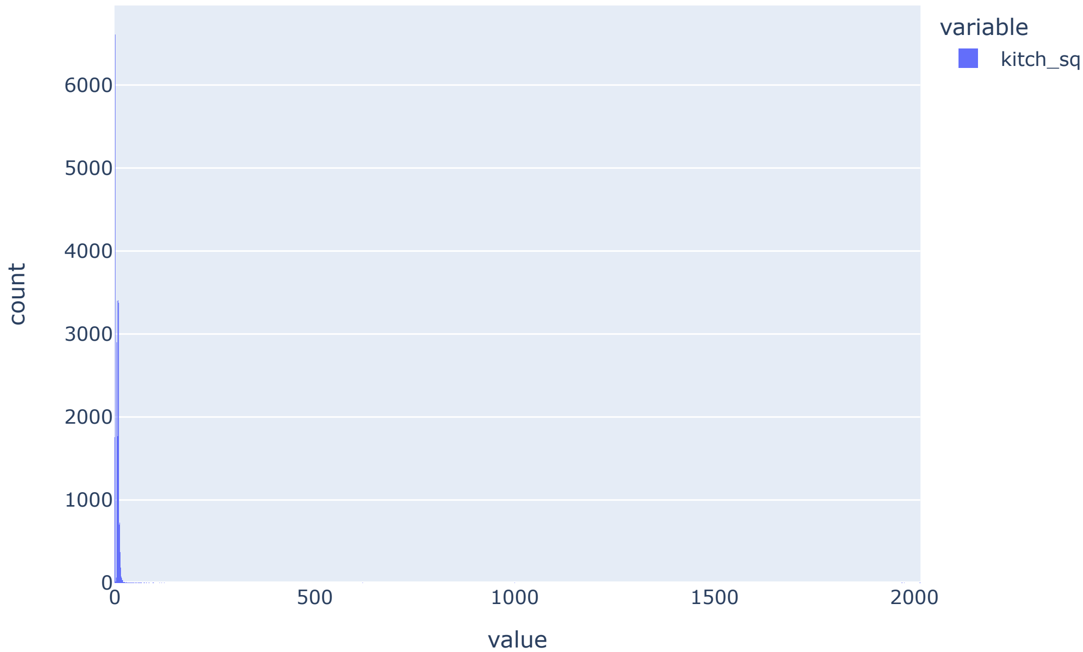
[40]:
```
px.histogram(interiors.life_sq)
```




```
[41]: interiors.loc[(interiors.life_sq < 5), 'life_sq'] = np.nan # deal with bad life␣
      ↪square.
      interiors.loc[(interiors.life_sq >= interiors.full_sq), 'life_sq'] = np.nan
```

4.1.3 Kitchen Square

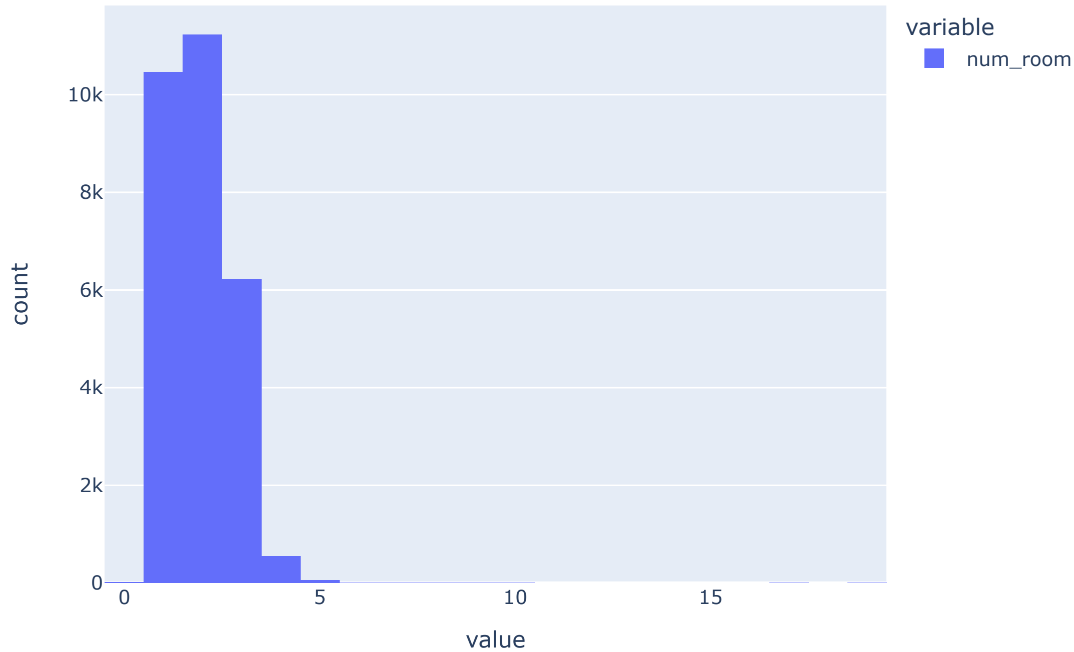
```
[42]: px.histogram(interiors['kitch_sq'])
```



[43]:
```
# Remove too small values
interiors.loc[interiors['kitch_sq'].isin([0,1,2,3]), 'kitch_sq'] = np.nan
# Remove abnormal values

interiors.loc[(interiors['kitch_sq'] >= interiors['full_sq']) |␣
↪(interiors['kitch_sq'] >= interiors['life_sq']) ,'kitch_sq'] = np.nan
```

4.1.4 Number of rooms

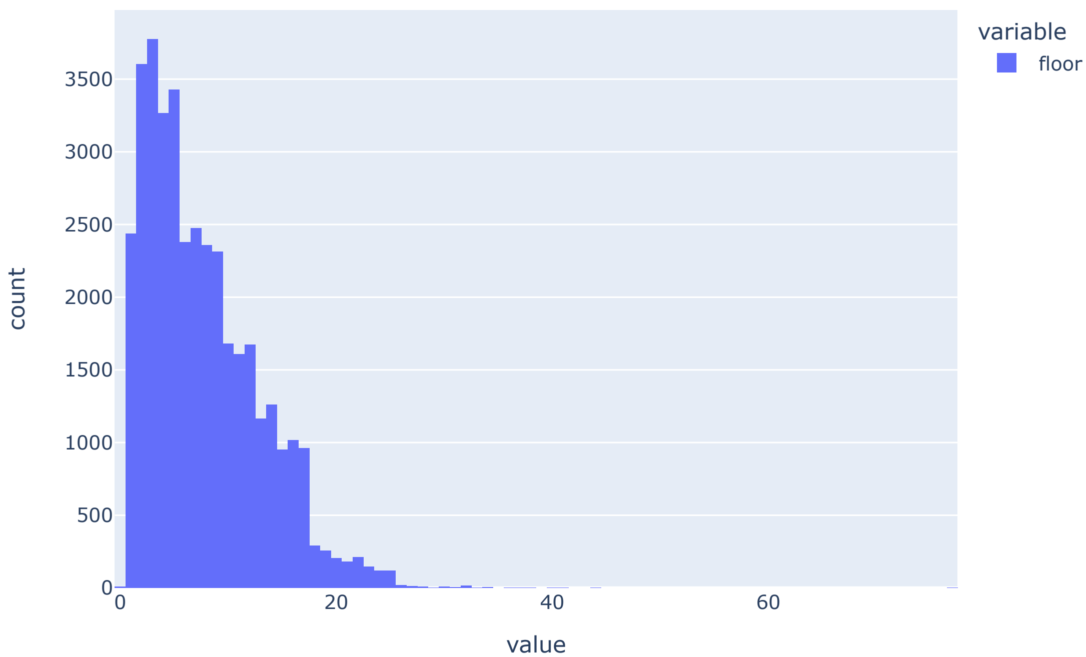
[44]:
```
px.histogram(interiors.num_room)
```



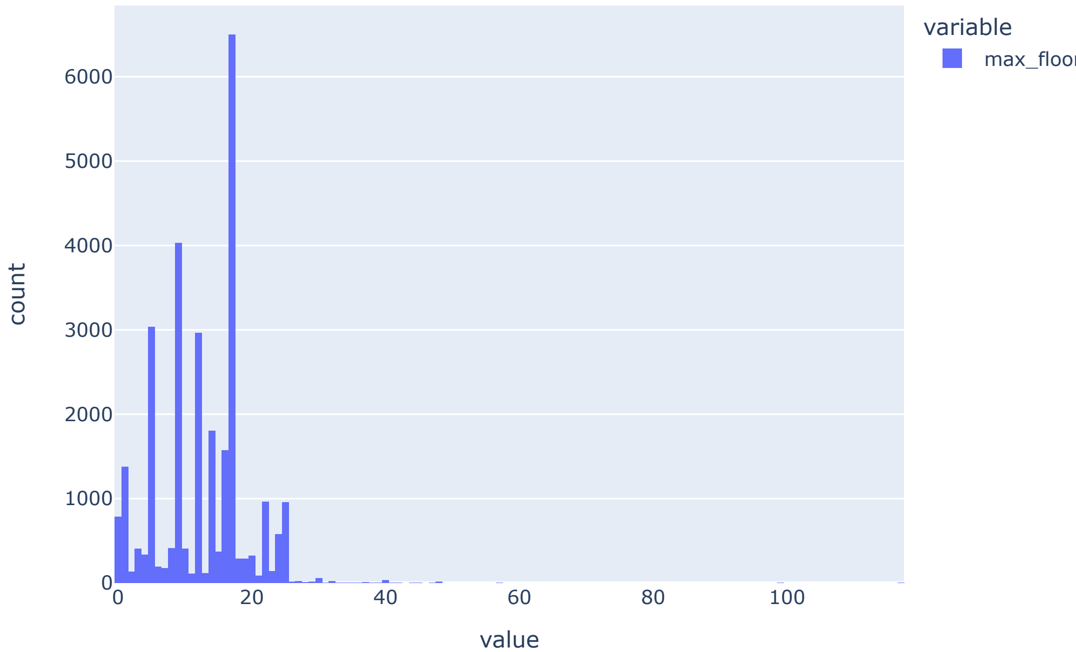
[45]:
```
# Finding apartment with a incorrect number of rooms.
threshold_for_sq_per_room = 5

interiors.loc[(interiors['life_sq']/interiors['num_room'] <␣
↪threshold_for_sq_per_room) | (interiors.num_room <= 0), 'num_room'] = np.nan
```

4.1.5 Floor & Max Floor

[46]:
```
px.histogram(interiors.floor)
```



[47]:
```
px.histogram(interiors.max_floor)
```



[48]:
```
interiors.loc[interiors['floor'] == 77, 'floor'] = np.nan
interiors.loc[interiors['floor'] > interiors['max_floor'], 'max_floor'] = np.nan
interiors.loc[interiors['floor'] == 0, 'floor'] = np.nan
interiors.loc[interiors['max_floor'] == 0, 'max_floor'] = np.nan
```

4.1.6 Build Year

[49]:
```
interiors.build_year.value_counts().index.unique()
```

[49]:
```
Index([    2014.0,     2015.0,        0.0,     2016.0,     2013.0,     2017.0,
              1.0,     1969.0,     1970.0,     1968.0,
       …
           1945.0,       71.0,     1904.0, 20052009.0,     1876.0,     1886.0,
           1925.0,     1691.0,       20.0,     1898.0],
      dtype='float64', name='build_year', length=127)
```

[50]:
```
# Build Year
interiors.loc[(interiors['build_year'] < 1500) , 'build_year'] = np.nan
interiors.loc[interiors['build_year'] == 4965,'build_year'] = 1965
interiors.loc[interiors['build_year'] == 20052009, 'build_year'] = np.nan
```

[51]:
```
# Product type
interiors['product_type'].value_counts()
```

[51]:
```
product_type
Investment       24446
OwnerOccupier    13654
Name: count, dtype: int64
```

[52]:
```
interiors['product_type'].isnull().sum()
```

[52]:
```
33
```

[53]:
```
interiors['product_type'].fillna(interiors['product_type'].mode()[0],␣
 ↪inplace=True)
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/3817222879.py:1
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[54]:
```
# State
print(interiors['state'].value_counts())
# Remove outlier.
interiors.loc[interiors['state'] == 33.0, 'state'] =  3
```

```
state
2.0     8506
3.0     7703
1.0     7121
4.0      549
33.0       1
Name: count, dtype: int64
```

[55]:
```
# Material
interiors['material'].value_counts()
```

[55]:
```
material
1.0    19438
2.0     3951
5.0     2048
4.0     1963
6.0     1159
3.0        2
Name: count, dtype: int64
```

[56]:
```
full_data[interiors.columns] = interiors
```

4.2 Data Cleaning

4.2.1 Missing Values - Shallow Cleaning

The way we cleaned our data:

1. Got the list of the features that has more than 30% missing values.
2. Calculated each feature (from the list above) correlation with the response variable.
3. If a feature had more than 30% missing values and a correlation below 0.3 (in absolute value) it was removed.

[57]:
```
missing_vals = pd.DataFrame(full_data.isnull().mean()*100)
features = missing_vals.index
vals = missing_vals.values.flatten()

# Create a dataframe with the pct of missing values for each feature
missing_vals = pd.DataFrame({'Features':features,'vals':vals})

# Get the feature
features_to_delete = missing_vals[missing_vals['vals'] > 30]['Features']

# Get the corrs with price
corss = np.round(full_data[['price_doc'] + features_to_delete.to_list()].
 ↪corr(),2).loc['price_doc']

# Remove the first correlation (price with itself).
corrs_with_price = corss.to_list()[1:]

corr_w_price = pd.DataFrame({'Features to delete':features_to_delete,'corr with␣
 ↪price':corrs_with_price})

corr_w_price
```

[57]:

	Features to delete	corr with price
3	life_sq	0.56
5	max_floor	0.13
7	build_year	0.04
9	kitch_sq	0.38
10	state	0.13
24	hospital_beds_raion	0.15
160	cafe_sum_500_min_price_avg	0.04
161	cafe_sum_500_max_price_avg	0.04
162	cafe_avg_price_500	0.04

We decided to not delete max floor, state, build year and hospital beds since they give us valuable information about surroundings, building and apartment

[58]:
```
corr_w_price.drop([7,24,5,10], inplace=True)
```

[59]:
```
# Get the features that has low correlation with the response variable
features_to_delete = corr_w_price[abs(corr_w_price['corr with price']) < 0.
↪3]['Features to delete'].tolist()
print(features_to_delete)
full_data = full_data.drop(columns = features_to_delete)
```

```
['cafe_sum_500_min_price_avg', 'cafe_sum_500_max_price_avg',
'cafe_avg_price_500']
```

4.2.2 Missing Values - Deep cleaning

By reading the data dictionary text file, we understood that some of the neighborhood features can be separated into groups. Thus, we created separated text files for each group of features and uploaded it to github so it'd be much easier to divide them into groups

[60]:
```
import requests

# Paths to text files from github
paths = ['https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/areas.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/buildings.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/demographics.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/distances.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/education.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/facilities.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/interior.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/surroundings.txt']

# Groups
groups = {
    'areas':[],
    'buildings':[],
    'demographics':[],
    'distances':[],
    'education':[],
    'facilities':[],
    'interior': [],
    'surroundings':[]
```

```
}

keys = list(groups.keys())

for i in range(len(paths)):
    # Http request to get the file data
    response = requests.get(paths[i])
    # read File lines
    lines = response.text.splitlines()
    for line in lines:
        # Split by \t
        col = line.split('\t')[0]
        # Interior file has : in it.
        if keys[i] == 'interior':
            col = col.split(':')[0]
        groups[keys[i]].append(col)
```

[61]:
```
def naIndicator(data, groups):
    nas_indicators = []
    for group in groups:
        data_cols = data.columns[np.where(data.columns.isin(groups[group]))]
        na_indicator = np.any(data[data_cols].isnull().sum() > 0)
        nas_indicators.append((group, na_indicator))
    return nas_indicators
```

[62]:
```
naIndicator(full_data, groups)
```

[62]:
```
[('areas', False),
 ('buildings', True),
 ('demographics', False),
 ('distances', True),
 ('education', True),
 ('facilities', True),
 ('interior', True),
 ('surroundings', True)]
```

It seems that few groups are still suffering from missing data.

[63]:
```
# Create a data frame for each group:
groups_dfs = {group:full_data[full_data.columns[np.where(full_data.columns.
 ↪isin(groups[group]))]] for group in groups}
```

Buildings

[64]:
```
buildings = groups_dfs['buildings']
# Get rows with missing values.
missing_vals = buildings.loc[buildings.isna().sum(axis=1) > 0]
```

```python
# Check if the missing values df has some data in it.
print(f'Number of rows in df: {missing_vals.shape[0]} \n\n Number of missing␣
↪values in each columns: \n\n {missing_vals.isna().sum()}')
```

```
Number of rows in df: 6209

 Number of missing values in each columns:

 raion_build_count_with_material_info     6209
build_count_block                        6209
build_count_wood                         6209
build_count_frame                        6209
build_count_brick                        6209
build_count_monolith                     6209
build_count_panel                        6209
build_count_foam                         6209
build_count_slag                         6209
build_count_mix                          6209
raion_build_count_with_builddate_info    6209
build_count_before_1920                  6209
build_count_1921-1945                    6209
build_count_1946-1970                    6209
build_count_1971-1995                    6209
build_count_after_1995                   6209
dtype: int64
```

In order to fill these columns we need domain knowledge. However each of these columns represent an amount therefore we'll replace nan values with 0.

[65]:
```python
buildings[buildings.filter(like='count').columns] = buildings[buildings.
↪filter(like='count').columns].replace(np.nan, 0) # 0 for count data
buildings[buildings.filter(like='info').columns] = buildings[buildings.
↪filter(like='info').columns].replace(np.nan,-1)
buildings.isnull().sum()
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/222925215.py:1:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/222925215.py:2:
SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[65]:
```
raion_build_count_with_material_info     0
build_count_block                        0
build_count_wood                         0
build_count_frame                        0
build_count_brick                        0
build_count_monolith                     0
build_count_panel                        0
build_count_foam                         0
build_count_slag                         0
build_count_mix                          0
raion_build_count_with_builddate_info    0
build_count_before_1920                  0
build_count_1921-1945                    0
build_count_1946-1970                    0
build_count_1971-1995                    0
build_count_after_1995                   0
dtype: int64
```

[66]:
```
groups_dfs['buildings'] = buildings
```

[67]:
```
groups_with_missing_vals = ['interior','distances','surroundings']
```

Interior

[68]:
```
interior = groups_dfs['interior']
# Get interior Data Frame
interior_na = interior.isnull().sum()
print(interior_na[interior_na > 0])
```

```
full_sq          40
life_sq       11762
floor           177
max_floor     11711
material       9572
build_year    16116
num_room       9600
kitch_sq      18118
state         14253
price_doc      7662
dtype: int64
```

Few Helpers:

[69]:
```python
def handleFullsqLifesq(data, nas_full, nas_life, sa_med_fullsq,␣
↪sa_life_full_prop):
    data['missing_life'] = data['life_sq'].isnull().astype(int)
    data['missing_full'] = data['full_sq'].isnull().astype(int)
    for index in data.index:
        sub_area = data.loc[index,'sub_area']

        fullsq = data.loc[index, 'full_sq']
        lifesq = data.loc[index, 'life_sq']

        mask_sa_pct = (sa_life_full_prop['sub_area'].isin([sub_area]))
        mask_sa_full = (sa_med_fullsq['sub_area'].isin([sub_area]))

        sa_prop = sa_life_full_prop.loc[mask_sa_pct]['PCT'].values[0]
        full_sq_med =sa_med_fullsq.loc[mask_sa_full]['median'].values[0]

        flag = 0

        """
        If life sq is missing - fill it with full sq * avg pct of life sq out␣
↪of full sq.
        If full sq is missing - fill it with life sq / avg pct of life sq out␣
↪of full sq.
        If both are missing - fill full sq with median of full sq and then␣
↪calculate life sq using the median value.
        """
        if not nas_full[index] and nas_life[index]:
            flag = 2
            data.loc[index, 'life_sq'] = fullsq * sa_prop
        elif not nas_life[index] and nas_full[index]:
            flag = 3
            data.loc[index, 'full_sq'] = lifesq / sa_prop
        elif nas_life[index] and nas_full[index]:
            flag = 4
            data.loc[index, 'full_sq'] = full_sq_med # get median value to fill␣
↪na
            data.loc[index, 'life_sq'] =  full_sq_med * sa_prop # calculate␣
↪life sq using the median value

        # if check to see if the algorithm work correctly.
        if np.any(data['life_sq'] >= data['full_sq']):
            print(flag)
            break
    return {1:'Done', 'Full SQ': data['full_sq'].isna().sum(), 'Life SQ':␣
↪data['life_sq'].isna().sum(), 'Life SQ >= Full SQ': (data['life_sq'] >=␣
↪data['full_sq']).any()}
```

[70]:
```
def HandleFloorMaxFloor(data, sa):
    data['missing_floor'] = data['floor'].isnull().astype(int)
    data['missing_maxfloor'] = data['max_floor'].isnull().astype(int)
    # fill na in max floor
    med_max_floor = sa['max_floor'].transform('median')
    med_max_floor = np.round(med_max_floor.fillna(med_max_floor.median())) #␣
↪fill the missing value with the median of medians.
    data['max_floor'] = data['max_floor'].fillna(med_max_floor)
    # Fill floor:
    med_floor = sa['floor'].transform('median')
    med_floor = np.round(med_floor.fillna(med_floor.median())) # fill the␣
↪missing value with the median of medians.
    data.loc[:,'floor'] = data['floor'].fillna(med_floor)

    data.loc[data['floor'] > data['max_floor'],'floor'] = data['max_floor']

    return {1:'Done', 'Max Floor': data['max_floor'].isna().sum(), 'Floor':␣
↪data['floor'].isna().sum(), 'Floor > Max Floor': (data['floor']>␣
↪data['max_floor']).any()}
```

[71]:
```
def HandleNumRooms(data, sa):
    data['missing_num_room'] = data['num_room'].isnull().astype(int)
    # fill na with the median number of rooms for each sub area and product␣
↪type.
    data.loc[:,'num_room'] = data['num_room'].fillna(sa['num_room'].
↪transform('median'))

    # Make sure data makes sense.
    mask = data['life_sq']/data['num_room'] < 5
    #  5 square meter per room is a conservative value.
    data.loc[mask | (data['num_room'].isna()), 'num_room'] = data['life_sq'] //␣
↪5
    data.loc[data['num_room'] == 0, 'num_room'] = 1

    return {1:'Done', 'Num Room': data['num_room'].isna().sum(), 'LifeSQ/
↪NumRoom < 5': (data['life_sq']/data['num_room']<5).any()}
```

[72]:
```
def HandleBuildYear(data):
    data['missing_build_year'] = data['build_year'].isnull().astype(int)

    sa_median_build_year = data.groupby(['sub_area'])['build_year'].
↪transform('median')

    data.loc[:,'build_year'] = data['build_year'].fillna(sa_median_build_year)

    return {1:"Done","Build_Year": data['build_year'].isna().sum()}
```

[73]:
```python
def handleStateMaterial(data):
    data['missing_state'] = data['state'].isnull().astype(int)
    data['missing_material'] = data['material'].isnull().astype(int)

    data['year'] = pd.to_datetime(data['timestamp']).dt.year
    data['age'] = data['year'] - data['build_year']

    state_modes = data['state'].fillna(data.
↪groupby(['age','sub_area'])['state'].transform(lambda val: val.mode()[0] if␣
↪len(val.mode())>0 else None))
    state_modes = state_modes.fillna(state_modes.mode()[0])

    data.loc[:,'state'] = data['state'].fillna(state_modes)

    material_modes = data['material'].fillna(data.
↪groupby(['build_year','sub_area'])['material'].transform(lambda val: val.
↪mode()[0] if len(val.mode())>0 else None))
    material_modes = material_modes.fillna(material_modes.mode()[0])
    data.loc[:,'material'] = data['material'].fillna(material_modes)

    return {1:'Done', 'State': data['state'].isna().sum(), 'Material':␣
↪data['material'].isna().sum()}
```

[74]:
```python
def HandleKitchSQ(data, sa_kitch_life_prop):
    data['missing_kitch_sq'] = data['kitch_sq'].isnull().astype(int)
    for idx in data.index:
        life_sq = data.loc[idx,'life_sq']
        kitch_sq = data.loc[idx,'kitch_sq']
        product_type = data.loc[idx,'product_type']
        sub_area = data.loc[idx,'sub_area']

        if not np.isnan(kitch_sq) and kitch_sq < life_sq:
            continue
        mask = (sa_kitch_life_prop['sub_area'].isin([sub_area]))
        avg_ratio = sa_kitch_life_prop.loc[mask]['PCT'].values[0]

        data.loc[idx, 'kitch_sq'] = life_sq * avg_ratio

    return {1:'Done', 'Kitch SQ': data['kitch_sq'].isna().sum(), 'Kitch SQ >=␣
↪Full SQ': (data['kitch_sq'] >= data['full_sq']).any(), 'Kitch SQ >= Life SQ':
↪ (data['kitch_sq'] >= data['life_sq']).any()}
```

[75]:
```python
# Calculate life sq / full sq
interior.loc[:,'life_full_ratio'] = interior['life_sq']/interior['full_sq']
(interior['life_full_ratio'] >= 1).any()
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/4061903555.py:2
: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[75]: False

[76]:
```python
# product type and sub area group by
sa_grps = interior.groupby(['sub_area'])

# Calculate the avg pct of life sq out of full sq for each  sub area:
avg_life_full_prop = (sa_grps['life_full_ratio'].mean()).reset_index().
↪rename(columns={'life_full_ratio':'PCT'})
```

[77]:
```python
# median full sq
sa_med_fullsq = sa_grps['full_sq'].median().reset_index().rename(columns =␣
↪{'full_sq':'median'}) # cause median is robust to outliers.

# Get the NAS of full sq
nas_full = interior['full_sq'].isnull()
# Get the NAS of life sq
nas_life = interior['life_sq'].isnull()
```

[78]:
```python
handleFullsqLifesq(interior, nas_full, nas_life, sa_med_fullsq,␣
↪avg_life_full_prop)
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/2830605670.py:2
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/2830605670.py:3
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

[78]: {1: 'Done', 'Full SQ': 0, 'Life SQ': 0, 'Life SQ >= Full SQ': False}

[79]:
```
interior = interior.drop(columns = 'life_full_ratio')
```

[80]:
```
HandleFloorMaxFloor(interior, sa_grps)
```

[80]: {1: 'Done', 'Max Floor': 0, 'Floor': 0, 'Floor > Max Floor': False}

[81]:
```
HandleNumRooms(interior, sa_grps)
```

[81]: {1: 'Done', 'Num Room': 0, 'LifeSQ/NumRoom < 5': False}

[82]:
```
# Sanity check:
interior.isna().sum()
```

[82]:
```
id                     0
timestamp              0
full_sq                0
life_sq                0
floor                  0
max_floor              0
material            9572
build_year         16116
num_room               0
kitch_sq           18118
state              14253
product_type           0
sub_area               0
price_doc           7662
missing_life           0
missing_full           0
missing_floor          0
missing_maxfloor       0
missing_num_room       0
dtype: int64
```

[83]:
```
HandleBuildYear(interior)
```

[83]: {1: 'Done', 'Build_Year': 0}

[84]:
```
handleStateMaterial(interior)
```

[84]: {1: 'Done', 'State': 0, 'Material': 0}

[85]:
```
interior.drop(columns = ['year','age'], inplace=True) # Drop cols.
```

[86]:
```
interior.isna().sum()
```

[86]:
```
id                      0
timestamp               0
full_sq                 0
life_sq                 0
floor                   0
max_floor               0
material                0
build_year              0
num_room                0
kitch_sq            18118
state                   0
product_type            0
sub_area                0
price_doc            7662
missing_life            0
missing_full            0
missing_floor           0
missing_maxfloor        0
missing_num_room        0
missing_build_year      0
missing_state           0
missing_material        0
dtype: int64
```

[87]:
```
# Fill missing values in kitch_sq column

# Get the kitch_sq/full sq
interior['kitch_life_ratio'] = interior['kitch_sq']/interior['life_sq']
interior['kitch_life_ratio'] = interior['kitch_life_ratio'].
 ↪fillna(interior['kitch_life_ratio'].mean()) # Fill missing values with the␣
 ↪mean.

# Get the avg kitch_sq/full sq for each sub area and product type.
sa_avg_kitch_life = interior.groupby(['sub_area'])['kitch_life_ratio'].mean().
 ↪reset_index().rename(columns={'kitch_life_ratio':'PCT'})
```

[88]:
```
HandleKitchSQ(interior, sa_avg_kitch_life)
```

[88]:
```
{1: 'Done',
 'Kitch SQ': 0,
 'Kitch SQ >= Full SQ': False,
 'Kitch SQ >= Life SQ': False}
```

[89]:
```
groups_dfs['interior'] = interior.drop(columns = ['kitch_life_ratio'])
```

Distances

[90]:
```python
# Get distances data frame
distances = groups_dfs['distances']
distances_na = distances.isnull().sum()
# Get the features that have missing values.
distances_na[distances_na > 0]
```

[90]:
```
metro_min_walk             59
metro_km_walk              59
railroad_station_walk_km   59
railroad_station_walk_min  59
ID_railroad_station_walk   59
dtype: int64
```

[91]:
```python
distances['sub_area'] = full_data['sub_area']
for c in distances_na[distances_na > 0].index:
    if 'ID' not in c:
        distances[c].fillna(distances.groupby('sub_area')[c].
 ↪transform('median'), inplace=True)
    else:
        distances[c].fillna(distances.groupby('sub_area')[c].transform(lambda x:
 ↪ x.mode()[0]), inplace=True)
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/3585862785.py:1
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/3585862785.py:4
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/3585862785.py:4
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/3585862785.py:4
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/3585862785.py:4
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/3585862785.py:6
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[92]:
```
distances.drop(columns='sub_area', inplace=True)
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_66217/2186917241.py:1
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[93]:
```
groups_dfs['distances'] = distances
```

Surroundings

[94]:
```
# Get surroundings data frame
surroundings = groups_dfs['surroundings']
surr_na = surroundings.isnull().sum()
```

```python
# Get the features that have missing values.
surr_na[surr_na > 0]
```

[94]:
```
cafe_sum_1000_min_price_avg    7746
cafe_sum_1000_max_price_avg    7746
cafe_avg_price_1000            7746
cafe_sum_1500_min_price_avg    5020
cafe_sum_1500_max_price_avg    5020
cafe_avg_price_1500            5020
green_part_2000                  19
cafe_sum_2000_min_price_avg    2149
cafe_sum_2000_max_price_avg    2149
cafe_avg_price_2000            2149
cafe_sum_3000_min_price_avg    1173
cafe_sum_3000_max_price_avg    1173
cafe_avg_price_3000            1173
prom_part_5000                  270
cafe_sum_5000_min_price_avg     425
cafe_sum_5000_max_price_avg     425
cafe_avg_price_5000             425
dtype: int64
```

To address these missing values, we decided to use KNNImputer since each feature here represent an avg price of a shop within a certain distance. However, before doing that we need to check the variabilility of each feature which is the STD/Mean.

[95]:
```python
# Get the columns
surr_na_cols = surr_na[surr_na > 0].index
# Check the std/mean ratio
surroundings[surr_na_cols].std()/surroundings[surr_na_cols].mean()
```

[95]:
```
cafe_sum_1000_min_price_avg    0.318793
cafe_sum_1000_max_price_avg    0.288062
cafe_avg_price_1000            0.298280
cafe_sum_1500_min_price_avg    0.271707
cafe_sum_1500_max_price_avg    0.245080
cafe_avg_price_1500            0.253864
green_part_2000                0.676771
cafe_sum_2000_min_price_avg    0.275928
cafe_sum_2000_max_price_avg    0.250875
cafe_avg_price_2000            0.259165
cafe_sum_3000_min_price_avg    0.286584
cafe_sum_3000_max_price_avg    0.269686
cafe_avg_price_3000            0.275521
prom_part_5000                 0.549153
cafe_sum_5000_min_price_avg    0.196017
cafe_sum_5000_max_price_avg    0.182153
cafe_avg_price_5000            0.187165
```

```
dtype: float64
```

[96]:
```
# Create an imputer
imputer = KNNImputer(n_neighbors=10)

# Use imputer to fill missing values.
surroundings.loc[:,surr_na_cols] = imputer.
↪fit_transform(surroundings[surr_na_cols])
```

[97]:
```
# Check if knn imputer worked
np.any(surroundings.isna().sum() > 0)
```

[97]:
```
False
```

[98]:
```
# Update surroundings data frame
groups_dfs['surroundings'] = surroundings
```

Education

[99]:
```
education = groups_dfs['education']
education.isna().sum()
```

[99]:
```
preschool_quota                          8284
preschool_education_centers_raion           0
school_quota                             8280
school_education_centers_raion              0
school_education_centers_top_20_raion       0
university_top_20_raion                     0
sport_objects_raion                         0
additional_education_raion                  0
dtype: int64
```

each of the columns with missing values give an information about the number of seats.. so we decided to fill them with -1 to indicate there's no data.

[100]:
```
education.loc[:, 'preschool_quota'] = education['preschool_quota'].fillna(-1)
education.loc[:, 'school_quota'] = education['school_quota'].fillna(-1)
```

[101]:
```
groups_dfs['education'] = education
```

Facilities

[102]:
```
faci = groups_dfs['facilities']
faci.isna().sum()
```

[102]:
```
hospital_beds_raion          17859
healthcare_centers_raion         0
shopping_centers_raion           0
office_raion                     0
thermal_power_plant_raion        0
```

```
incineration_raion              0
oil_chemistry_raion             0
radiation_raion                 0
railroad_terminal_raion         0
big_market_raion                0
nuclear_reactor_raion           0
detention_facility_raion        0
dtype: int64
```

[103]:
```
# Again a feature that indicates a count... can be fillied with -1 - no data.

faci.loc[:,'hospital_beds_raion'] = faci['hospital_beds_raion'].fillna(-1)
```

[104]:
```
groups_dfs['facilities'] = faci
```

Cleaning the data

[105]:
```
# Check if we missed any missing value:
group_na_indicators = []

for group in groups_dfs:
    indicator = np.any(groups_dfs.get(group).isna().sum() > 0)
    group_na_indicators.append((group, indicator))

group_na_indicators
```

[105]:
```
[('areas', False),
 ('buildings', False),
 ('demographics', False),
 ('distances', False),
 ('education', False),
 ('facilities', False),
 ('interior', True),
 ('surroundings', False)]
```

[106]:
```
nas = full_data.isna().sum()
# Change np.nan values into -1 so they could be replaced with the real value␣
 ↪from groups dfs
full_data.loc[:, nas[nas > 0].index] = train.loc[:, nas[nas > 0].index].
 ↪replace(np.nan, -999)

for group in groups_dfs:
    df = groups_dfs.get(group)
    # Insert clean columns to df.
    full_data.loc[df.index, df.columns] = df
```

[107]:
```
full_data = full_data.replace(-999,np.nan)
```

[108]:
```python
# get columns with missing values - supposed to be the distances columns only
x = full_data.isna().sum()
print(x[x>0])
```

```
price_doc    7662
dtype: int64
```

[109]:
```python
# full_data.to_csv("~/desktop/cleand_data.csv", index=False)
```

4.3 Loading Cleaned Data Frame

[4]:
```python
import requests

# Paths to text files from github
paths = ['https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/areas.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/buildings.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/demographics.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/distances.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/education.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/facilities.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/interior.txt',
         'https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/main/
↪text_files/surroundings.txt']

# Groups
groups = {
    'areas':[],
    'buildings':[],
    'demographics':[],
    'distances':[],
    'education':[],
    'facilities':[],
    'interior': [],
    'surroundings':[]
}

keys = list(groups.keys())

for i in range(len(paths)):
    # Http request to get the file data
```

```
    response = requests.get(paths[i])
    # read File lines
    lines = response.text.splitlines()
    for line in lines:
        # Split by \t
        col = line.split('\t')[0]
        # Interior file has : in it.
        if keys[i] == 'interior':
            col = col.split(':')[0]
        groups[keys[i]].append(col)
```

[5]:
```
full_data = pd.read_csv("https://raw.githubusercontent.com/LidorErez98/
 ↪Sberbank_ML/main/cleand_data.csv")
```

[6]:
```
sanity = full_data.isna().sum()
sanity[sanity > 0]
```

[6]:
```
price_doc    7662
dtype: int64
```

[7]:
```
groups_dfs = {group:full_data[full_data.columns[np.where(full_data.columns.
 ↪isin(groups[group]))]] for group in groups}
```

4.4 Feature Engineering

[8]:
```
def create_time_features(data):
    data['year'] = pd.to_datetime(data['timestamp']).dt.year
    data['month'] = pd.to_datetime(data['timestamp']).dt.month
    data['day'] = pd.to_datetime(data['timestamp']).dt.day
    data['day_of_week'] = pd.to_datetime(data['timestamp']).dt.dayofweek
    data['week_of_year'] = pd.to_datetime(data['timestamp']).dt.days_in_month
    data['quarter'] = pd.to_datetime(data['timestamp']).dt.quarter

    # Part 5
    data['monthyear'] = data['year']*100 + data['month']
    data['weekyear'] = data['year']*100 + data['week_of_year']

    month_year_counts = data['monthyear'].value_counts().to_dict()
    data['monthyear_count'] = data['monthyear'].map(month_year_counts)

    week_year_counts = data['weekyear'].value_counts().to_dict()
    data['weekyear_count'] = data['weekyear'].map(week_year_counts)
```

[9]:
```
full_data['timestamp'] = pd.to_datetime(full_data['timestamp'])
create_time_features(full_data)
full_data['log_price'] = np.log(full_data['price_doc'] + 1)
full_data['building_age'] = full_data['year'] - full_data['build_year']
```

```
full_data['price_sq'] = full_data['price_doc']/full_data['full_sq']
```

[10]:
```
dens_columns = [col + '_dens' for col in groups_dfs['demographics'].columns]
```

[11]:
```
full_data[dens_columns] = full_data[groups_dfs['demographics'].columns].
 ↪apply(lambda x:x/full_data['area_m'])
```

[12]:
```
groups_dfs['demographics'].loc[:,dens_columns] = full_data[dens_columns]
```

[13]:
```
full_data["ratio_life_sq_full_sq"] = full_data["life_sq"] / full_data["full_sq"]
```

[14]:
```
# adding ratio_life_sq_full_sq to interior group
groups_dfs['interior'].loc[:,'ratio_life_sq_full_sq'] =␣
 ↪full_data['ratio_life_sq_full_sq']
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/198719290.py:2:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[15]:
```
#kitchen ratio from the life sq
full_data["ratio_kitch_sq_full_sq"] = full_data["kitch_sq"] /␣
 ↪full_data["full_sq"]
```

[16]:
```
# adding ratio_kitch_sq_life_sq to interior group
groups_dfs['interior'].loc[:,'ratio_kitch_sq_full_sq'] =␣
 ↪full_data['ratio_kitch_sq_full_sq']
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/975391471.py:2:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[17]:
```
# calculate the room size
full_data["room_size"] = (full_data["life_sq"] - full_data['kitch_sq']) /␣
 ↪full_data["num_room"]
```

[18]:
```
# adding room size to interior group
groups_dfs['interior'].loc[:, 'room_size'] = full_data['room_size']
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/4252726323.py:2
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[19]:
```
full_data['extra_sq'] = full_data['full_sq'] - full_data['life_sq']
```

[20]:
```
full_data['area_diff'] = full_data['full_sq'] - full_data['kitch_sq'] # Part 5
```

[21]:
```
# adding extra sq to interior group
groups_dfs['interior'].loc[:,'extra_sq'] = full_data['extra_sq']
groups_dfs['interior'].loc[:,'area_diff'] = full_data['area_diff']
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/2361000648.py:2
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/2361000648.py:3
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[22]:
```
# calculate the floor ratio
full_data['floor_ratio'] = full_data['floor'] / full_data['max_floor']
```

[23]:
```
# adding floor ratio to interior group
groups_dfs['interior'].loc[:,'floor_ratio'] = full_data['floor_ratio']
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3316551521.py:2
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[24]:
```
# num of floor from top
full_data['floor_from_top'] = full_data['max_floor'] - full_data['floor']
```

[25]:
```
# adding floor from top to interior group
groups_dfs['interior'].loc[:,'floor_from_top'] = full_data['floor_from_top']
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/757007344.py:2:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[26]:
```
percapita = [c + '_percapita' for c in groups_dfs['education'].columns]
full_data[percapita] = full_data[groups_dfs['education'].columns].apply(lambda␣
 ↪edu: edu/full_data['raion_popul'])
```

[27]:
```
groups_dfs['education'].loc[:,percapita] = full_data[percapita]
```

[28]:
```
minmaxscaler = MinMaxScaler()
cafe_count = full_data.filter(like="cafe_count")
minmaxscaler.fit_transform(cafe_count)
# replace with cafe count columns in full_data
full_data.loc[:,cafe_count.columns] = minmaxscaler.fit_transform(cafe_count)
```

[29]:
```
groups_dfs['surroundings'].loc[:,groups_dfs['surroundings'].
 ↪filter(like="cafe_count").columns] = full_data.filter(like="cafe_count")
```

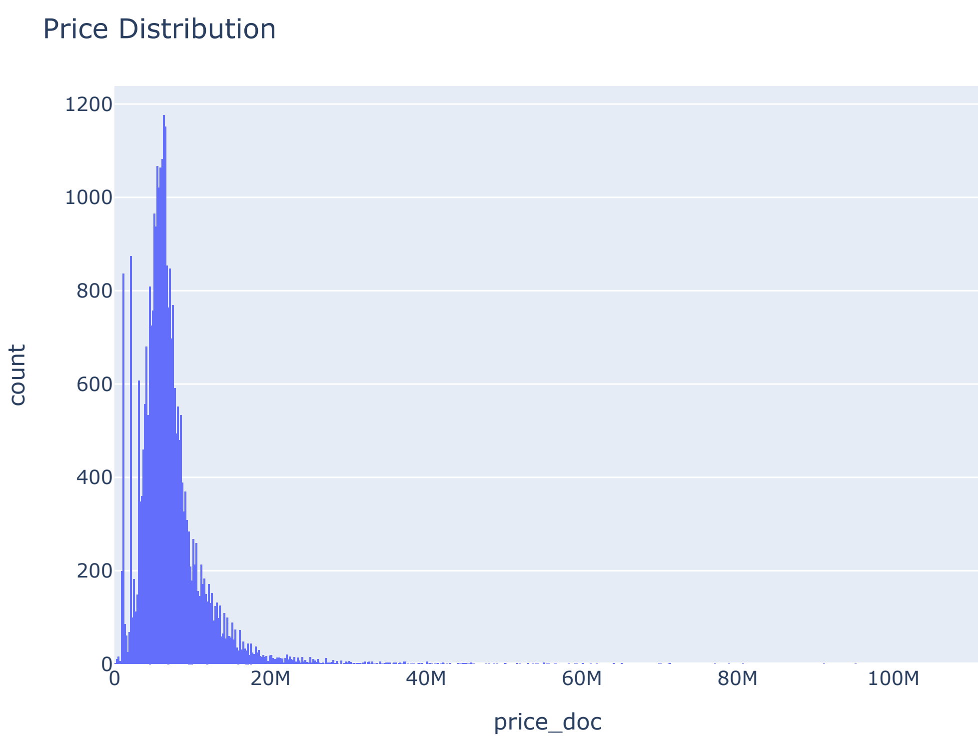
[30]:
```
full_data.drop(columns = [c.replace('_dens','') for c in dens_columns],␣
 ↪inplace=True)
full_data.drop(columns = [c.replace('_percapita','') for c in percapita],␣
 ↪inplace=True)
```

```
groups_dfs['demographics'] = groups_dfs['demographics'].drop(columns = [c.
    ↪replace('_dens','') for c in dens_columns])
groups_dfs['education'] = groups_dfs['education'].drop(columns = [c.
    ↪replace('_percapita','') for c in percapita])
```

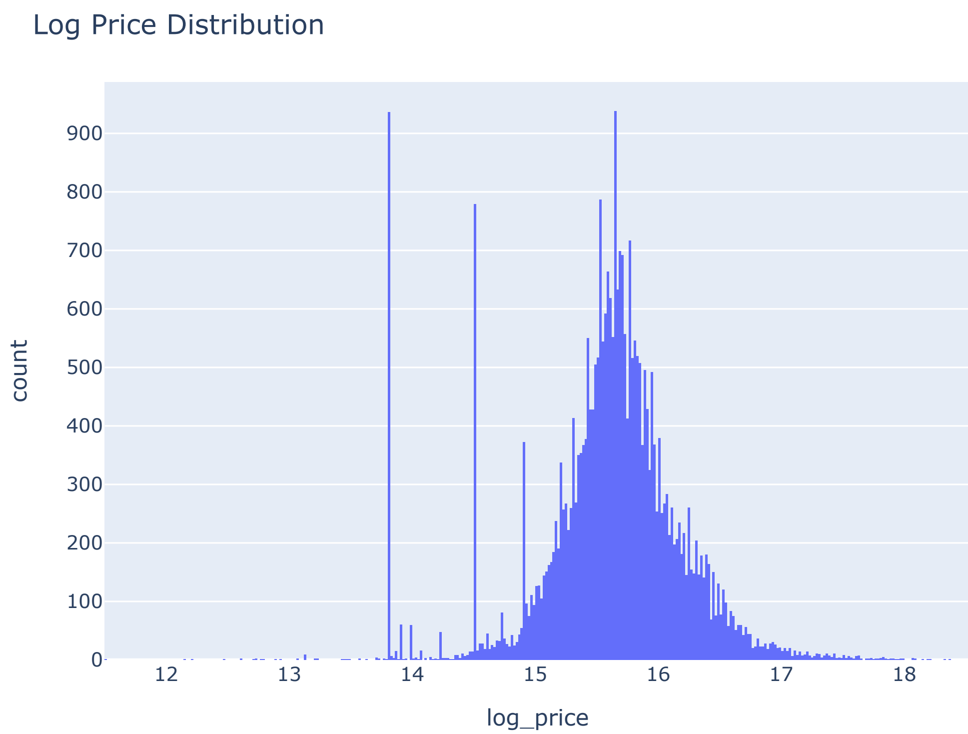
4.5 Explanatory Data Analysis

[31]:
```
import plotly.express as px
```

[32]:
```
fig = px.histogram(full_data, x='price_doc')
fig.update_layout(title = 'Price Distribution')
fig.show()
```

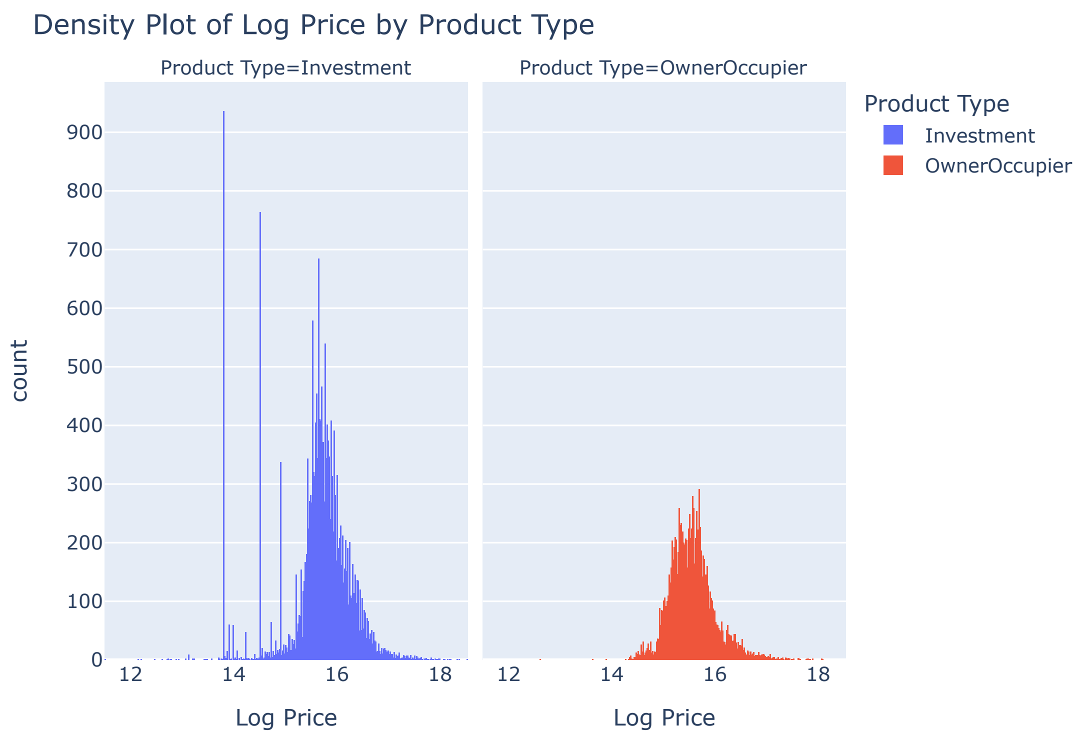


[33]:
```
fig = px.histogram(full_data, x='log_price')
fig.update_layout(title = 'Log Price Distribution')
fig.show()
```



Log price might be a better target variable than the actual price.

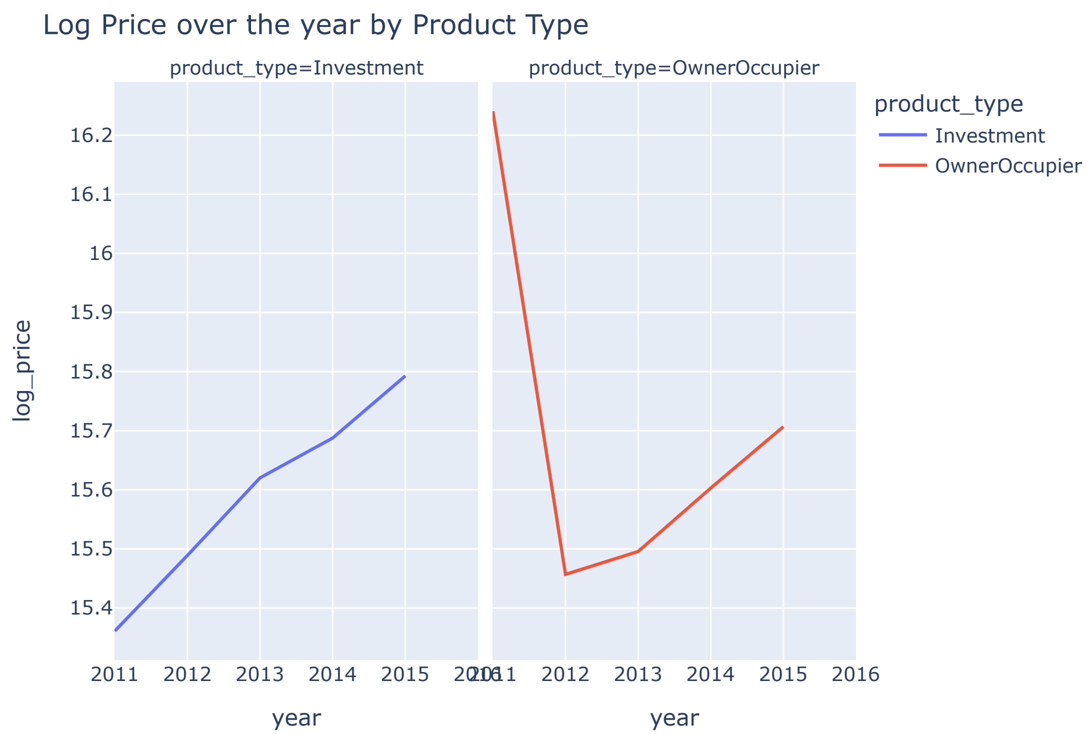
[34]:
```
fig = px.histogram(full_data, x='log_price',␣
↪facet_col='product_type',labels={'log_price':'Log Price', 'product_type':
↪'Product Type'}, color='product_type')
fig.update_layout(title = 'Density Plot of Log Price by Product Type')
fig.show()
```



[35]:
```
fig = px.box(full_data, x='log_price', y='product_type',labels={'log_price':
 ↪'Log Price', 'product_type':'Product Type'}, color='product_type')
fig.update_layout(title = 'Density Plot of Log Price by Product Type')
fig.show()
```

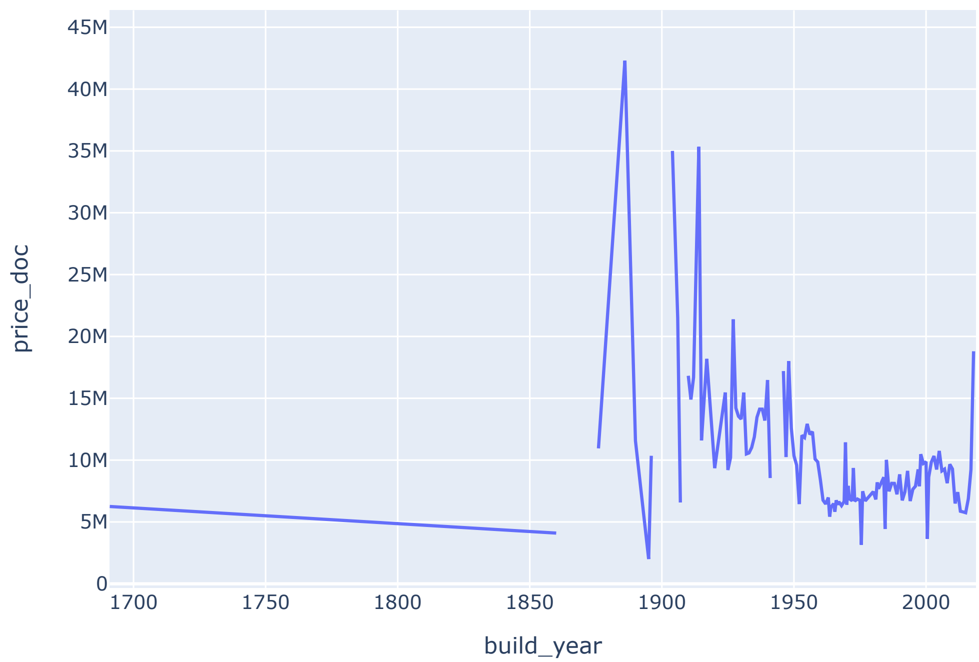
There are many outliers in the log price column this might have an effect on our model predictions.

[36]:
```
cp = full_data.copy()
cp['year'] = cp['timestamp'].dt.year
temp = (cp.groupby(['year','product_type'])['log_price'].mean()).reset_index()
temp['year'] = temp['year'].apply(lambda x: str(x))
year_order = sorted(temp['year'].unique())
fig = px.line(temp, x='year', y='log_price', facet_col = 'product_type',␣
 ↪category_orders={'year': year_order}, color='product_type')
fig.update_layout(title = 'Log Price over the year by Product Type')
fig.show()
```

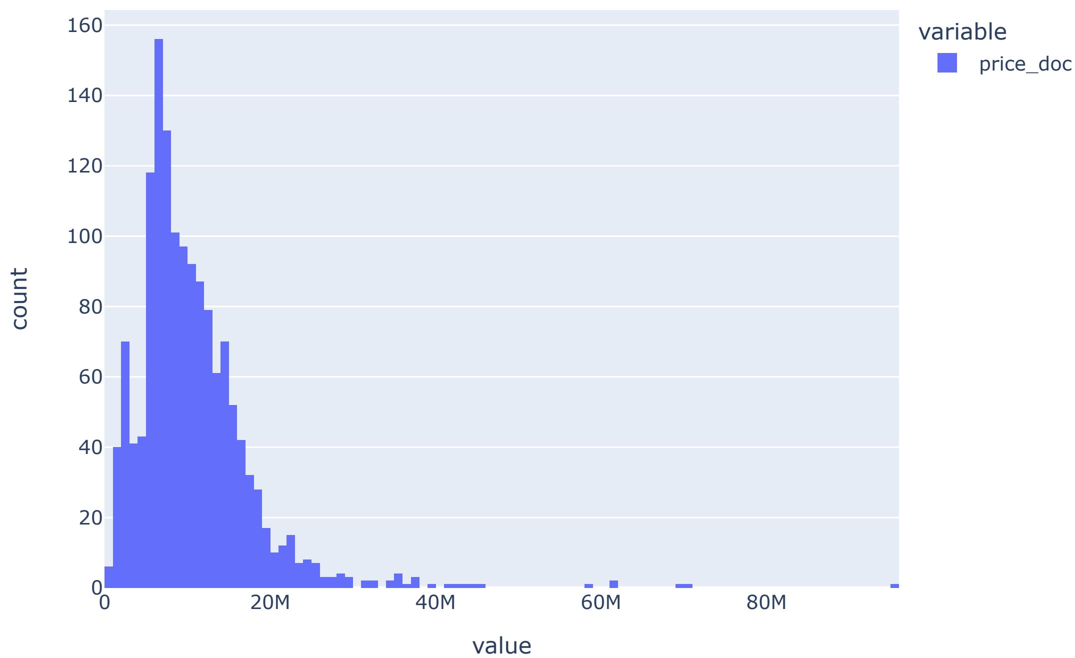


Prices different behavior among product types indicate that we might need to train separated models for each product type.

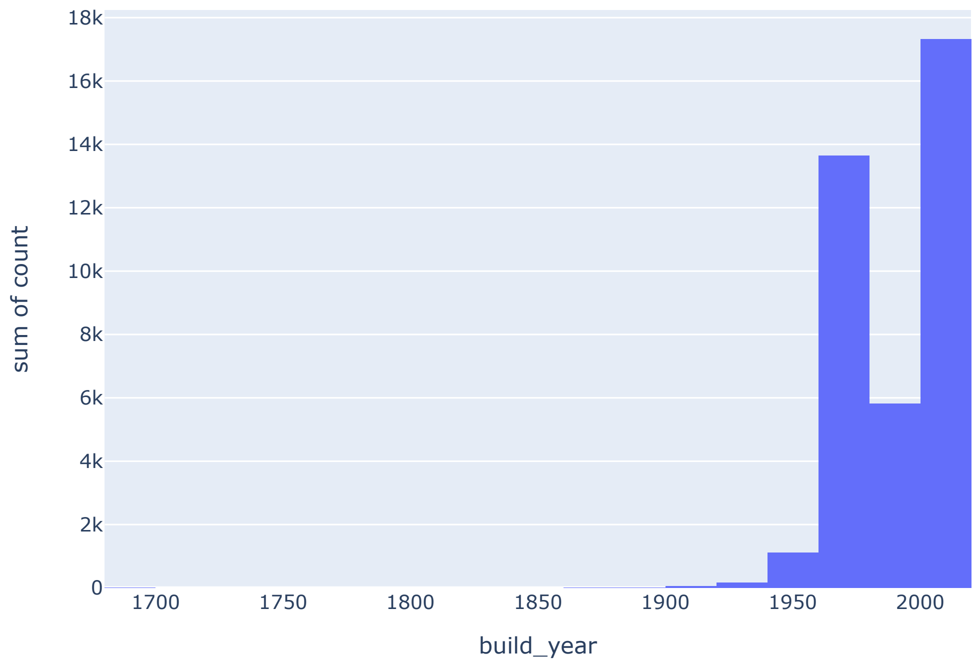
[37]:
```
temp = (full_data.groupby("build_year")['price_doc'].mean()).reset_index()
fig = px.line(temp, x='build_year',y='price_doc')
fig.show()
```



[38]:
```
temp = full_data.loc[full_data.build_year <= 1960]
fig = px.histogram(temp['price_doc'])
fig.show()
```



[39]:
```
temp = (full_data['build_year'].value_counts()).reset_index()
fig = px.histogram(temp,x='build_year',y='count')
fig.show()
```

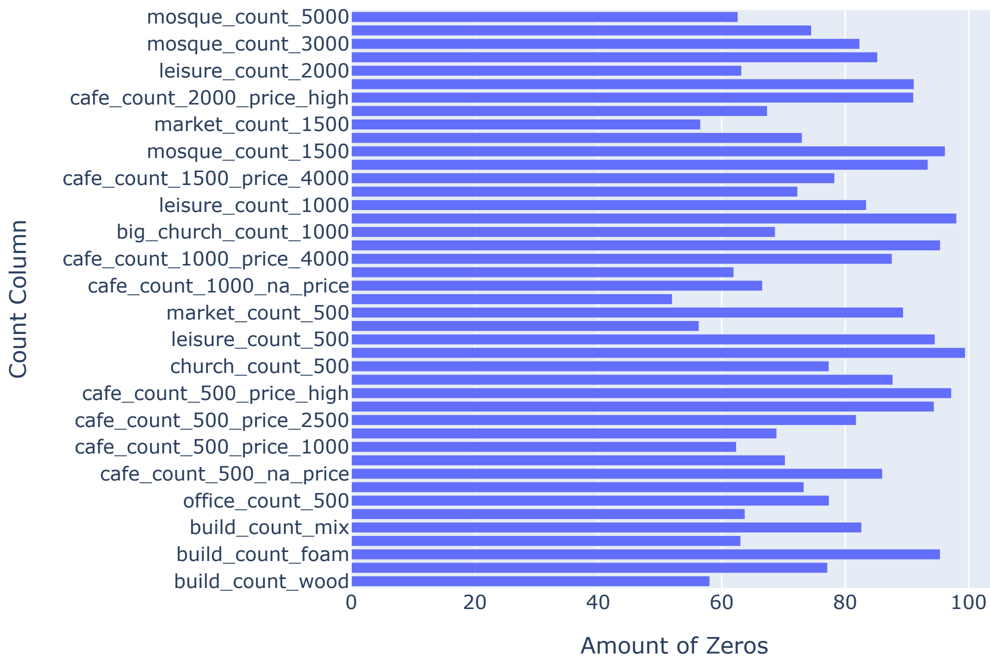


We found that for old building prices are higher when the build year is less than 1960 so we will add a binary feature to describe if a house is "vintage"

[40]:
```
full_data.loc[:,'is_vintage'] = full_data['build_year'].apply(lambda year:␣
↪(year <= 1960)).astype(int)
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/2529656510.py:1
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`
```

[41]:
```
# Checking for sparsity in count data:

zeroes_pct = np.round((full_data.filter(like='count') == 0).mean()*100,2).
↪reset_index().rename(columns = {0:'Amount of Zeros', 'index':'Count Column'})

px.bar(zeroes_pct[zeroes_pct['Amount of Zeros'] >= 50], x='Amount of␣
↪Zeros',y='Count Column')
```



We checked the amount of zeros in our count columns to figure out how sparse they are. We then created a bar plot of the count columns that have at least 50% zeros. To cope with this issue we decided to turn each of the count column that is shown on the plot to a binary column which will indicate the absence of data

[42]:
```
sparse_count_columns = zeroes_pct[zeroes_pct['Amount of Zeros'] >= 50]['Count␣
↪Column'].values
sparse_count_columns
```

[42]:
```
array(['build_count_wood', 'build_count_frame', 'build_count_foam',
       'build_count_slag', 'build_count_mix', 'build_count_before_1920',
       'office_count_500', 'trc_count_500', 'cafe_count_500_na_price',
       'cafe_count_500_price_500', 'cafe_count_500_price_1000',
       'cafe_count_500_price_1500', 'cafe_count_500_price_2500',
       'cafe_count_500_price_4000', 'cafe_count_500_price_high',
       'big_church_count_500', 'church_count_500', 'mosque_count_500',
       'leisure_count_500', 'sport_count_500', 'market_count_500',
       'office_count_1000', 'cafe_count_1000_na_price',
       'cafe_count_1000_price_2500', 'cafe_count_1000_price_4000',
       'cafe_count_1000_price_high', 'big_church_count_1000',
       'mosque_count_1000', 'leisure_count_1000', 'market_count_1000',
       'cafe_count_1500_price_4000', 'cafe_count_1500_price_high',
```

```
        'mosque_count_1500', 'leisure_count_1500', 'market_count_1500',
        'cafe_count_2000_price_4000', 'cafe_count_2000_price_high',
        'mosque_count_2000', 'leisure_count_2000',
        'cafe_count_3000_price_high', 'mosque_count_3000',
        'cafe_count_5000_price_high', 'mosque_count_5000'], dtype=object)
```

[43]:
```
for col in sparse_count_columns:
    full_data.loc[:, col + '_binary'] = ((full_data[col] > 0)).astype(int)

full_data.filter(like='_binary')
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
```

```
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
```

```
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
```

```
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
```

```
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
```

```
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
```

```
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
```

```
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`

/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/3714908506.py:2
: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling
`frame.insert` many times, which has poor performance.  Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame,
use `newframe = frame.copy()`
```

[43]:

```
       build_count_wood_binary  build_count_frame_binary  \
0                            0                         0
1                            1                         0
2                            0                         0
3                            1                         1
4                            0                         0
…                          …                       …
38128                        0                         0
38129                        0                         0
38130                        1                         1
38131                        0                         0
38132                        0                         0

       build_count_foam_binary  build_count_slag_binary  \
0                            0                        0
1                            0                        0
2                            0                        1
3                            0                        1
4                            0                        1
…                          …                      …
38128                        0                        0
38129                        0                        0
38130                        0                        1
38131                        0                        0
38132                        0                        0

       build_count_mix_binary  build_count_before_1920_binary  \
```

```
0                                0                                      0
1                                0                                      1
2                                0                                      1
3                                1                                      1
4                                1                                      1
…                            …                                      …
38128                            0                                      0
38129                            0                                      0
38130                            1                                      1
38131                            0                                      0
38132                            0                                      0

       office_count_500_binary  trc_count_500_binary  \
0                            0                     0
1                            0                     0
2                            0                     0
3                            0                     0
4                            1                     1
…                        …                     …
38128                        1                     1
38129                        0                     0
38130                        1                     1
38131                        0                     1
38132                        1                     1

       cafe_count_500_na_price_binary  cafe_count_500_price_500_binary  …  \
0                                   0                                0  …
1                                   0                                1  …
2                                   0                                0  …
3                                   0                                0  …
4                                   1                                1  …
…                               …                              …  …
38128                               0                                1  …
38129                               0                                0  …
38130                               1                                1  …
38131                               0                                1  …
38132                               0                                0  …

       leisure_count_1500_binary  market_count_1500_binary  \
0                              0                         1
1                              1                         0
2                              0                         1
3                              0                         1
4                              1                         1
…                          …                         …
38128                          1                         1
38129                          0                         0
```

```
38130                          1                          1
38131                          0                          1
38132                          0                          0

       cafe_count_2000_price_4000_binary  cafe_count_2000_price_high_binary  \
0                                      1                                  0
1                                      1                                  0
2                                      0                                  0
3                                      0                                  0
4                                      1                                  1
…                                    …                                …
38128                                  0                                  0
38129                                  0                                  0
38130                                  1                                  1
38131                                  1                                  0
38132                                  0                                  0

       mosque_count_2000_binary  leisure_count_2000_binary  \
0                             0                          0
1                             0                          1
2                             0                          0
3                             0                          0
4                             0                          1
…                           …                        …
38128                         0                          1
38129                         0                          0
38130                         1                          1
38131                         0                          0
38132                         1                          0

       cafe_count_3000_price_high_binary  mosque_count_3000_binary  \
0                                      0                         0
1                                      0                         0
2                                      0                         0
3                                      0                         0
4                                      1                         1
…                                    …                       …
38128                                  0                         1
38129                                  0                         0
38130                                  1                         1
38131                                  0                         0
38132                                  0                         1

       cafe_count_5000_price_high_binary  mosque_count_5000_binary
0                                      0                         1
1                                      0                         1
2                                      0                         0
```

3	1	0
4	1	1
…	…	…
38128	1	1
38129	0	0
38130	1	1
38131	0	0
38132	0	1

[38133 rows x 43 columns]

[44]:
```
full_data = full_data.drop(columns = sparse_count_columns)
```

[45]:
```
for group in groups_dfs:
    if np.any(groups_dfs[group].columns.isin(sparse_count_columns)):
        idx = np.where(groups_dfs[group].columns.isin(sparse_count_columns))
        cols_to_drop = groups_dfs[group].columns[idx]
        groups_dfs[group] = groups_dfs[group].drop(columns = cols_to_drop)
```

4.6 Macro CSV

[46]:
```
macro = pd.read_csv("https://raw.githubusercontent.com/LidorErez98/Sberbank_ML/
↪main/Data/macro.csv")
min_date, max_date = full_data['timestamp'].min(),full_data['timestamp'].max()
macro['date'] = pd.to_datetime(macro['timestamp'])
macro = macro[(macro['date'] >= min_date) & (macro['date'] <= max_date)]
macro['year'] = macro['date'].dt.year
macro['month'] = macro['date'].dt.month
macro['quarter'] = macro['date'].dt.quarter
```

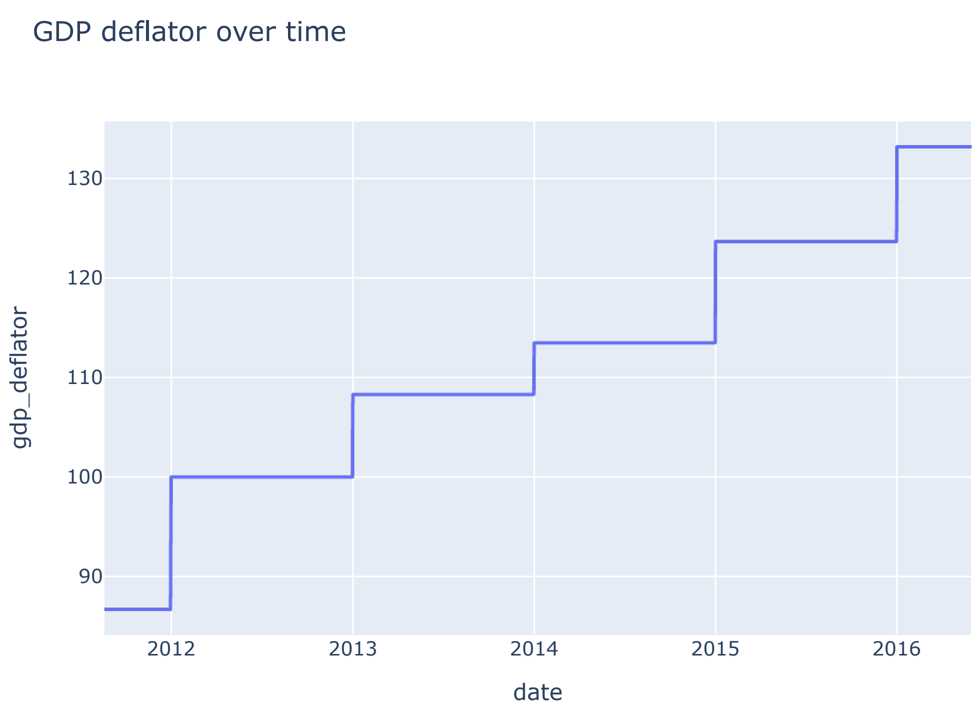
[47]:
```
# split to sub dataframes
inflation = macro[['date', 'quarter', 'year', 'month', 'cpi', 'ppi',␣
↪'gdp_deflator']]
gdp = macro[['date', 'quarter', 'year', 'month', 'gdp_quart',␣
↪'gdp_quart_growth', 'gdp_deflator', 'gdp_annual', 'gdp_annual_growth']]
# salary = macro[['date', 'quarter', 'year', 'month', 'salary',␣
↪'salary_growth', 'real_dispos_income_per_cap_growth', 'salary_growth']]
# mortgage = macro[['date', 'quarter', 'year', 'month', 'mortgage_rate',␣
↪'mortgage_growth', 'deposits_rate', 'deposits_growth']]
# investmeent = macro[['date', 'quarter', 'year', 'month',␣
↪'invest_fixed_assets', 'invest_fixed_assets_phys',␣
↪'profitable_enterpr_share', 'unprofitable_enterpr_share',␣
↪'share_own_revenues', 'overdue_wages_per_cap', 'fin_res_per_cap',␣
↪'invest_fixed_assets']]
```

```
# consumption = macro[['date', 'quarter', 'year', 'month', 'income_per_cap',␣
↪'real_dispos_income_per_cap_growth', 'salary', 'salary_growth',␣
↪'retail_trade_turnover', 'retail_trade_turnover_per_cap',␣
↪'retail_trade_turnover_growth', 'labor_force']]
# interest = macro[['date', 'quarter', 'year', 'month',  'deposits_rate',␣
↪'deposits_growth', 'mortgage_rate', 'mortgage_growth']]
# governmenr = macro[['date', 'quarter', 'year', 'month', 'balance_trade',␣
↪'balance_trade_growth', 'usdrub', 'eurrub', 'micex_rgbi_tr', 'micex']]
```

4.6.1 INFLATION

[48]:
```
inflation_df = inflation.copy()
# using plotly express to plot the data ovr time
def deflator_to_inflation_rate(deflator1, defltor0):
    return (deflator1 - defltor0)/defltor0
```

[49]:
```
fig = px.line(inflation_df, x='date', y='gdp_deflator', title='GDP deflator␣
↪over time')
fig.show()
print(inflation_df['gdp_deflator'].describe())
```




```
count     1746.000000
```

```
mean      111.344730
std        12.220722
min        86.721000
25%       100.000000
50%       113.465000
75%       123.661000
max       133.160000
Name: gdp_deflator, dtype: float64
```

Inflation Integrity

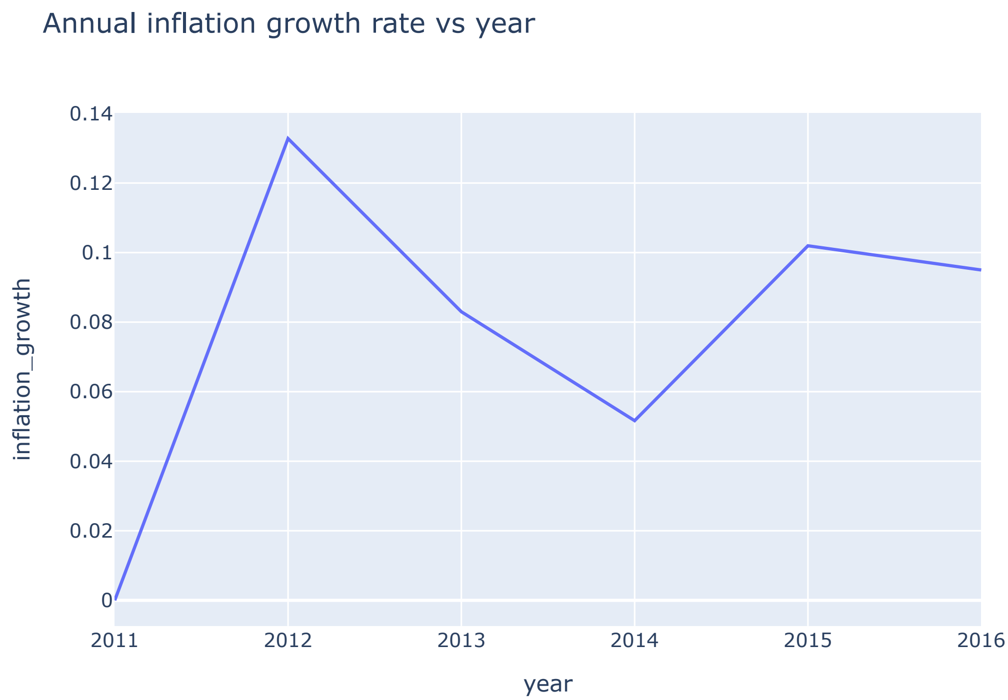
[50]:
```
# fix the gdp deflator of 2011 to 100 and the rest of them accordingly
fix = inflation_df[inflation_df['year'] == 2011]['gdp_deflator'].values[0] - 100
inflation_df['gdp_deflator'] = inflation_df['gdp_deflator'] - fix
inflation_df['inflation_from_8_2011'] = inflation_df['gdp_deflator'].
 ↪apply(lambda x: deflator_to_inflation_rate(x, 100))
```

[51]:
```
# for annual inflation growth rate we will use the inflation from 8/2011
annual_inflation_df = pd.DataFrame(inflation_df[['year', 'gdp_deflator',␣
 ↪'inflation_from_8_2011']].drop_duplicates(subset=['year']))
annual_inflation_df.set_index('year', inplace=True)
for i in range(2012, 2017):
    annual_inflation_df.loc[i, 'inflation_growth'] = annual_inflation_df.loc[i,␣
 ↪'inflation_from_8_2011'] - annual_inflation_df.loc[i-1,␣
 ↪'inflation_from_8_2011']
annual_inflation_df.loc[2011, 'inflation_growth'] = 0
annual_inflation_df
```

[51]:

year	gdp_deflator	inflation_from_8_2011	inflation_growth
2011	100.000	0.00000	0.00000
2012	113.279	0.13279	0.13279
2013	121.578	0.21578	0.08299
2014	126.744	0.26744	0.05166
2015	136.940	0.36940	0.10196
2016	146.439	0.46439	0.09499

[52]:
```
# annual inflation growth rate vs year ployline
fig = px.line(annual_inflation_df, x=annual_inflation_df.index,␣
 ↪y='inflation_growth', title='Annual inflation growth rate vs year')
fig.show()
```



[53]:

```
# add the inflation growth rate to the inflation dataframe
inflation_df.set_index('year', inplace=True)
inflation_df['annual_inflation_growth'] =␣
 ↪annual_inflation_df['inflation_growth']
inflation_df.reset_index(inplace=True)
inflation_df
```

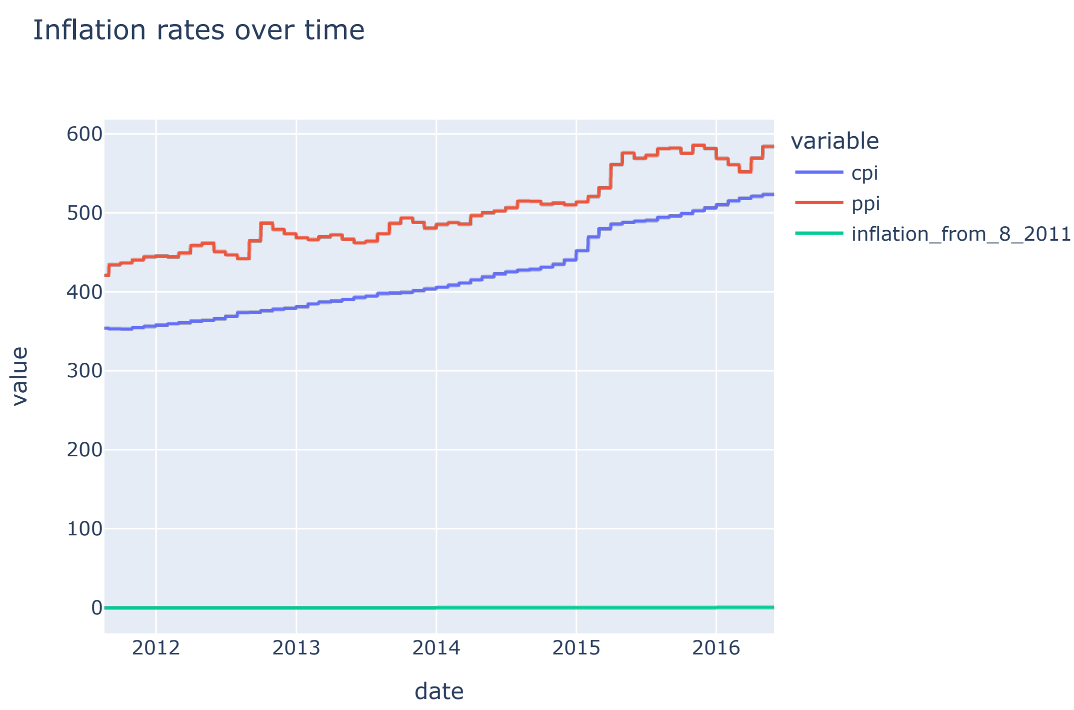
[53]:

```
      year        date  quarter  month    cpi    ppi  gdp_deflator  \
0     2011  2011-08-20        3      8  354.0  420.7       100.000
1     2011  2011-08-21        3      8  354.0  420.7       100.000
2     2011  2011-08-22        3      8  354.0  420.7       100.000
3     2011  2011-08-23        3      8  354.0  420.7       100.000
4     2011  2011-08-24        3      8  354.0  420.7       100.000
…     …          …        …      …      …      …            …
1741  2016  2016-05-26        2      5  523.2  584.0       146.439
1742  2016  2016-05-27        2      5  523.2  584.0       146.439
1743  2016  2016-05-28        2      5  523.2  584.0       146.439
1744  2016  2016-05-29        2      5  523.2  584.0       146.439
1745  2016  2016-05-30        2      5  523.2  584.0       146.439

      inflation_from_8_2011  annual_inflation_growth
```

```
0                   0.00000                  0.00000
1                   0.00000                  0.00000
2                   0.00000                  0.00000
3                   0.00000                  0.00000
4                   0.00000                  0.00000
…                      …                       …
1741                0.46439                  0.09499
1742                0.46439                  0.09499
1743                0.46439                  0.09499
1744                0.46439                  0.09499
1745                0.46439                  0.09499

[1746 rows x 9 columns]
```

cpi and ppi correlation with inflation rate

[54]:
```
# plot 3 lines together
fig = px.line(inflation_df, x='date', y=['cpi', 'ppi',␣
↪'inflation_from_8_2011'], title='Inflation rates over time',␣
↪color='variable')
fig.show()
```

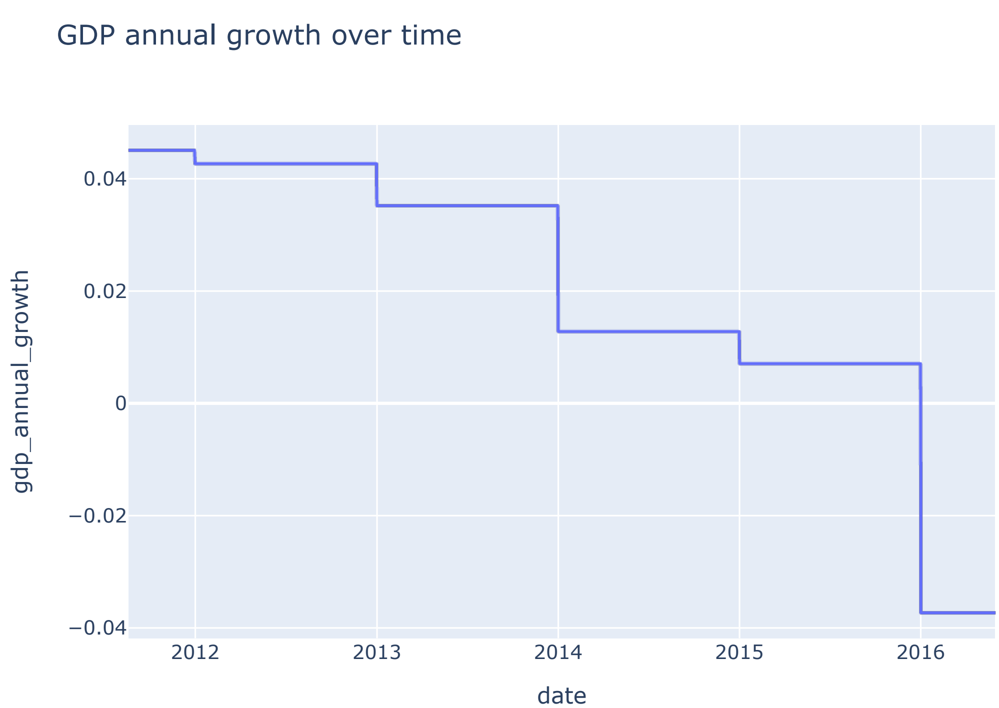


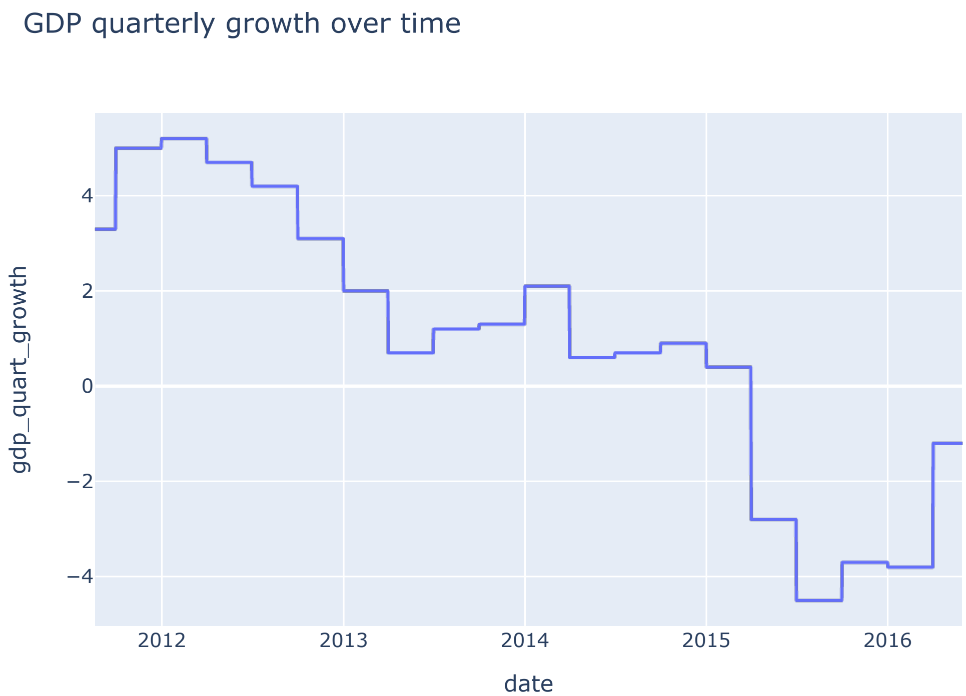
[55]:
```
inflation_df['cpi'].values[0]
```

[55]: 354.0

[56]:
```
# turn cpi and ppi to growths frrom 8 2011
cpifix = 100 - inflation_df['cpi'].values[0]
ppifix = 100 - inflation_df['ppi'].values[0]
inflation_df['cpi'] = inflation_df['cpi'] + cpifix
inflation_df['ppi'] = inflation_df['ppi'] + ppifix
inflation_df['cpi_growth'] = inflation_df['cpi'].apply(lambda x:␣
↪deflator_to_inflation_rate(x, 100))
inflation_df['ppi_growth'] = inflation_df['ppi'].apply(lambda x:␣
↪deflator_to_inflation_rate(x, 100))
```

[57]:
```
gdp_dfd = gdp.copy(deep=True)
# check the gdp growth over time( anuualy vs quarterly) plotly express
fig = px.line(gdp_dfd, x='date', y='gdp_annual_growth', title='GDP annual␣
↪growth over time')
fig.show()
fig = px.line(gdp_dfd, x='date', y='gdp_quart_growth', title='GDP quarterly␣
↪growth over time')
fig.show()
```





[58]:
```
# create a df with the quart gdp growth for each quarter in each year no␣
↪duplicates
gdp_dfd_quart = pd.DataFrame(gdp_dfd[['year', 'quarter', 'gdp_quart_growth']].
↪drop_duplicates())
# setting yaer, quarter as index, and divide by 100 to get the percentage
gdp_dfd_quart['gdp_quart_growth'] = gdp_dfd_quart['gdp_quart_growth'].
↪apply(lambda x: x/100)
gdp_dfd_quart.set_index(['year', 'quarter'], inplace=True)
gdp_dfd_quart['gdp_quart_growth_since_2011'] =␣
↪gdp_dfd_quart['gdp_quart_growth'].cumsum()
gdp_dfd_quart
```

[58]:

year	quarter	gdp_quart_growth	gdp_quart_growth_since_2011
2011	3	0.033	0.033
	4	0.050	0.083
2012	1	0.052	0.135
	2	0.047	0.182
	3	0.042	0.224
	4	0.031	0.255
2013	1	0.020	0.275

```
     2             0.007                   0.282
     3             0.012                   0.294
     4             0.013                   0.307
2014 1             0.021                   0.328
     2             0.006                   0.334
     3             0.007                   0.341
     4             0.009                   0.350
2015 1             0.004                   0.354
     2            -0.028                   0.326
     3            -0.045                   0.281
     4            -0.037                   0.244
2016 1            -0.038                   0.206
     2            -0.012                   0.194
```

[59]:

```
# adding the gdp growth sum to relevant rows in the gdp df
gdp_dfd.set_index(['year', 'quarter'], inplace=True)
gdp_dfd['gdp_quart_growth_since_2011'] =␣
 ↪gdp_dfd_quart['gdp_quart_growth_since_2011']
gdp_dfd.reset_index(inplace=True)
gdp_dfd.head(17)
```

[59]:

```
    year  quarter        date  month  gdp_quart  gdp_quart_growth  \
0   2011        3  2011-08-20      8    14313.7               3.3
1   2011        3  2011-08-21      8    14313.7               3.3
2   2011        3  2011-08-22      8    14313.7               3.3
3   2011        3  2011-08-23      8    14313.7               3.3
4   2011        3  2011-08-24      8    14313.7               3.3
5   2011        3  2011-08-25      8    14313.7               3.3
6   2011        3  2011-08-26      8    14313.7               3.3
7   2011        3  2011-08-27      8    14313.7               3.3
8   2011        3  2011-08-28      8    14313.7               3.3
9   2011        3  2011-08-29      8    14313.7               3.3
10  2011        3  2011-08-30      8    14313.7               3.3
11  2011        3  2011-08-31      8    14313.7               3.3
12  2011        3  2011-09-01      9    14313.7               3.3
13  2011        3  2011-09-02      9    14313.7               3.3
14  2011        3  2011-09-03      9    14313.7               3.3
15  2011        3  2011-09-04      9    14313.7               3.3
16  2011        3  2011-09-05      9    14313.7               3.3

    gdp_deflator  gdp_annual  gdp_annual_growth  gdp_quart_growth_since_2011
0         86.721     46308.5           0.045037                        0.033
1         86.721     46308.5           0.045037                        0.033
2         86.721     46308.5           0.045037                        0.033
3         86.721     46308.5           0.045037                        0.033
4         86.721     46308.5           0.045037                        0.033
5         86.721     46308.5           0.045037                        0.033
```

6	86.721	46308.5	0.045037	0.033
7	86.721	46308.5	0.045037	0.033
8	86.721	46308.5	0.045037	0.033
9	86.721	46308.5	0.045037	0.033
10	86.721	46308.5	0.045037	0.033
11	86.721	46308.5	0.045037	0.033
12	86.721	46308.5	0.045037	0.033
13	86.721	46308.5	0.045037	0.033
14	86.721	46308.5	0.045037	0.033
15	86.721	46308.5	0.045037	0.033
16	86.721	46308.5	0.045037	0.033

[60]:
```
gdp_dfd_year = pd.DataFrame(gdp_dfd[['year', 'gdp_annual_growth']].
 ↪drop_duplicates())
gdp_dfd_year.set_index('year', inplace=True)
gdp_dfd_year['gdp_annual_growth_since_2011'] =␣
 ↪gdp_dfd_year['gdp_annual_growth'].cumsum()
gdp_dfd_year
```

[60]:

	gdp_annual_growth	gdp_annual_growth_since_2011
year		
2011	0.045037	0.045037
2012	0.042644	0.087681
2013	0.035179	0.122859
2014	0.012795	0.135654
2015	0.007065	0.142719
2016	-0.037267	0.105452

[61]:
```
# insert the new feature to gdp dfd
gdp_dfd.set_index('year', inplace=True)
gdp_dfd['gdp_annual_growth_since_2011'] =␣
 ↪gdp_dfd_year['gdp_annual_growth_since_2011']
gdp_dfd.reset_index(inplace=True)
gdp_dfd['gdp_quart_growth'] = gdp_dfd['gdp_quart_growth'].apply(lambda x: x/100)
```

[62]:
```
#adjusting the gdp growth rates so they will start from 0 in 2011
gdp_dfd['gdp_annual_growth'] = gdp_dfd['gdp_annual_growth'] -␣
 ↪gdp_dfd[gdp_dfd['year'] == 2011]['gdp_annual_growth'].values[0]
gdp_dfd['gdp_annual_growth_since_2011'] =␣
 ↪gdp_dfd['gdp_annual_growth_since_2011'] - gdp_dfd[gdp_dfd['year'] ==␣
 ↪2011]['gdp_annual_growth_since_2011'].values[0]
gdp_dfd['gdp_quart_growth'] = gdp_dfd['gdp_quart_growth'] -␣
 ↪gdp_dfd[gdp_dfd['year'] == 2011]['gdp_quart_growth'].values[0]
gdp_dfd['gdp_quart_growth_since_2011'] = gdp_dfd['gdp_quart_growth_since_2011']␣
 ↪- gdp_dfd[gdp_dfd['year'] == 2011]['gdp_quart_growth_since_2011'].values[0]
```

[63]:
```
gdp_dfd.head(17)
```

[63]:

```
    year  quarter        date  month  gdp_quart  gdp_quart_growth  \
0   2011        3  2011-08-20      8    14313.7               0.0
1   2011        3  2011-08-21      8    14313.7               0.0
2   2011        3  2011-08-22      8    14313.7               0.0
3   2011        3  2011-08-23      8    14313.7               0.0
4   2011        3  2011-08-24      8    14313.7               0.0
5   2011        3  2011-08-25      8    14313.7               0.0
6   2011        3  2011-08-26      8    14313.7               0.0
7   2011        3  2011-08-27      8    14313.7               0.0
8   2011        3  2011-08-28      8    14313.7               0.0
9   2011        3  2011-08-29      8    14313.7               0.0
10  2011        3  2011-08-30      8    14313.7               0.0
11  2011        3  2011-08-31      8    14313.7               0.0
12  2011        3  2011-09-01      9    14313.7               0.0
13  2011        3  2011-09-02      9    14313.7               0.0
14  2011        3  2011-09-03      9    14313.7               0.0
15  2011        3  2011-09-04      9    14313.7               0.0
16  2011        3  2011-09-05      9    14313.7               0.0

    gdp_deflator  gdp_annual  gdp_annual_growth  gdp_quart_growth_since_2011  \
0         86.721     46308.5                0.0                          0.0
1         86.721     46308.5                0.0                          0.0
2         86.721     46308.5                0.0                          0.0
3         86.721     46308.5                0.0                          0.0
4         86.721     46308.5                0.0                          0.0
5         86.721     46308.5                0.0                          0.0
6         86.721     46308.5                0.0                          0.0
7         86.721     46308.5                0.0                          0.0
8         86.721     46308.5                0.0                          0.0
9         86.721     46308.5                0.0                          0.0
10        86.721     46308.5                0.0                          0.0
11        86.721     46308.5                0.0                          0.0
12        86.721     46308.5                0.0                          0.0
13        86.721     46308.5                0.0                          0.0
14        86.721     46308.5                0.0                          0.0
15        86.721     46308.5                0.0                          0.0
16        86.721     46308.5                0.0                          0.0

    gdp_annual_growth_since_2011
0                            0.0
1                            0.0
2                            0.0
3                            0.0
4                            0.0
5                            0.0
6                            0.0
7                            0.0
```

```
8                            0.0
9                            0.0
10                           0.0
11                           0.0
12                           0.0
13                           0.0
14                           0.0
15                           0.0
16                           0.0
```

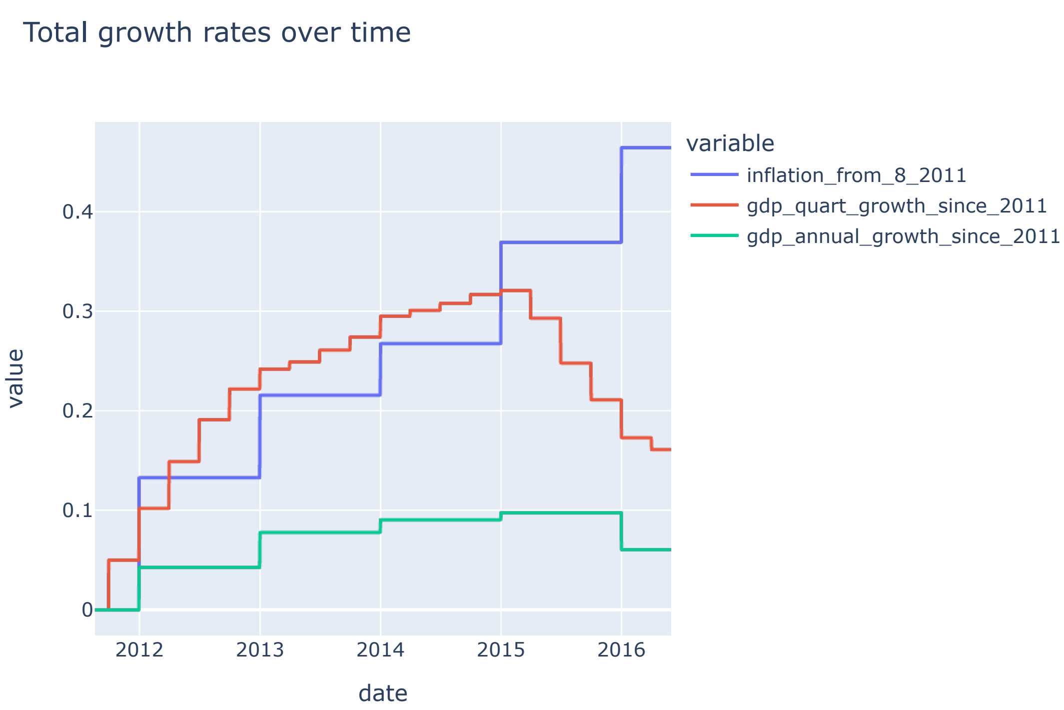
merge inflation gdp

[64]:
```
# merge new features together to one df
inf_gdp_merged = inflation_df.merge(gdp_dfd, on=['date', 'quarter', 'year',␣
 ↪'month'])
# date feature and growth features only
inf_gdp_merged_total_growth = inf_gdp_merged[['date', 'quarter', 'year',␣
 ↪'month', 'inflation_from_8_2011', 'gdp_quart_growth_since_2011',␣
 ↪'gdp_annual_growth_since_2011']]
inf_gdp_merged_period_growth = inf_gdp_merged[['date', 'quarter', 'year',␣
 ↪'month', 'annual_inflation_growth', 'gdp_quart_growth', 'gdp_annual_growth']]
inf_gdp_merged_total_growth
```

[64]:
```
           date  quarter  year  month  inflation_from_8_2011  \
0    2011-08-20        3  2011      8                0.00000
1    2011-08-21        3  2011      8                0.00000
2    2011-08-22        3  2011      8                0.00000
3    2011-08-23        3  2011      8                0.00000
4    2011-08-24        3  2011      8                0.00000
…           …        …    …      …                      …
1741 2016-05-26        2  2016      5                0.46439
1742 2016-05-27        2  2016      5                0.46439
1743 2016-05-28        2  2016      5                0.46439
1744 2016-05-29        2  2016      5                0.46439
1745 2016-05-30        2  2016      5                0.46439

      gdp_quart_growth_since_2011  gdp_annual_growth_since_2011
0                           0.000                      0.000000
1                           0.000                      0.000000
2                           0.000                      0.000000
3                           0.000                      0.000000
4                           0.000                      0.000000
…                               …                             …
1741                        0.161                      0.060415
1742                        0.161                      0.060415
1743                        0.161                      0.060415
1744                        0.161                      0.060415
1745                        0.161                      0.060415
```

[1746 rows x 7 columns]

[65]:
```
# plot all total_growth features together over time using plotly express
fig = px.line(inf_gdp_merged_total_growth, x='date',␣
 ↪y=['inflation_from_8_2011', 'gdp_quart_growth_since_2011',␣
 ↪'gdp_annual_growth_since_2011'], title='Total growth rates over time',␣
 ↪color='variable')
fig.show()
```

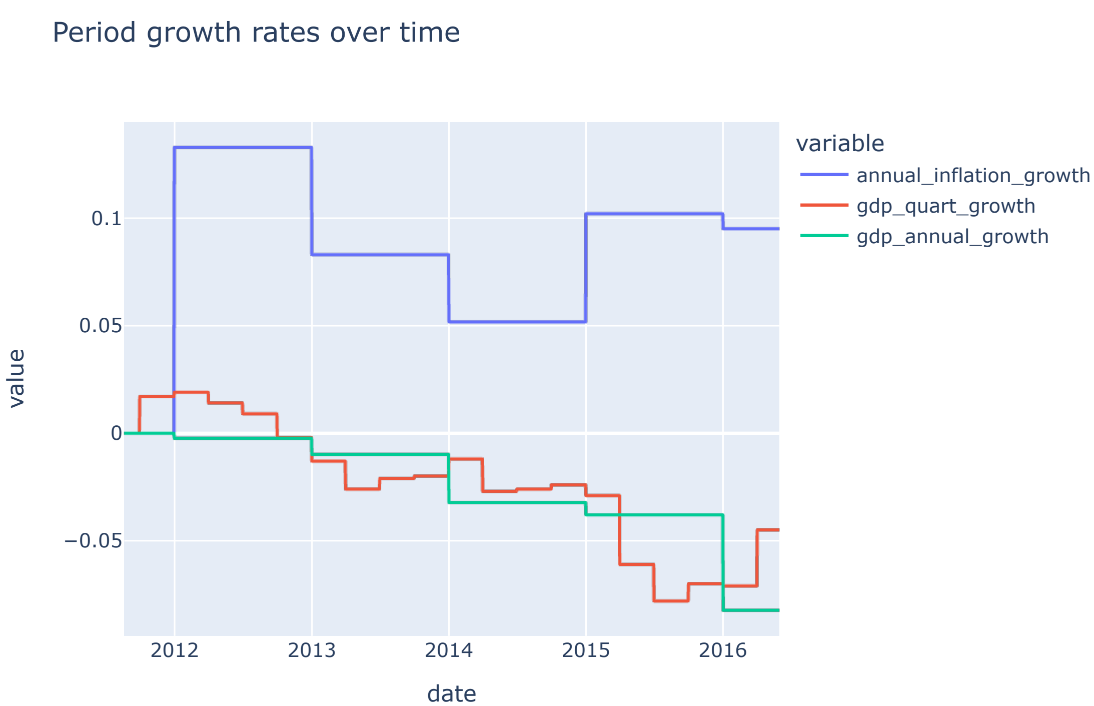


[66]:
```
inf_gdp_merged_period_growth
```

[66]:
```
            date  quarter  year  month  annual_inflation_growth  \
0     2011-08-20        3  2011      8                  0.00000
1     2011-08-21        3  2011      8                  0.00000
2     2011-08-22        3  2011      8                  0.00000
3     2011-08-23        3  2011      8                  0.00000
4     2011-08-24        3  2011      8                  0.00000
…          …       …    …     …                       …
1741  2016-05-26        2  2016      5                  0.09499
1742  2016-05-27        2  2016      5                  0.09499
```

```
1743 2016-05-28         2   2016       5                   0.09499
1744 2016-05-29         2   2016       5                   0.09499
1745 2016-05-30         2   2016       5                   0.09499

      gdp_quart_growth  gdp_annual_growth
0                0.000           0.000000
1                0.000           0.000000
2                0.000           0.000000
3                0.000           0.000000
4                0.000           0.000000
…                  …                …
1741            -0.045          -0.082304
1742            -0.045          -0.082304
1743            -0.045          -0.082304
1744            -0.045          -0.082304
1745            -0.045          -0.082304

[1746 rows x 7 columns]
```

[67]:
```
# plot period growth features together over time using plotly express
fig = px.line(inf_gdp_merged_period_growth, x='date',␣
 ↪y=['annual_inflation_growth', 'gdp_quart_growth', 'gdp_annual_growth'],␣
 ↪title='Period growth rates over time', color='variable')
fig.show()
```



[68]:
```
# group the period growth features by year and month
inf_gdp_merged_monthly_growth = inf_gdp_merged_period_growth.groupby(['year',␣
 ↪'month']).mean()
inf_gdp_merged_monthly_growth.reset_index(inplace=True)
inf_gdp_merged_monthly_growth
## the same per quarter
inf_gdp_merged_quarterly_growth = inf_gdp_merged_period_growth.groupby(['year',␣
 ↪'quarter']).mean()
inf_gdp_merged_quarterly_growth.reset_index(inplace=True)
inf_gdp_merged_quarterly_growth
```

[68]:
```
    year  quarter                date      month  annual_inflation_growth  \
0   2011        3 2011-09-09 12:00:00   8.714286                  0.00000
1   2011        4 2011-11-15 12:00:00  11.000000                  0.00000
2   2012        1 2012-02-15 00:00:00   2.000000                  0.13279
3   2012        2 2012-05-16 00:00:00   5.000000                  0.13279
4   2012        3 2012-08-15 12:00:00   7.989130                  0.13279
5   2012        4 2012-11-15 12:00:00  11.000000                  0.13279
6   2013        1 2013-02-14 12:00:00   2.000000                  0.08299
7   2013        2 2013-05-16 00:00:00   5.000000                  0.08299
8   2013        3 2013-08-15 12:00:00   7.989130                  0.08299
9   2013        4 2013-11-15 12:00:00  11.000000                  0.08299
10  2014        1 2014-02-14 12:00:00   2.000000                  0.05166
11  2014        2 2014-05-16 00:00:00   5.000000                  0.05166
12  2014        3 2014-08-15 12:00:00   7.989130                  0.05166
13  2014        4 2014-11-15 12:00:00  11.000000                  0.05166
14  2015        1 2015-02-14 12:00:00   2.000000                  0.10196
15  2015        2 2015-05-16 00:00:00   5.000000                  0.10196
16  2015        3 2015-08-15 12:00:00   7.989130                  0.10196
17  2015        4 2015-11-15 12:00:00  11.000000                  0.10196
18  2016        1 2016-02-15 00:00:00   2.000000                  0.09499
19  2016        2 2016-04-30 12:00:00   4.500000                  0.09499

    gdp_quart_growth  gdp_annual_growth
0              0.000           0.000000
1              0.017           0.000000
2              0.019          -0.002394
3              0.014          -0.002394
4              0.009          -0.002394
5             -0.002          -0.002394
6             -0.013          -0.009858
7             -0.026          -0.009858
8             -0.021          -0.009858
9             -0.020          -0.009858
```

```
10             -0.012          -0.032242
11             -0.027          -0.032242
12             -0.026          -0.032242
13             -0.024          -0.032242
14             -0.029          -0.037972
15             -0.061          -0.037972
16             -0.078          -0.037972
17             -0.070          -0.037972
18             -0.071          -0.082304
19             -0.045          -0.082304
```

[69]:
```
# for all ratings we found creating a wining function
def wine_rate(input_rating, input_period_in_year, output_period):
   return (1 + input_rating)**(output_period/input_period_in_year) - 1
```

[70]:
```
inf_gdp_merged_quarterly_growth['wined_annual_growth_tquart'] =␣
↪inf_gdp_merged_quarterly_growth['annual_inflation_growth'].apply(lambda x:␣
↪wine_rate(x, 1, 4))
inf_gdp_merged_quarterly_growth['wined_gdp_quart_growth_tquart'] =␣
↪inf_gdp_merged_quarterly_growth['gdp_quart_growth'].apply(lambda x:␣
↪wine_rate(x, 1, 4))
```

[71]:
```
# predict quarterly inflation growth rate using the annual inflation growth␣
↪rate, and the gdp growth rates
X = inf_gdp_merged_quarterly_growth[['annual_inflation_growth',␣
↪'gdp_quart_growth', 'gdp_annual_growth']]
# use a wined annual inflation growth rate as the constant for the model to␣
↪predict the quarterly inflation growth rate
```

[72]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


# Define the independent variables (features)
X = inf_gdp_merged_quarterly_growth[['annual_inflation_growth',␣
↪'gdp_quart_growth', 'gdp_annual_growth']]

# Use the annual inflation growth rate as a constant for prediction
constant = inf_gdp_merged_quarterly_growth['annual_inflation_growth'].mean()

# Predict the quarterly inflation growth rate using a simple aggregation (e.g.,␣
↪mean or median)
predicted_quarterly_inflation_growth = X.mean(axis=1)  # You can also use other␣
↪aggregation functions like median

# Print the predicted quarterly inflation growth rate
```

```python
print("Predicted Quarterly Inflation Growth Rate:")
print(predicted_quarterly_inflation_growth)
# MAKINg sure to add a zero in the first row
predicted_quarterly_inflation_growth = pd.concat([pd.Series([0]),
 ↪predicted_quarterly_inflation_growth], ignore_index=True)
# Print the predicted quarterly inflation growth rate
print("Predicted Quarterly Inflation Growth Rate:")
print(predicted_quarterly_inflation_growth)
```

```
Predicted Quarterly Inflation Growth Rate:
0     0.000000
1     0.005667
2     0.049799
3     0.048132
4     0.046465
5     0.042799
6     0.020044
7     0.015711
8     0.017377
9     0.017711
10    0.002473
11   -0.002527
12   -0.002194
13   -0.001527
14    0.011663
15    0.000996
16   -0.004671
17   -0.002004
18   -0.019438
19   -0.010771
dtype: float64
Predicted Quarterly Inflation Growth Rate:
0     0.000000
1     0.000000
2     0.005667
3     0.049799
4     0.048132
5     0.046465
6     0.042799
7     0.020044
8     0.015711
9     0.017377
10    0.017711
11    0.002473
12   -0.002527
13   -0.002194
14   -0.001527
15    0.011663
```

```
16    0.000996
17   -0.004671
18   -0.002004
19   -0.019438
20   -0.010771
dtype: float64
```

[73]:

```
# add the predicted quarterly inflation growth rate to the dataframe
inf_gdp_merged_quarterly_growth['predicted_quarterly_inflation_growth'] =␣
 ↪predicted_quarterly_inflation_growth
inf_gdp_merged_quarterly_growth
```

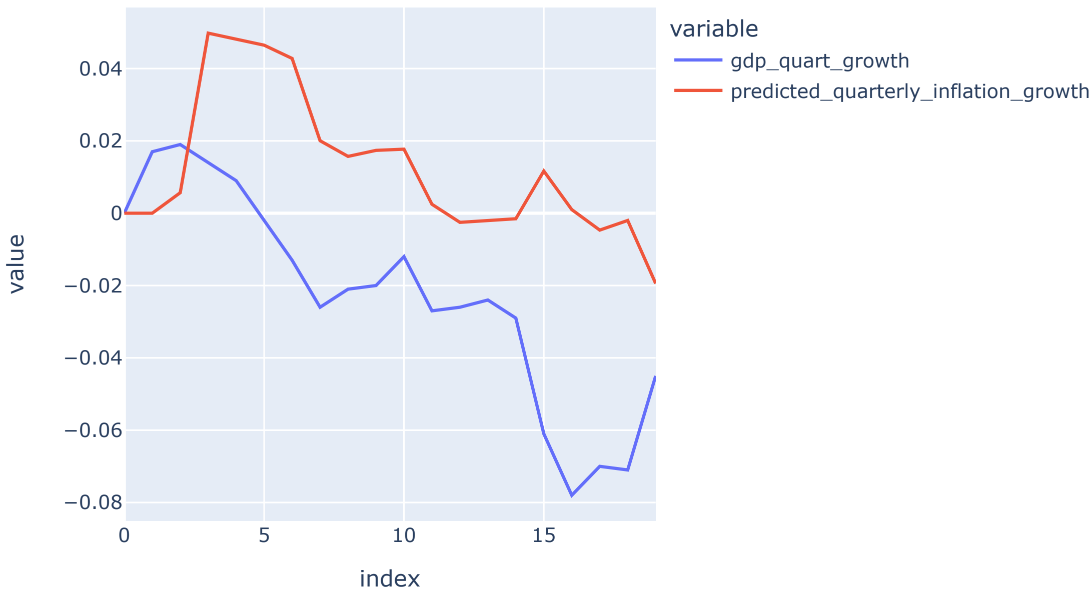
[73]:

```
    year  quarter                date      month  annual_inflation_growth  \
0   2011        3 2011-09-09 12:00:00   8.714286                  0.00000
1   2011        4 2011-11-15 12:00:00  11.000000                  0.00000
2   2012        1 2012-02-15 00:00:00   2.000000                  0.13279
3   2012        2 2012-05-16 00:00:00   5.000000                  0.13279
4   2012        3 2012-08-15 12:00:00   7.989130                  0.13279
5   2012        4 2012-11-15 12:00:00  11.000000                  0.13279
6   2013        1 2013-02-14 12:00:00   2.000000                  0.08299
7   2013        2 2013-05-16 00:00:00   5.000000                  0.08299
8   2013        3 2013-08-15 12:00:00   7.989130                  0.08299
9   2013        4 2013-11-15 12:00:00  11.000000                  0.08299
10  2014        1 2014-02-14 12:00:00   2.000000                  0.05166
11  2014        2 2014-05-16 00:00:00   5.000000                  0.05166
12  2014        3 2014-08-15 12:00:00   7.989130                  0.05166
13  2014        4 2014-11-15 12:00:00  11.000000                  0.05166
14  2015        1 2015-02-14 12:00:00   2.000000                  0.10196
15  2015        2 2015-05-16 00:00:00   5.000000                  0.10196
16  2015        3 2015-08-15 12:00:00   7.989130                  0.10196
17  2015        4 2015-11-15 12:00:00  11.000000                  0.10196
18  2016        1 2016-02-15 00:00:00   2.000000                  0.09499
19  2016        2 2016-04-30 12:00:00   4.500000                  0.09499

    gdp_quart_growth  gdp_annual_growth  wined_annual_growth_tquart  \
0              0.000           0.000000                    0.000000
1              0.017           0.000000                    0.000000
2              0.019          -0.002394                    0.646636
3              0.014          -0.002394                    0.646636
4              0.009          -0.002394                    0.646636
5             -0.002          -0.002394                    0.646636
6             -0.013          -0.009858                    0.375618
7             -0.026          -0.009858                    0.375618
8             -0.021          -0.009858                    0.375618
9             -0.020          -0.009858                    0.375618
10            -0.012          -0.032242                    0.223211
11            -0.027          -0.032242                    0.223211
```

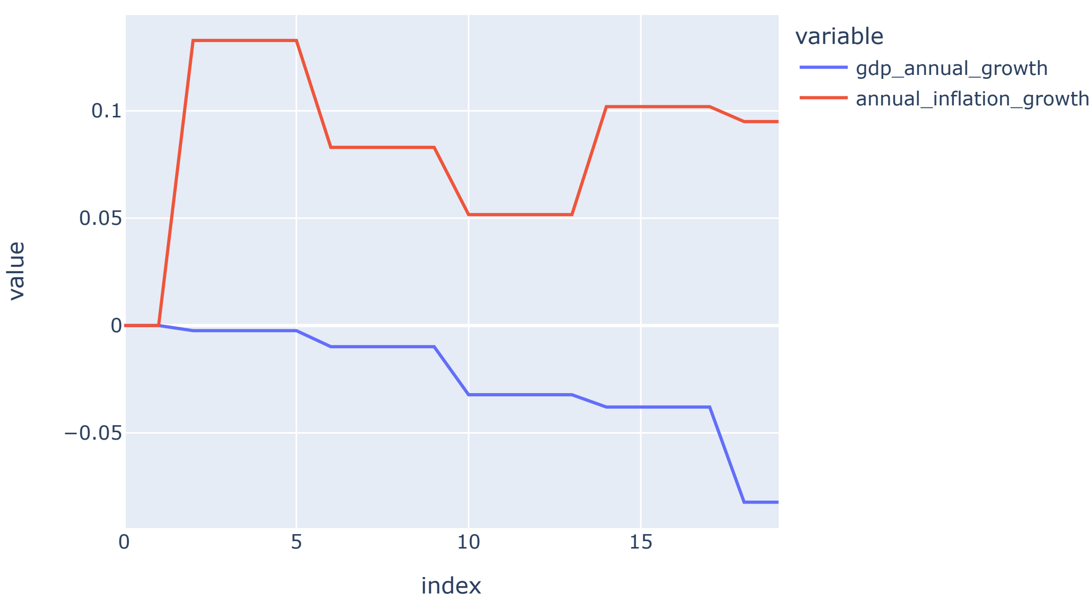
```
12          -0.026       -0.032242               0.223211
13          -0.024       -0.032242               0.223211
14          -0.029       -0.037972               0.474563
15          -0.061       -0.037972               0.474563
16          -0.078       -0.037972               0.474563
17          -0.070       -0.037972               0.474563
18          -0.071       -0.082304               0.437608
19          -0.045       -0.082304               0.437608

    wined_gdp_quart_growth_tquart  predicted_quarterly_inflation_growth
0                        0.000000                              0.000000
1                        0.069754                              0.000000
2                        0.078194                              0.005667
3                        0.057187                              0.049799
4                        0.036489                              0.048132
5                       -0.007976                              0.046465
6                       -0.050995                              0.042799
7                       -0.100014                              0.020044
8                       -0.081391                              0.015711
9                       -0.077632                              0.017377
10                      -0.047143                              0.017711
11                      -0.103704                              0.002473
12                      -0.100014                             -0.002527
13                      -0.092599                             -0.002194
14                      -0.111051                             -0.001527
15                      -0.222568                              0.011663
16                      -0.277357                              0.000996
17                      -0.251948                             -0.004671
18                      -0.255160                             -0.002004
19                      -0.168210                             -0.019438
```

[74]:
```
## plot the predicted quarterly inflation growth rate vs the actual quarterly␣
↪gdp_growth as well as the annual inflation growth rate and annual gdp growth␣
↪rate
### over time
fig = px.line(inf_gdp_merged_quarterly_growth, x=␣
↪inf_gdp_merged_quarterly_growth.index, y=[ 'gdp_quart_growth',␣
↪'predicted_quarterly_inflation_growth'], title='Annual inflation growth rate␣
↪vs predicted quarterly inflation growth rate vs gdp growth rate')
fig.show()
##plot annual igdp growth rate vs predicted quarterly inflation growth rate
fig = px.line(inf_gdp_merged_quarterly_growth, x=␣
↪inf_gdp_merged_quarterly_growth.index, y=['gdp_annual_growth',␣
↪'annual_inflation_growth'], title='Annual gdp growth rate vs predicted␣
↪quarterly inflation growth rate')
fig.show()
```

Annual inflation growth rate vs predicted quarterly inflation growth rate vs gd



Annual gdp growth rate vs predicted quarterly inflation growth rate



4.7 Categorical Variables

[75]:
```
def CategoricalHelper(data,target_encoding = False):
    data = data.select_dtypes('object')
    classes = {}
    for col in data.columns:
        if data[col].nunique() == 2:
            classes[col] = data[col].unique()
        elif col == 'ecology':
            classes[col] = 'Encoder'
        else:
            if target_encoding and col == 'sub_area':
                classes[col] = 'TargetEncoding'
            else:
                classes[col] = 'Dummies'
    return classes
```

[76]:
```
ecology = ['excellent','good','satisfactory','poor', 'no data']
ecology_encoding = [4,3,2,1,-1]
encoder = dict(zip(ecology,ecology_encoding))
encoder
```

[76]:
```
{'excellent': 4, 'good': 3, 'satisfactory': 2, 'poor': 1, 'no data': -1}
```

[77]:
```
def categoricalToNumbers(data,classes,encoder,m=300):
    # Data Contains categorical variables only.
    for c in classes:
        if type(classes[c]) == str:
            if classes[c] == 'Dummies':
                dummies = pd.get_dummies(data[c])*1
                data = data.drop(columns = c).join(dummies)
            elif classes[c] == 'TargetEncoding':
                sa_grp = data.groupby(c)['price_sq'].transform('mean')
                sa_count = data[c].value_counts().to_dict()
                sa_counts = data[c].map(sa_count)
                sa_weights = (sa_counts)/(sa_counts + m)
                data[c] = sa_grp*(sa_weights) + data['price_sq'].
 ↪mean()*(1-sa_weights)
            else:
                data.loc[:,c] = data[c].apply(lambda val: encoder[val])
        else:
            data.loc[:,c] = data[c].apply(lambda val: (val == 'yes')*1)
    return data
```

```
[78]: classes = CategoricalHelper(full_data, True)
      classes.pop('product_type')
      classes
```

```
[78]: {'sub_area': 'TargetEncoding',
       'culture_objects_top_25': array(['no', 'yes'], dtype=object),
       'thermal_power_plant_raion': array(['no', 'yes'], dtype=object),
       'incineration_raion': array(['no', 'yes'], dtype=object),
       'oil_chemistry_raion': array(['no', 'yes'], dtype=object),
       'radiation_raion': array(['no', 'yes'], dtype=object),
       'railroad_terminal_raion': array(['no', 'yes'], dtype=object),
       'big_market_raion': array(['no', 'yes'], dtype=object),
       'nuclear_reactor_raion': array(['no', 'yes'], dtype=object),
       'detention_facility_raion': array(['no', 'yes'], dtype=object),
       'water_1line': array(['no', 'yes'], dtype=object),
       'big_road1_1line': array(['no', 'yes'], dtype=object),
       'railroad_1line': array(['no', 'yes'], dtype=object),
       'ecology': 'Encoder'}
```

```
[79]: full_data = categoricalToNumbers(full_data, classes, encoder)
```

5 Correlation Analysis

On this part we wanted to understand if our independent features correlate with each other to check if there is some redundancy in our data. So for each product type we checked the correlation within each group.

We have separated the correlation analysis for each product type

```
[80]: # Get the dtypes for each group
      groups_dtypes = {group:list(groups_dfs[group].dtypes.unique()) for group in␣
       ↪groups_dfs} # we don't need buildings anymore
      groups_dtypes
```

```
[80]: {'areas': [dtype('float64'), dtype('int64')],
       'buildings': [dtype('float64')],
       'demographics': [dtype('float64')],
       'distances': [dtype('float64'), dtype('int64'), dtype('O')],
       'education': [dtype('float64')],
       'facilities': [dtype('float64'), dtype('int64'), dtype('O')],
       'interior': [dtype('int64'), dtype('O'), dtype('float64')],
       'surroundings': [dtype('O'), dtype('int64'), dtype('float64')]}
```

```
[81]: from matplotlib.colors import LinearSegmentedColormap

      def createHeatMap(corr, group, annot=True):
          plt.figure(figsize=(10,10))
```

```
    cmap: LinearSegmentedColormap =  sns.diverging_palette(220, 20,␣
↪as_cmap=True)
    sns.heatmap(corr, cmap=cmap, annot=annot, fmt=".2f")
    plt.title(f'Correlation Matrix of {group}')
    plt.tight_layout()
    plt.show()
```

[82]:
```
groups_corrs = ['areas','demographics','surroundings','interior','distances'] #␣
↪Groups for the correlation Analysis.
```

[83]:
```
# Change ID's to INT
groups_dfs['distances'].loc[:,groups_dfs['distances'].filter(like='ID').
↪columns] = groups_dfs['distances'].filter(like='ID').astype('int64')
```

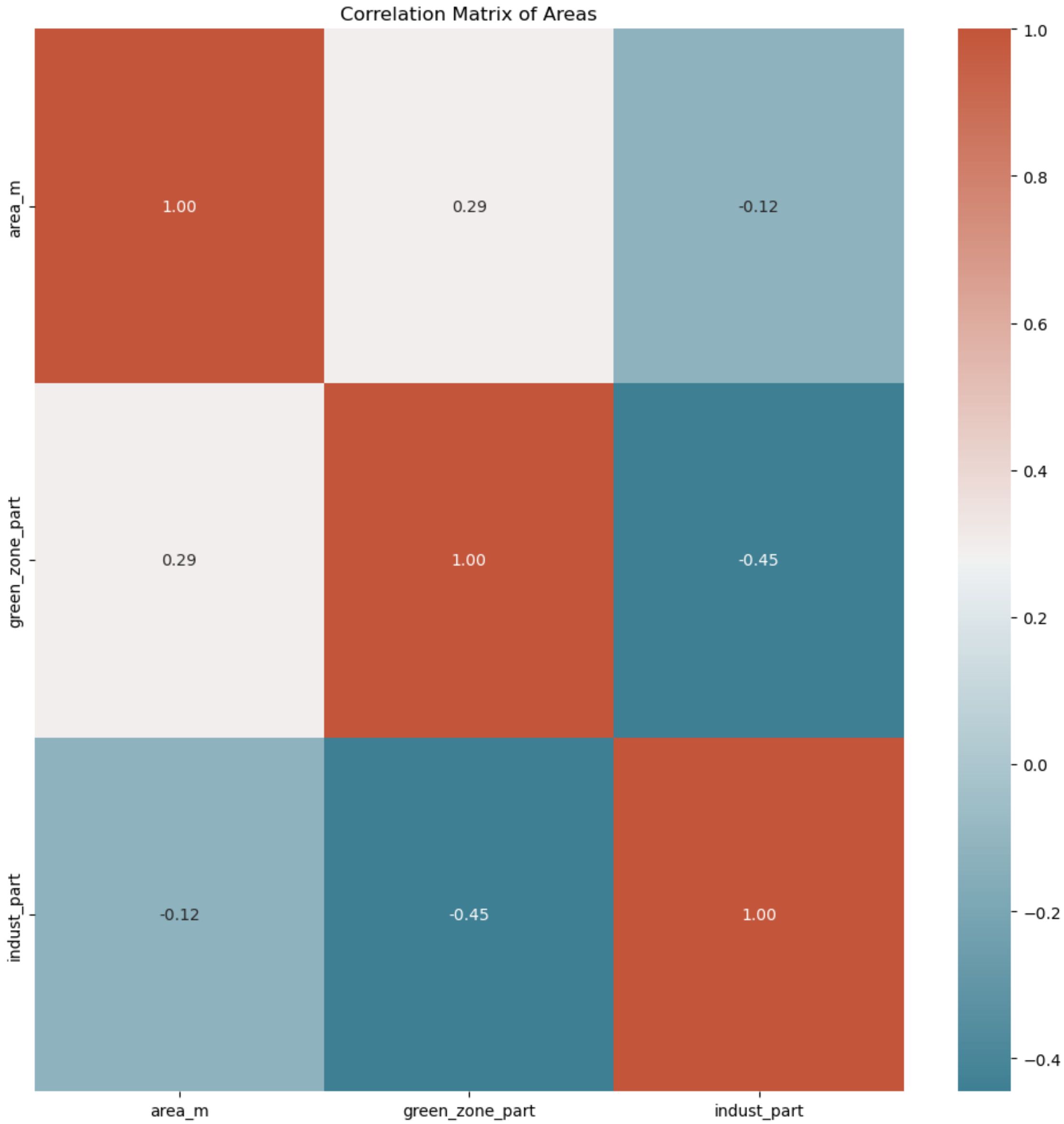
[84]:
```
# Create a dictionary for each group with the correlation matrix.
groups_floats = {group:groups_dfs[group].select_dtypes(['float64']) for group␣
↪in groups_dfs}
```

5.0.1 Investment

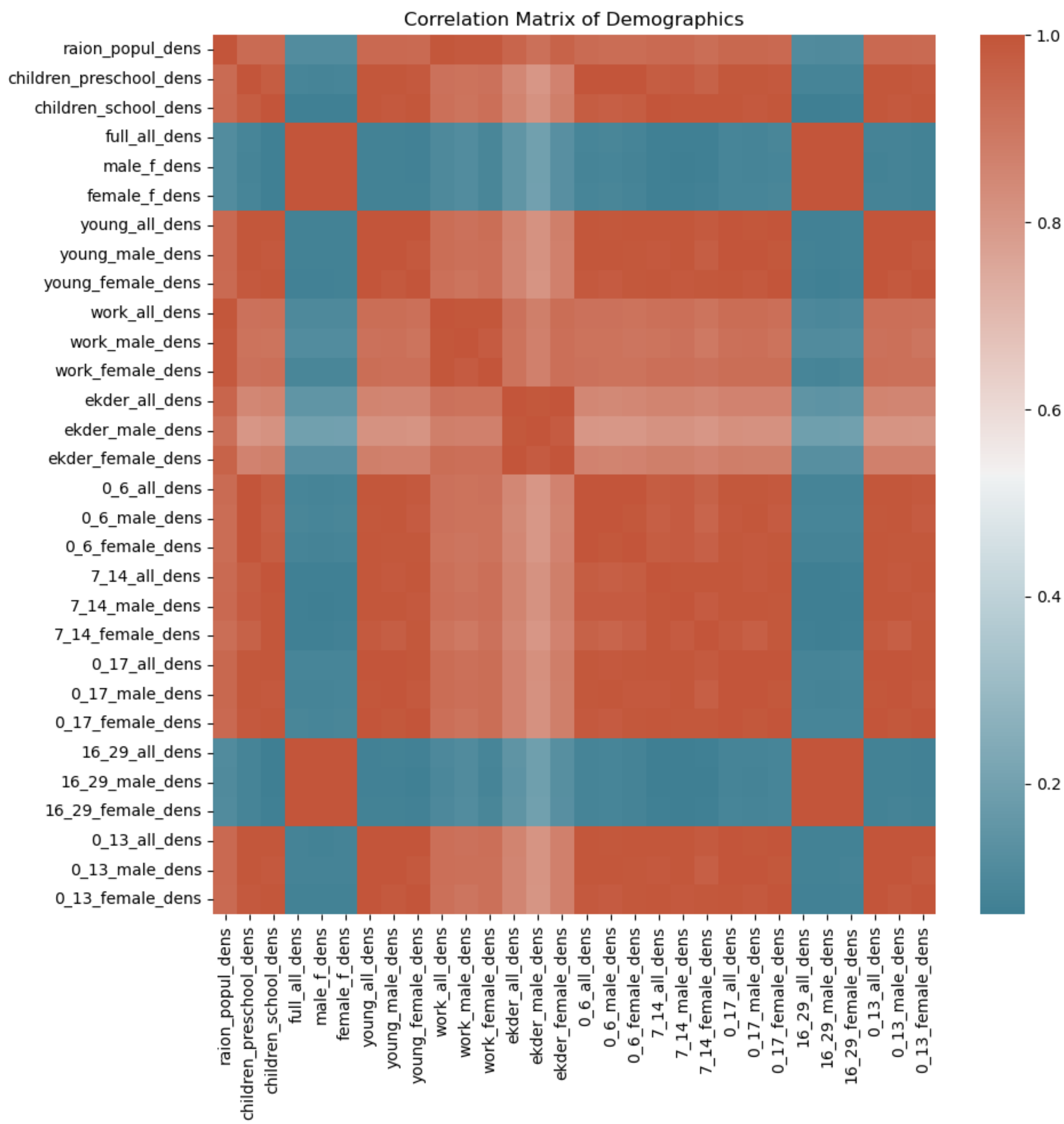
[85]:
```
investment = full_data[full_data['product_type'] == 'Investment']
investment.drop(columns = 'product_type', inplace=True)
```

```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/514358765.py:2:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

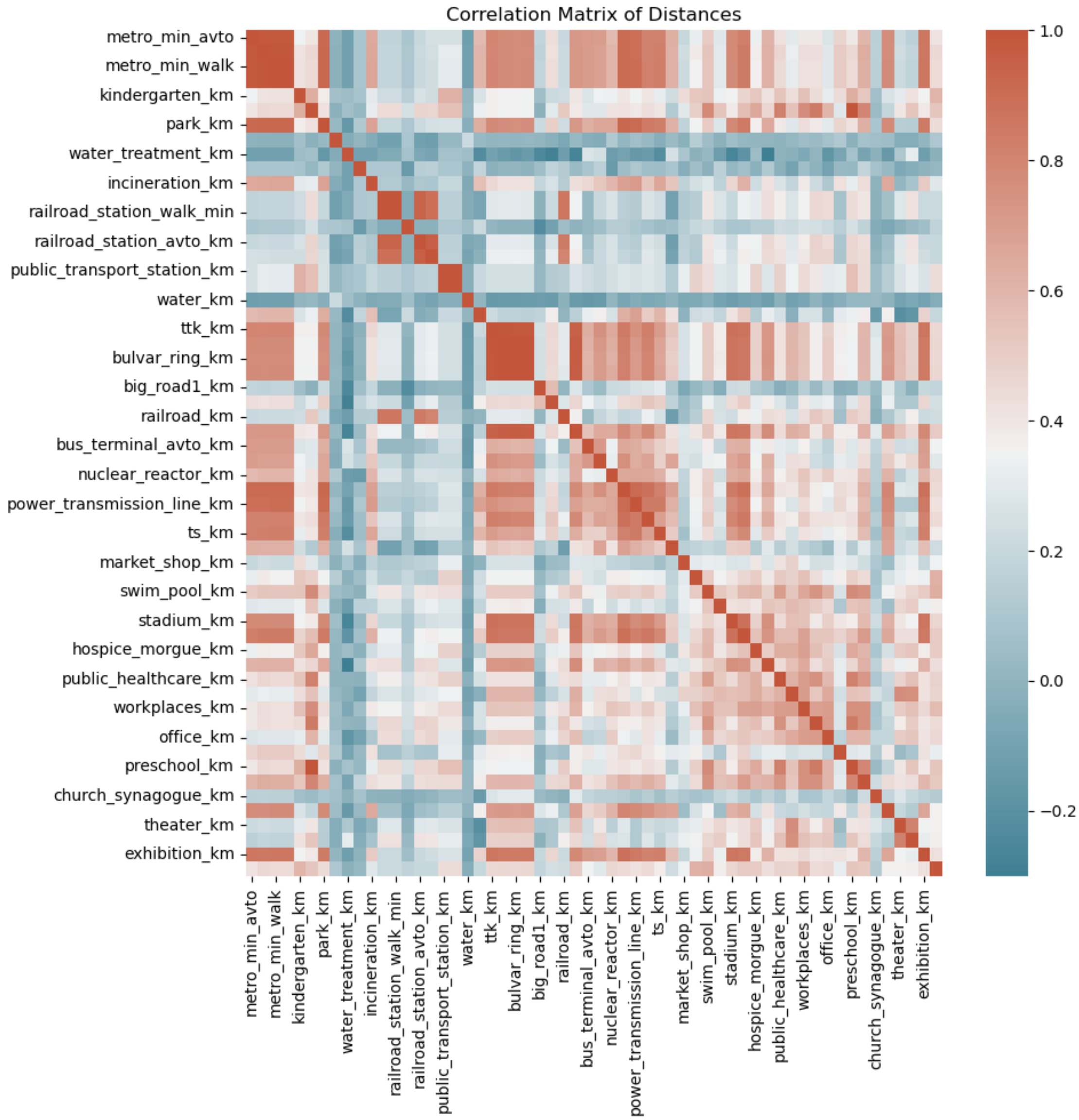
[86]:
```
createHeatMap(investment[groups_floats['areas'].columns].corr(), 'Areas')
```



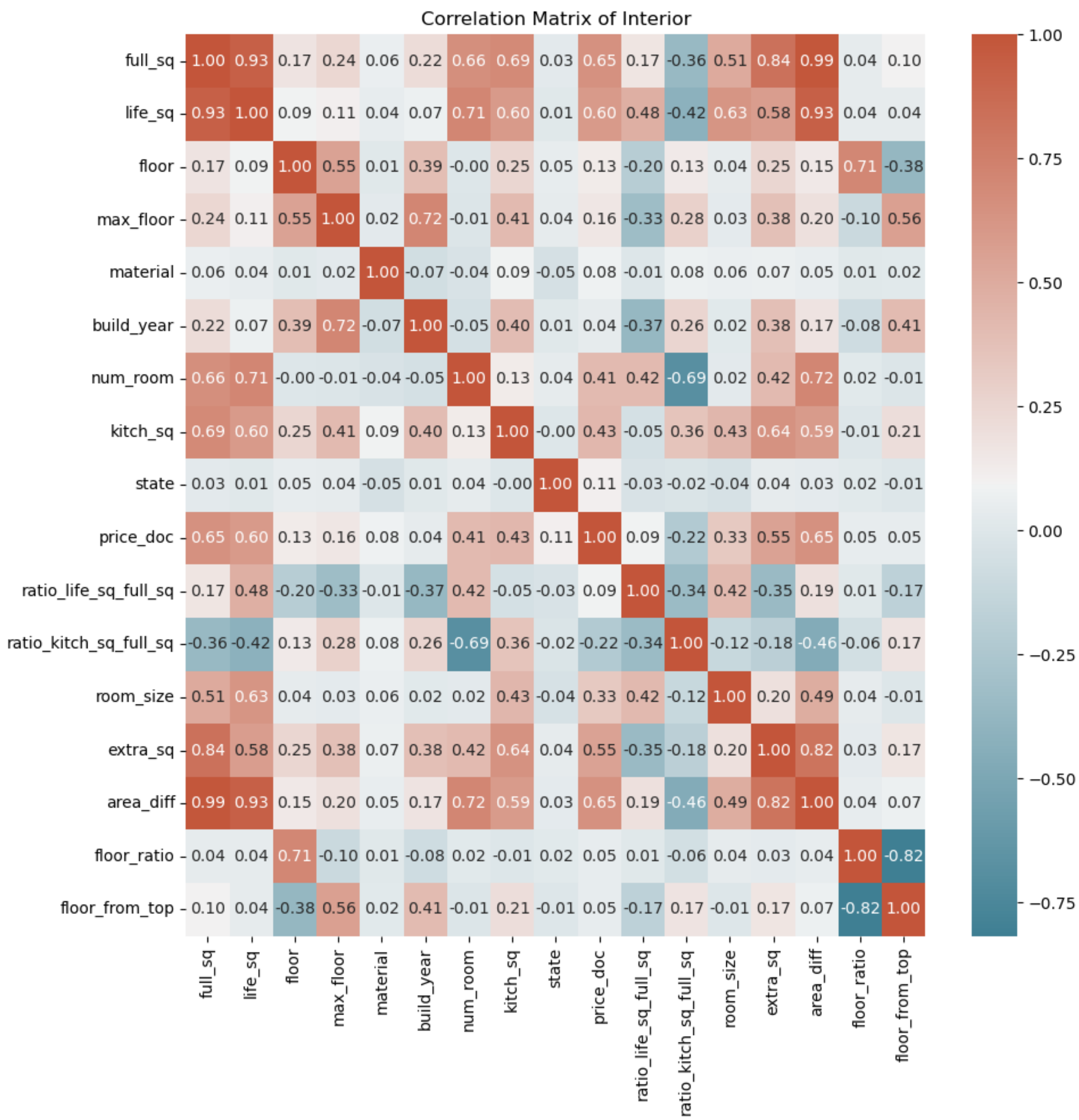
[87]:
```
createHeatMap(investment[groups_floats['demographics'].columns].corr(),␣
↪'Demographics', annot=False)
```



[88]:
```
createHeatMap(investment[groups_floats['distances'].columns].corr(),␣
↪'Distances', annot=False)
```

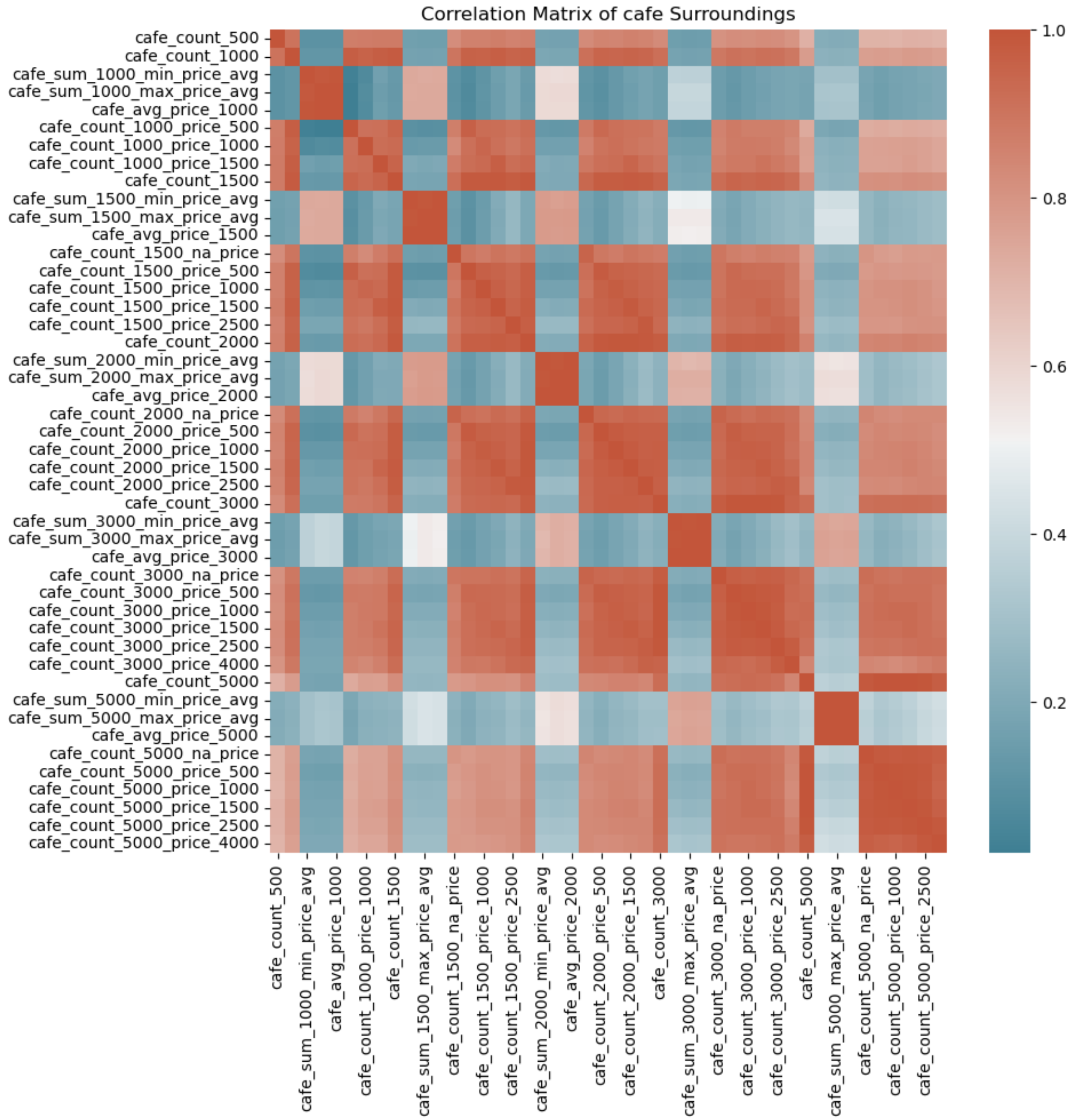



```
[89]: createHeatMap(investment[groups_floats['interior'].columns].corr(), 'Interior')
```



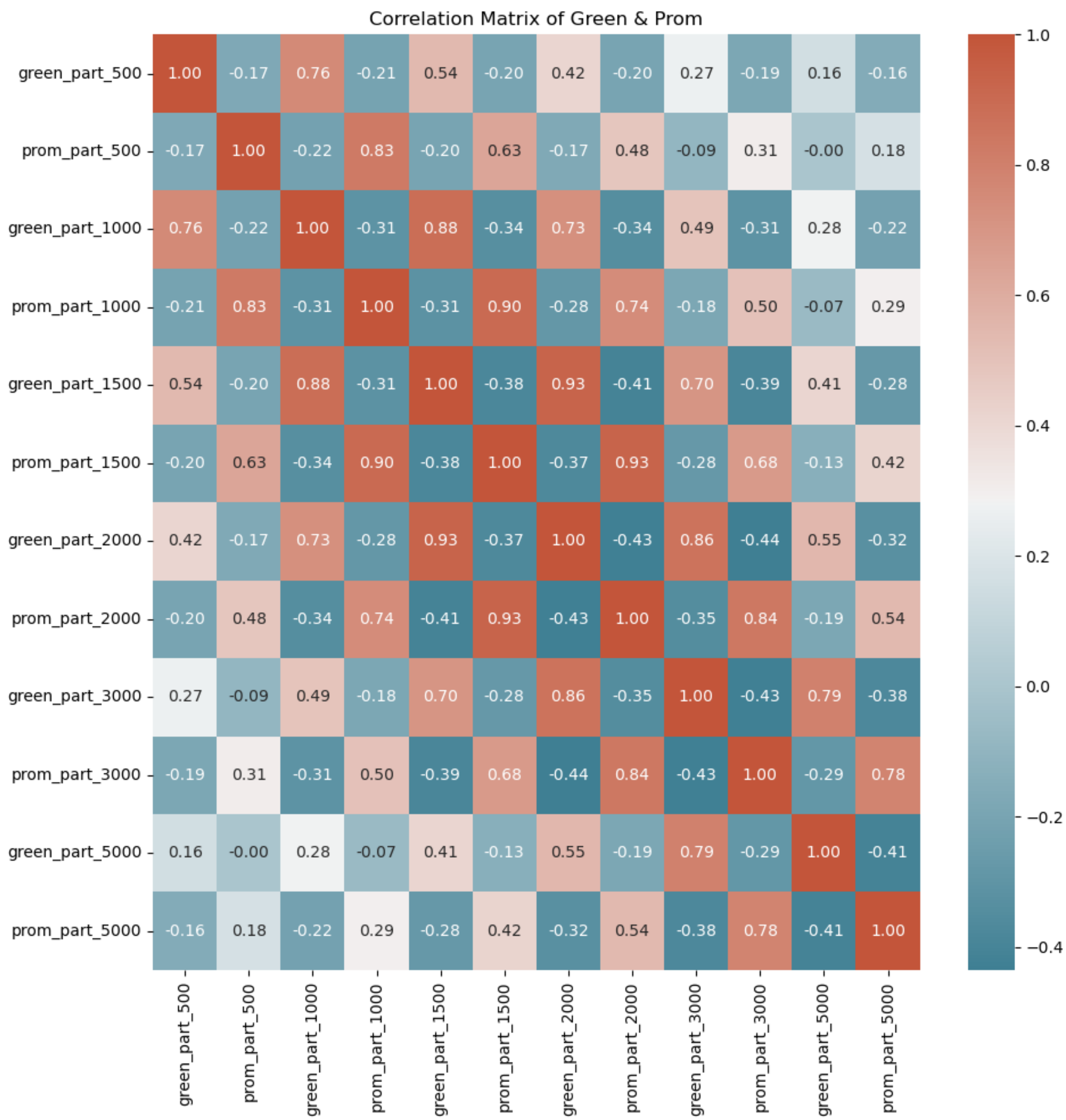
[90]:
```
cafe_invst = [col for col in investment.filter(like='cafe').columns if␣
↪'_binary' not in col]
green_invst = [col for col in groups_floats['surroundings'].columns if 'green'␣
↪in col.lower() or 'prom' in col.lower()]
```

[91]:
```
createHeatMap(investment[cafe_invst].corr(), 'cafe Surroundings', annot=False)
```



[92]:

```
createHeatMap(investment[green_invst].corr(), 'Green & Prom')
```



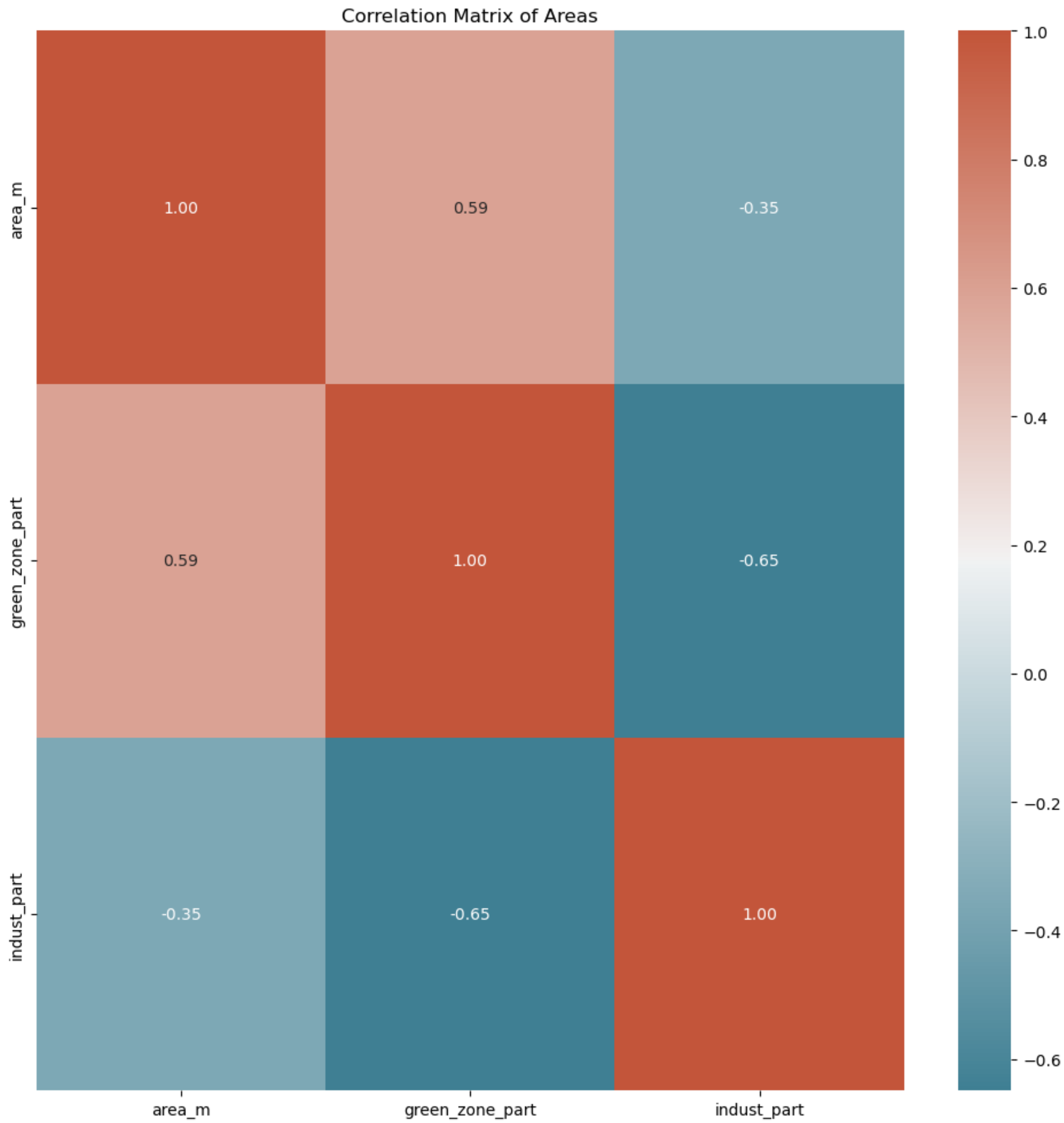
5.0.2 OwnerOcuppier

[93]:
```
ocuppier = full_data[full_data['product_type'] == 'OwnerOccupier']
ocuppier.drop(columns = 'product_type', inplace=True)
```

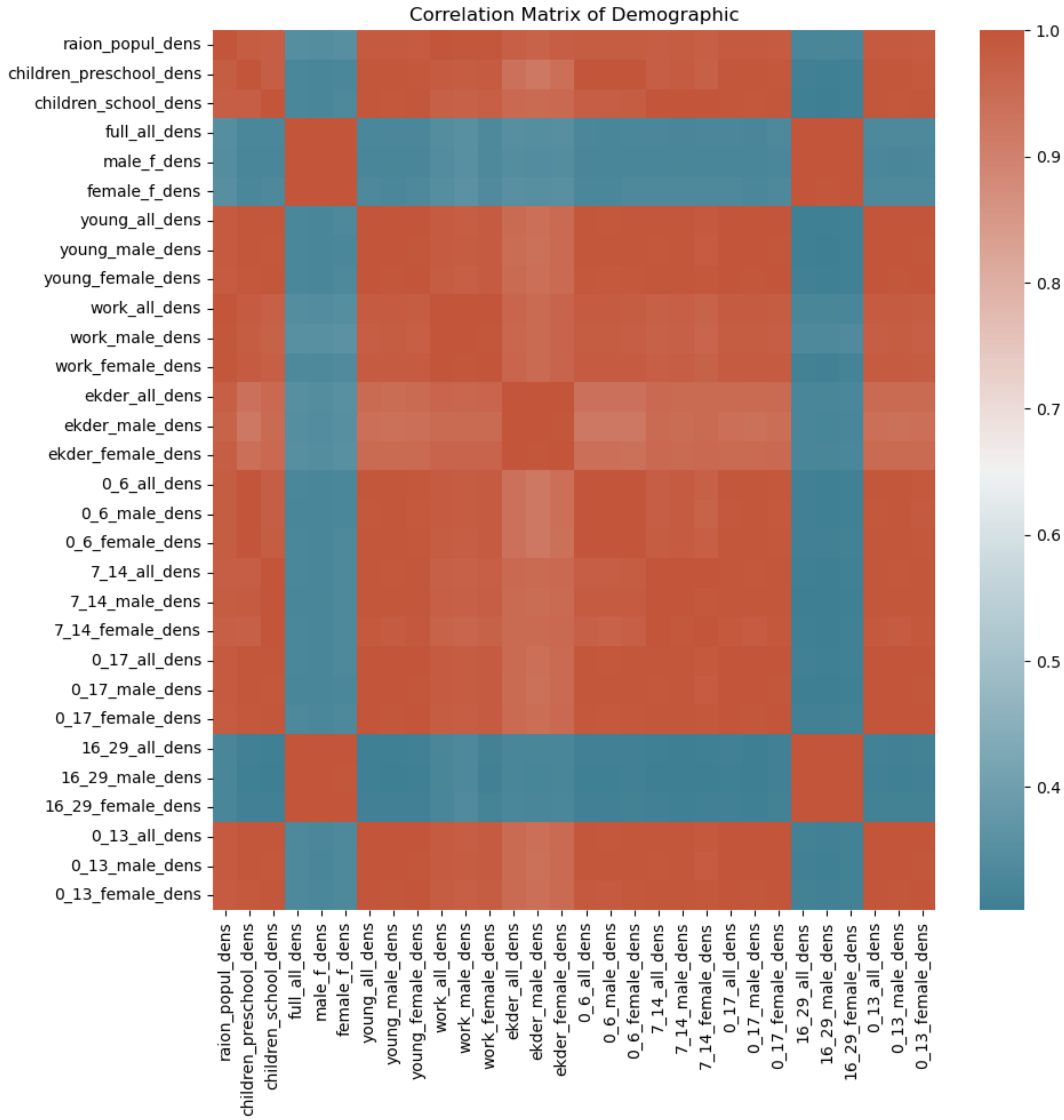
```
/var/folders/2y/5vlst1hd6jz9tggyvm776y3m0000gn/T/ipykernel_10945/1144162341.py:2
: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

[94]:
```
createHeatMap(ocuppier[groups_floats['areas'].columns].corr(), 'Areas')
```

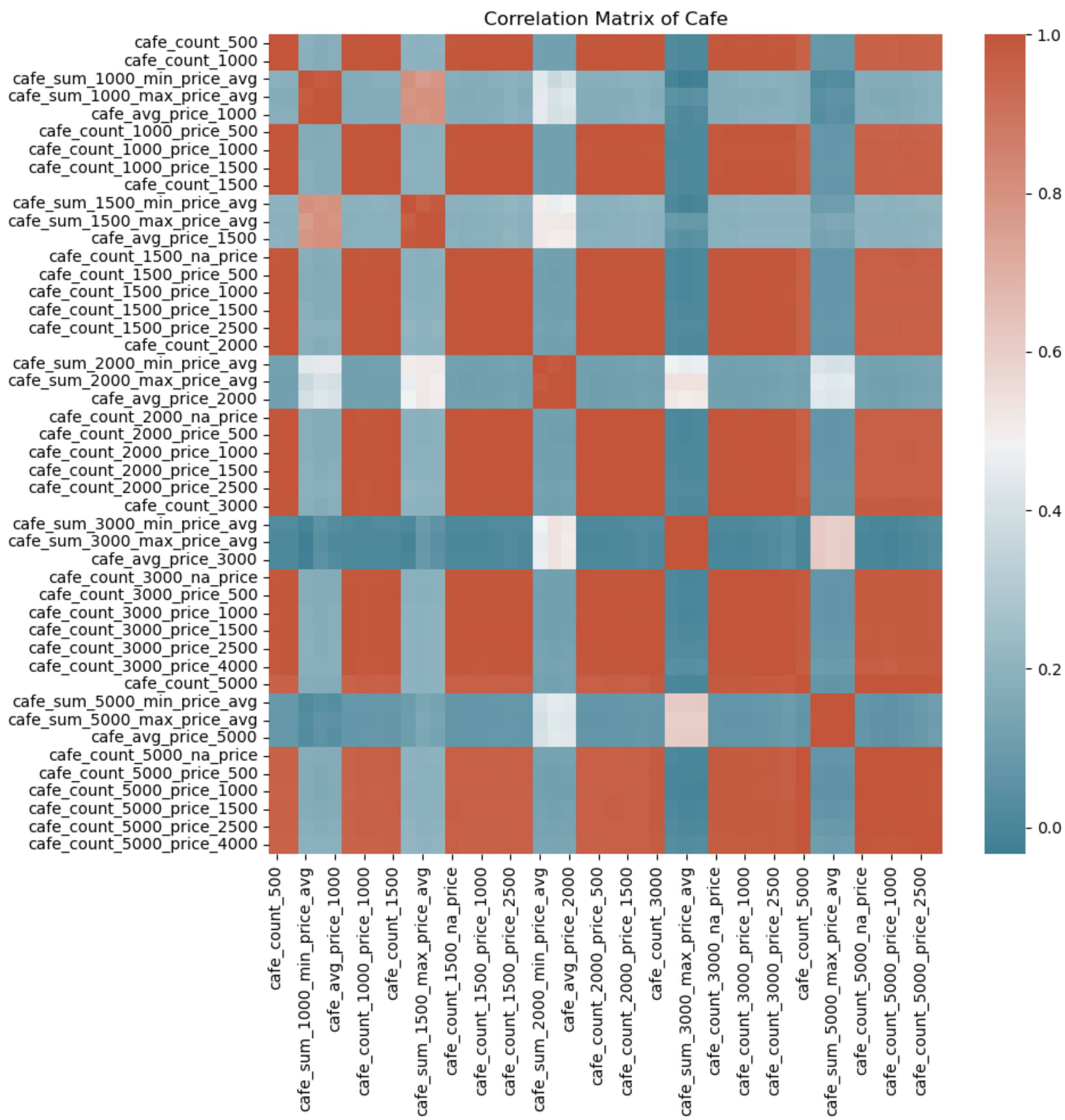


[95]:
```
createHeatMap(ocuppier[groups_dfs['demographics'].columns].corr(),␣
↪'Demographic', annot=False)
```

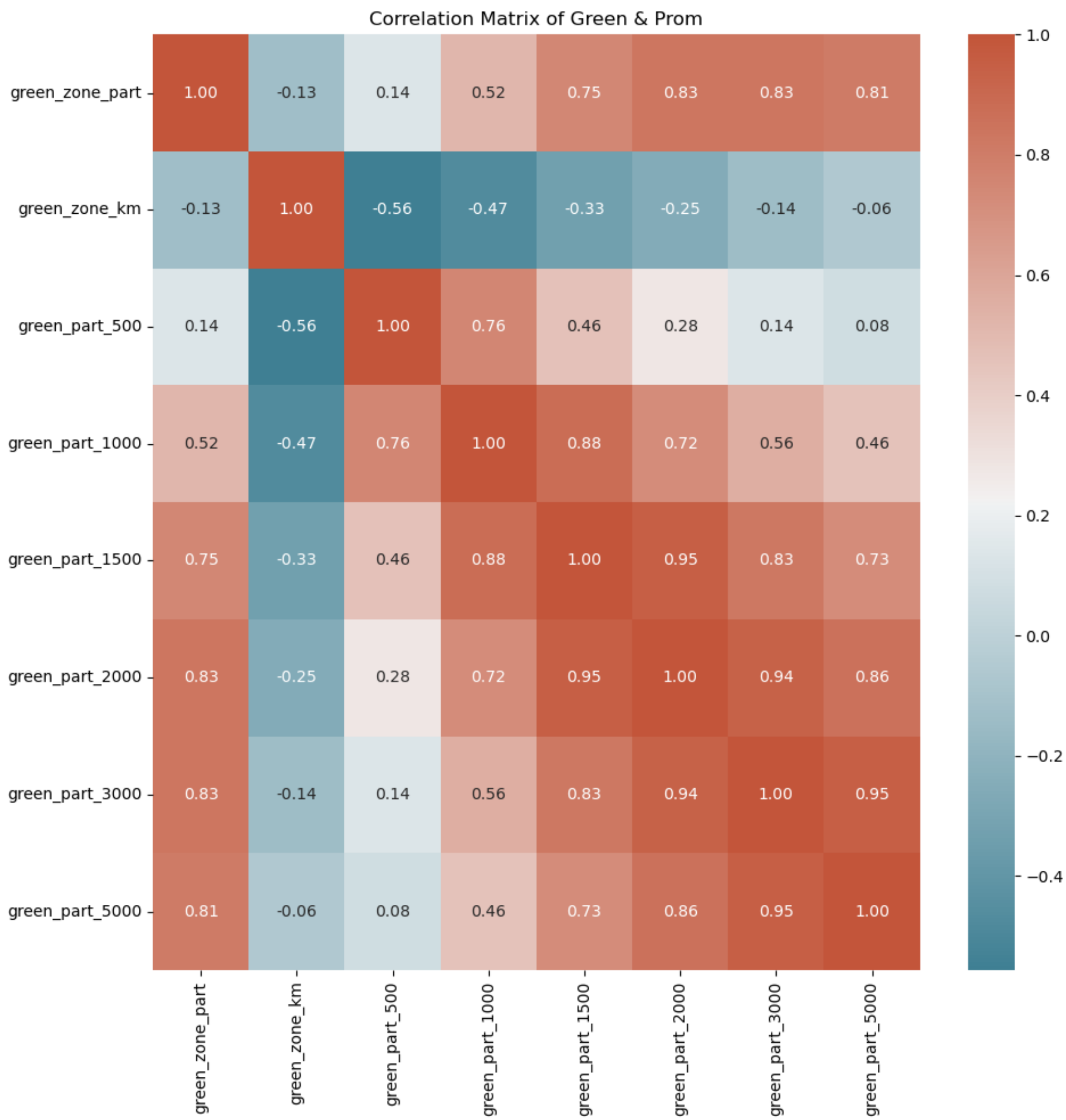


[96]:
```
cafe_ocu = ocuppier[[col for col in ocuppier.filter(like='cafe') if '_binary'␣
  ↪not in col]]
green_prom_ocu =  ocuppier[ocuppier.columns[np.where(ocuppier.columns.str.
  ↪contains('green','prom'))]]
```

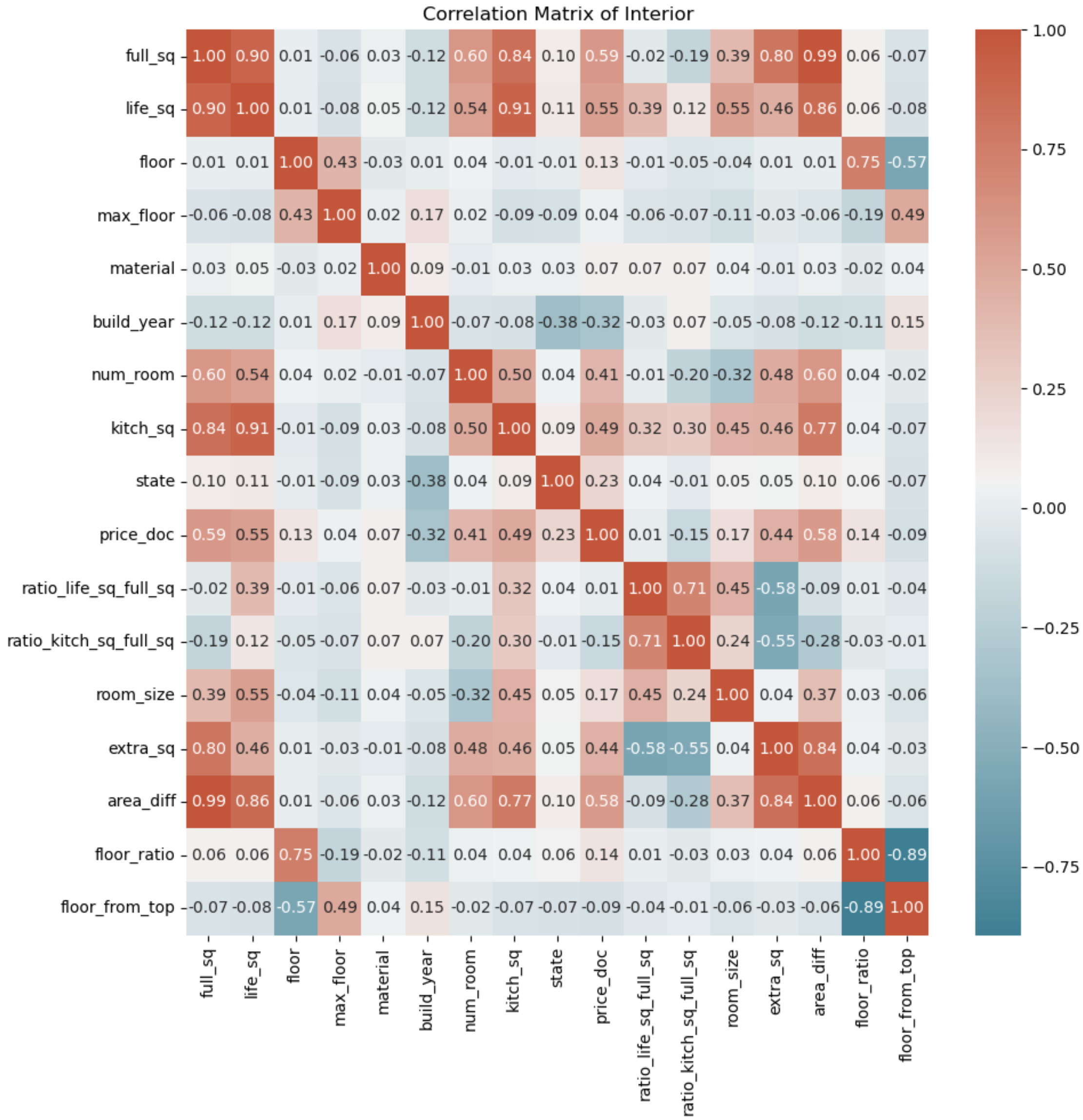
[97]:
```
createHeatMap(corr=cafe_ocu.corr(), group='Cafe', annot=False)
```



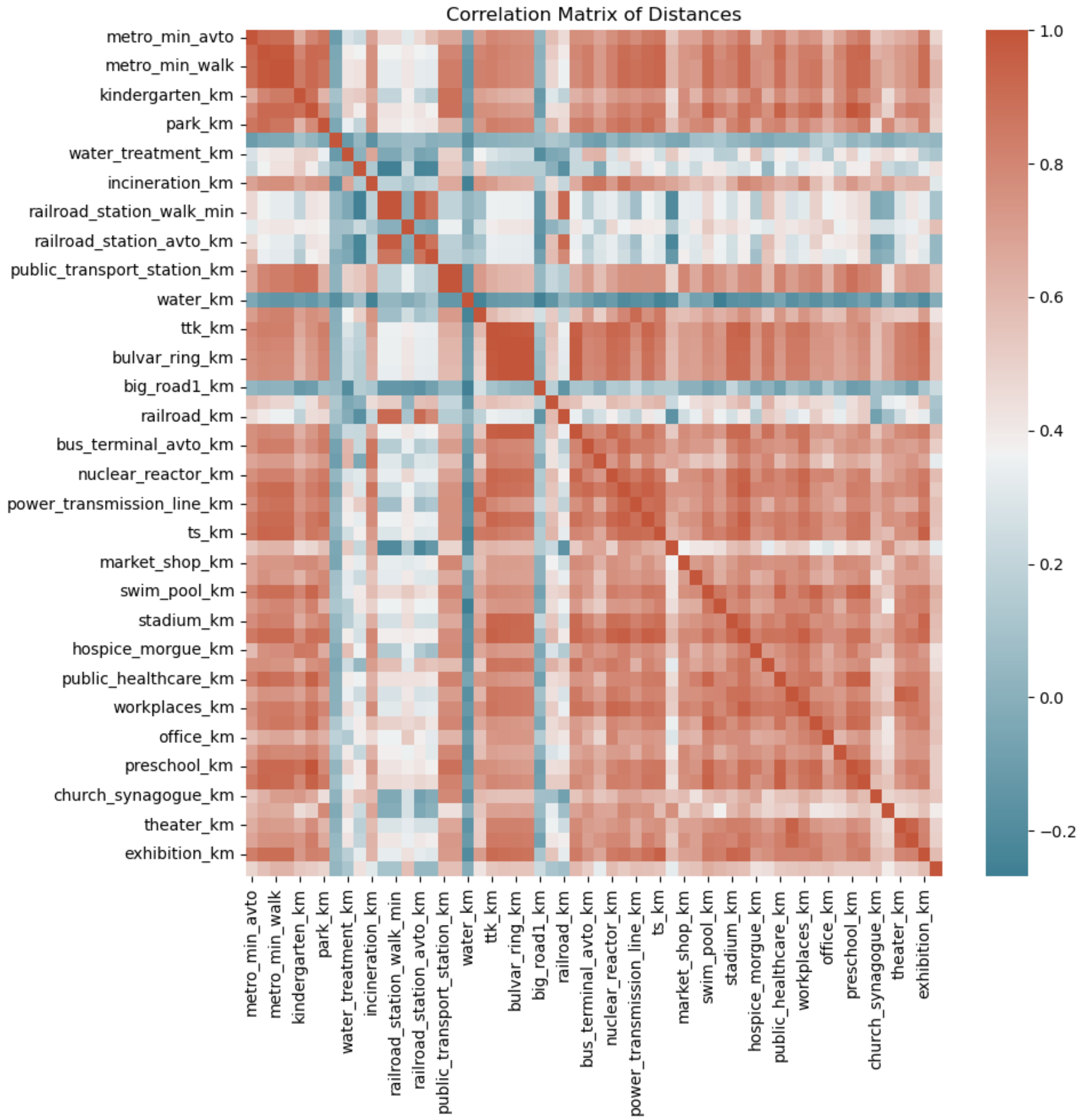
[98]:
```
createHeatMap(green_prom_ocu.corr(),'Green & Prom')
```



[99]:
```
createHeatMap(ocuppier[groups_floats['interior'].columns].corr(), 'Interior')
```



[100]:
```
createHeatMap(ocuppier[groups_floats['distances'].columns].corr(), 'Distances',
    annot=False)
```



We have some high correlation between independent features which might indicate that we have some redundancy in our data this we might need to use PCA / carefully remove them.

6 Principal Componenet Analysis

To handle the redundancy in our data, we'll use PCA. However, when using PCA choosing the right number of components is crucial. To achive that we'll use a scree plot and we will choose the number of componenets that hold between 85%-90% of the variance.

We created two functions that will help us understand what is the number of componenets that will preserve at least 85% of the variance.

[101]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Plotting explained variance ratio vs. number of components
def plotPCA(pca_data, ratio):
    scaler = StandardScaler()
    pca = PCA()

    # Scaler & PCA Pipeline.
    pipe = Pipeline([('scaler',scaler),('pca',pca)])
    pipe.fit(pca_data)

    # Visualize explained variance ratio
    plt.figure(figsize = (10,8))
    plt.plot(np.cumsum(pca.explained_variance_ratio_))
    plt.xticks(np.arange(0, len(np.cumsum(pca.explained_variance_ratio_))) )
    plt.xlabel('Number of Components')
    plt.ylabel('Cumulative Explained Variance Ratio')
    plt.title('Explained Variance Ratio vs. Number of Components')
    plt.grid(True)
    plt.show()
    return np.where(np.cumsum(pca.explained_variance_ratio_) >= ratio)[0].min()␣
↪+ 1

def pca_df(num_components, data, name):
    # Pca & Scaler pipeline.
    pca = PCA(num_components)
    scaler = StandardScaler()
    pipe_2 = Pipeline([('scaler',scaler),('pca',pca)])
    # Getting the new features.
    x = pipe_2.fit_transform(data)
    names = [f'{name}_pca{i}' for i in range(1,num_components+1)]
    return pd.DataFrame(x, columns=names)
```

6.1 Investment

6.1.1 Surroundings PCA

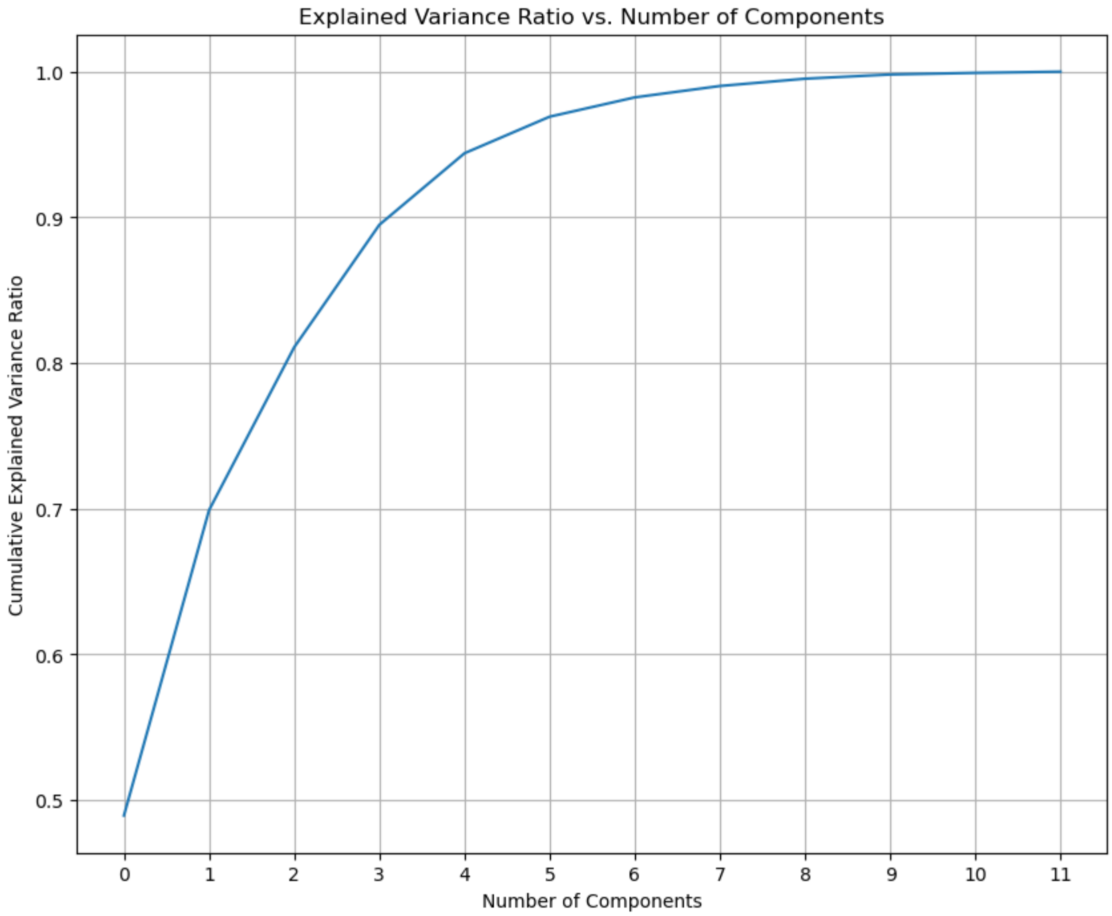
We've separated the surroundings into 2 sub groups:

1. Green + Prom
2. Cafe

Then we have applied PCA to reduce the correlation within each group.

[102]:
```
cafe_surroundings = groups_floats['surroundings'][cafe_invst]
green_surroundings = groups_floats['surroundings'][green_invst]
```

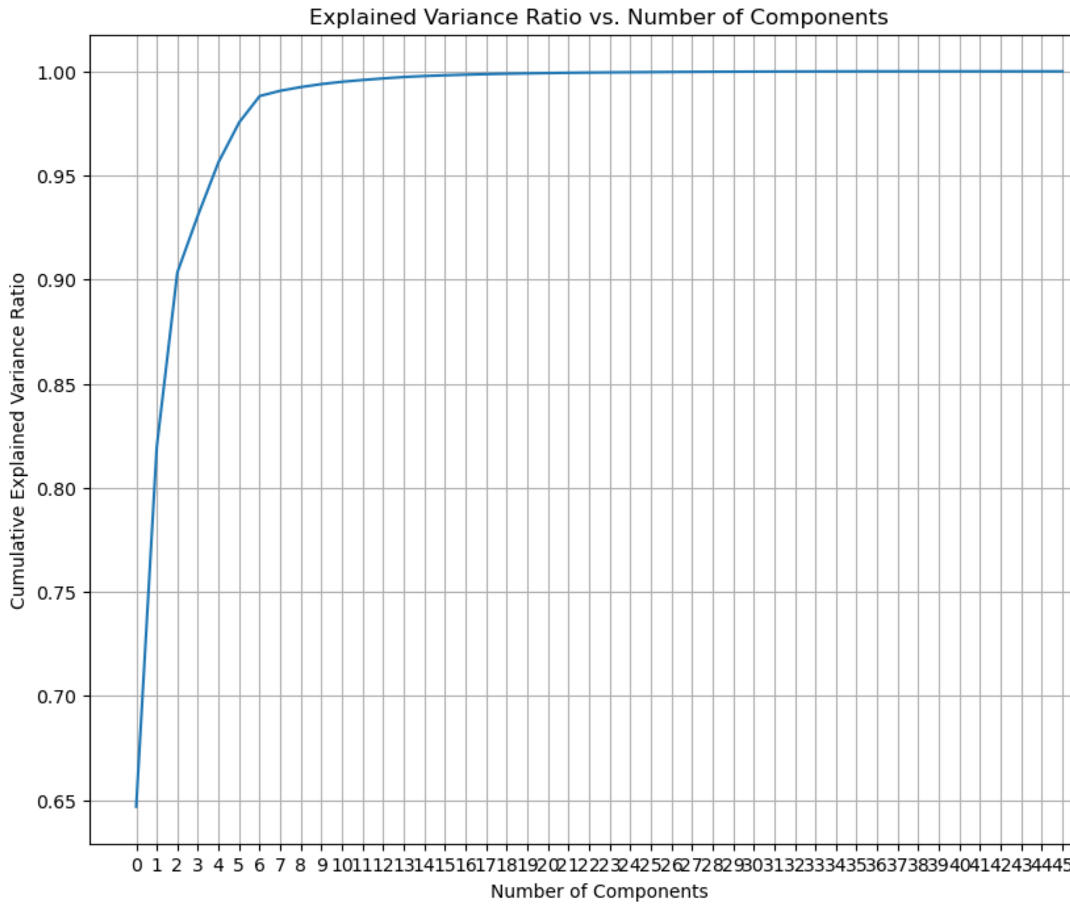
[103]:
```
plotPCA(green_surroundings, 0.95)
```



[103]: 6

[104]:
```
green_pca_invst = pca_df(6, green_surroundings, 'green')
```

[105]:
```
plotPCA(cafe_surroundings, 0.95)
```

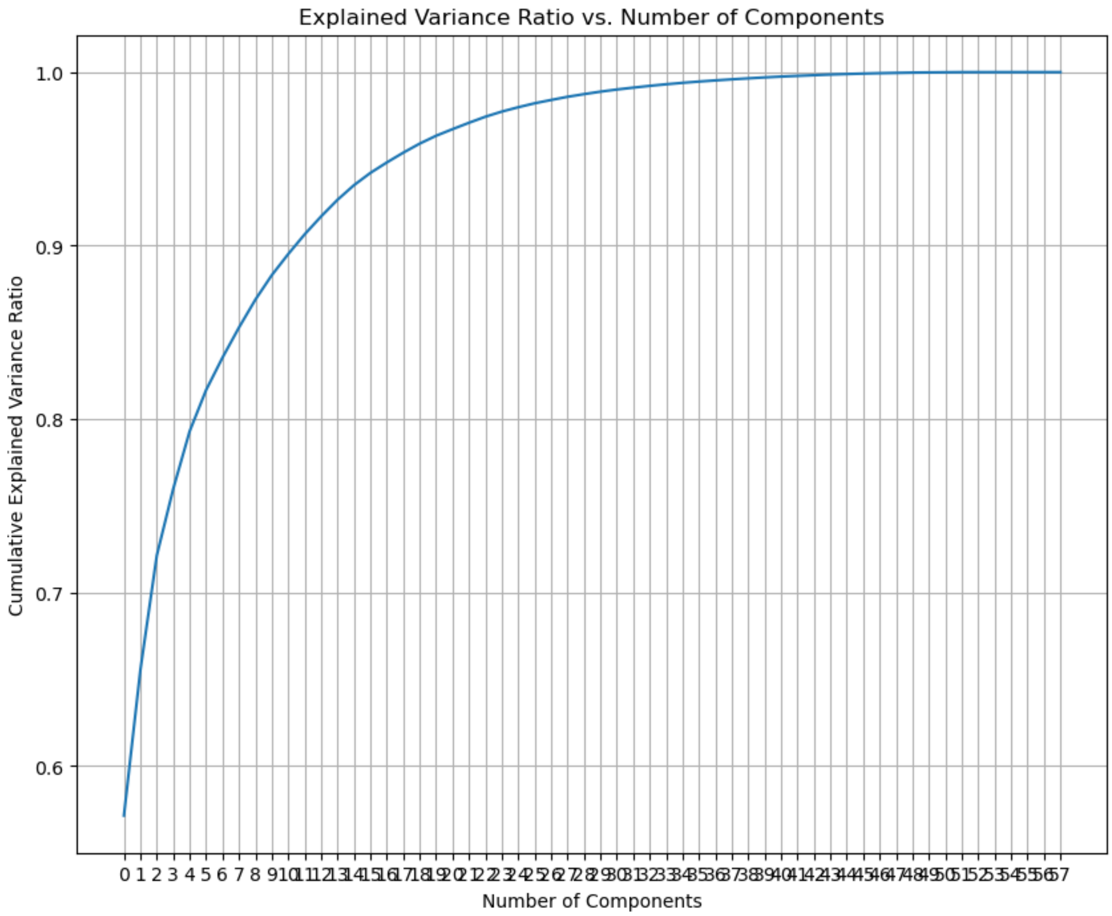


[105]: 5

[106]:
```
cafe_pca_invst = pca_df(5, cafe_surroundings, 'cafe')
```

6.1.2 Distances PCA

[107]:
```
distances = groups_floats['distances']

plotPCA(distances, 0.95)
```

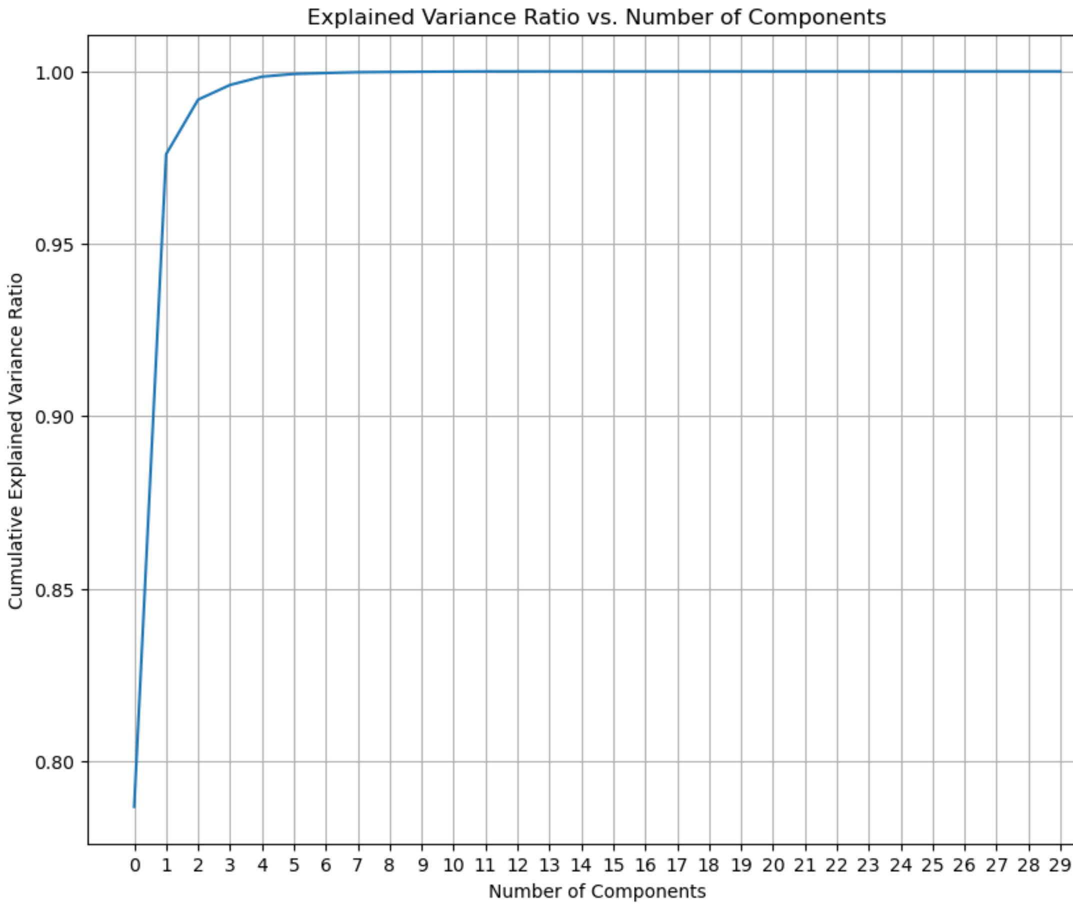


[107]: 18

[108]:
```
distances_pca_invst = pca_df(18,distances, 'distances')
```

6.1.3 Demographics PCA

[109]:
```
demographics = groups_floats['demographics']
plotPCA(demographics, 0.95)
```



[109]: 2

[110]:
```
demographics_pca_invst = pca_df(2,demographics,'demographic')
```

[111]:
```
# Combine the pca data frames:

# Demographics, Surroundings, Distances
pca_df_inv = pd.concat([cafe_pca_invst, green_pca_invst,␣
 ↪demographics_pca_invst,distances_pca_invst],axis=1)
surr_floats = list(groups_dfs['surroundings'].select_dtypes("float64").columns)
demographics = list(groups_dfs['demographics'].filter(like='dens').columns)
distances = list(groups_dfs['distances'].select_dtypes('float64').columns)
new_inv = investment.drop(columns=surr_floats+demographics+distances).
 ↪reset_index(drop=True)
new_inv[pca_df_inv.columns] = pca_df_inv
```

6.2 OwnerOcuppier

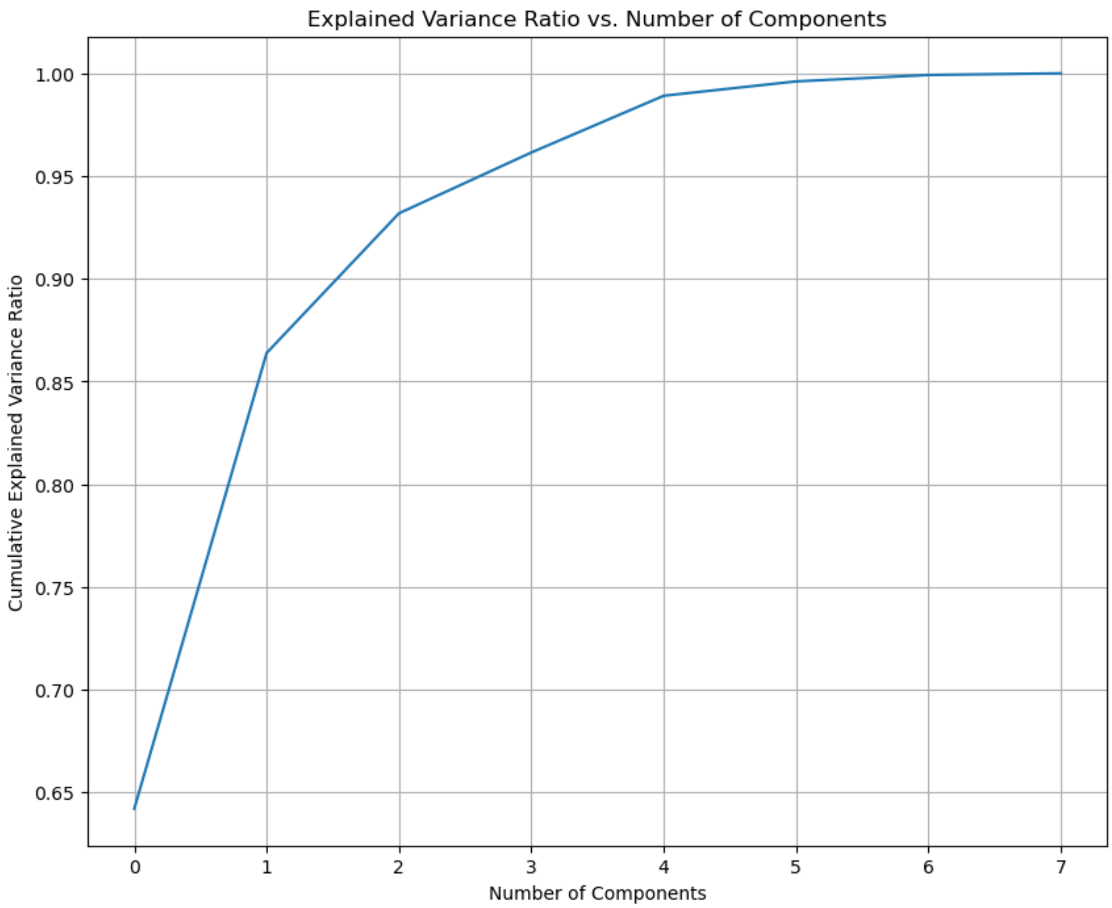
6.2.1 Surroundings PCA

We've separated the surroundings into 2 sub groups:

1. Green + Prom
2. Cafe

Then we have applied PCA to reduce the correlation within each group.

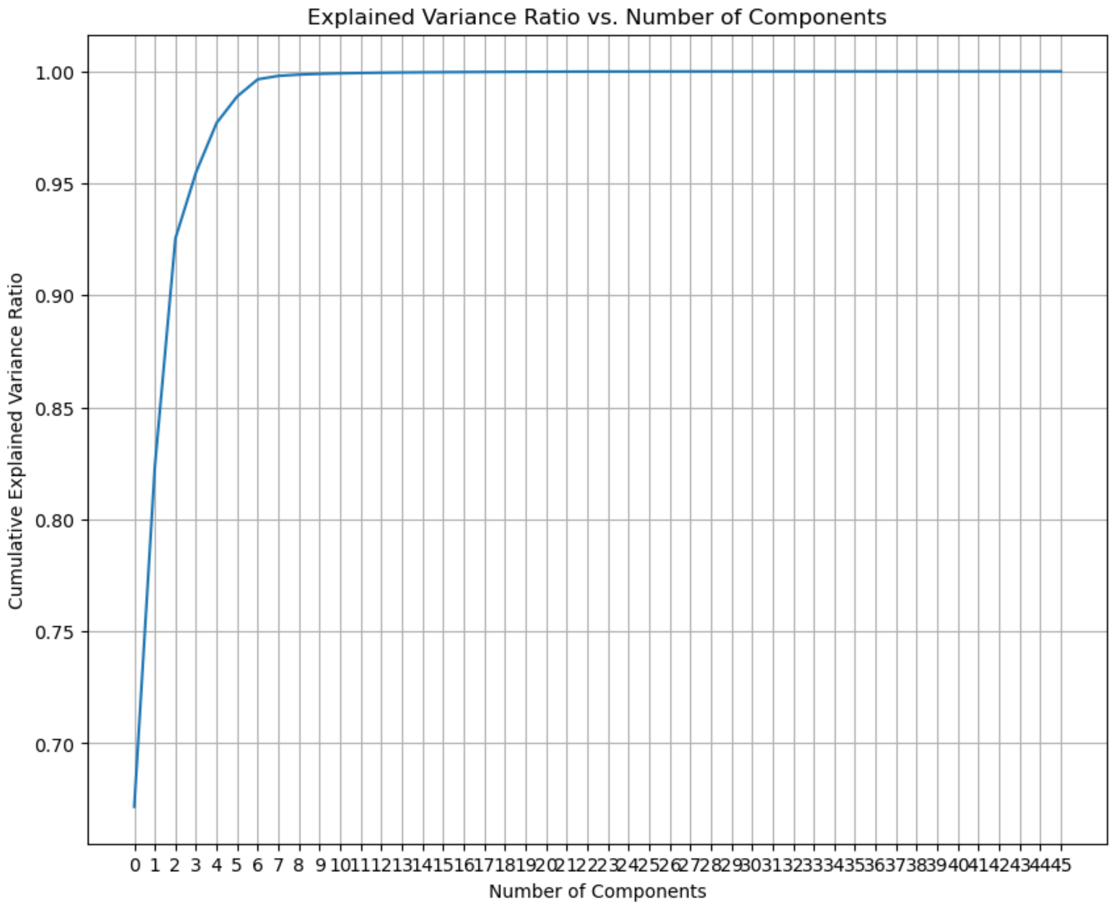
[112]:
```
plotPCA(green_prom_ocu, 0.95)
```



[112]: 4

[113]:
```
green_prom_pca_ocu = pca_df(4, green_prom_ocu, 'green_prom')
```

[114]:
```
plotPCA(cafe_ocu, 0.95)
```

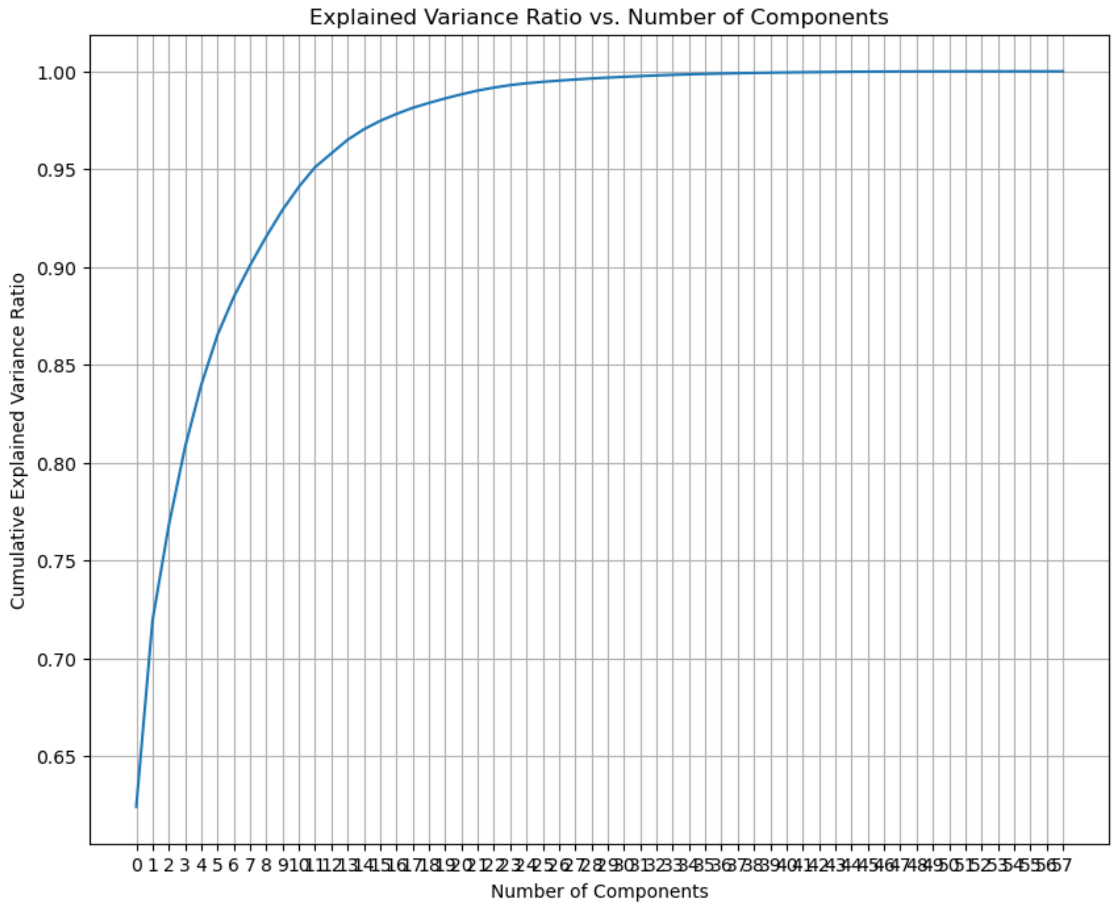


[114]: 4

[115]:
```
cafe_ocu_pca = pca_df(4, cafe_ocu, 'cafe')
```

6.2.2 Distances PCA

[116]:
```
distances_ocu = ocuppier[groups_dfs['distances'].columns].
  ↪select_dtypes('float64')

plotPCA(distances_ocu, 0.95)
```

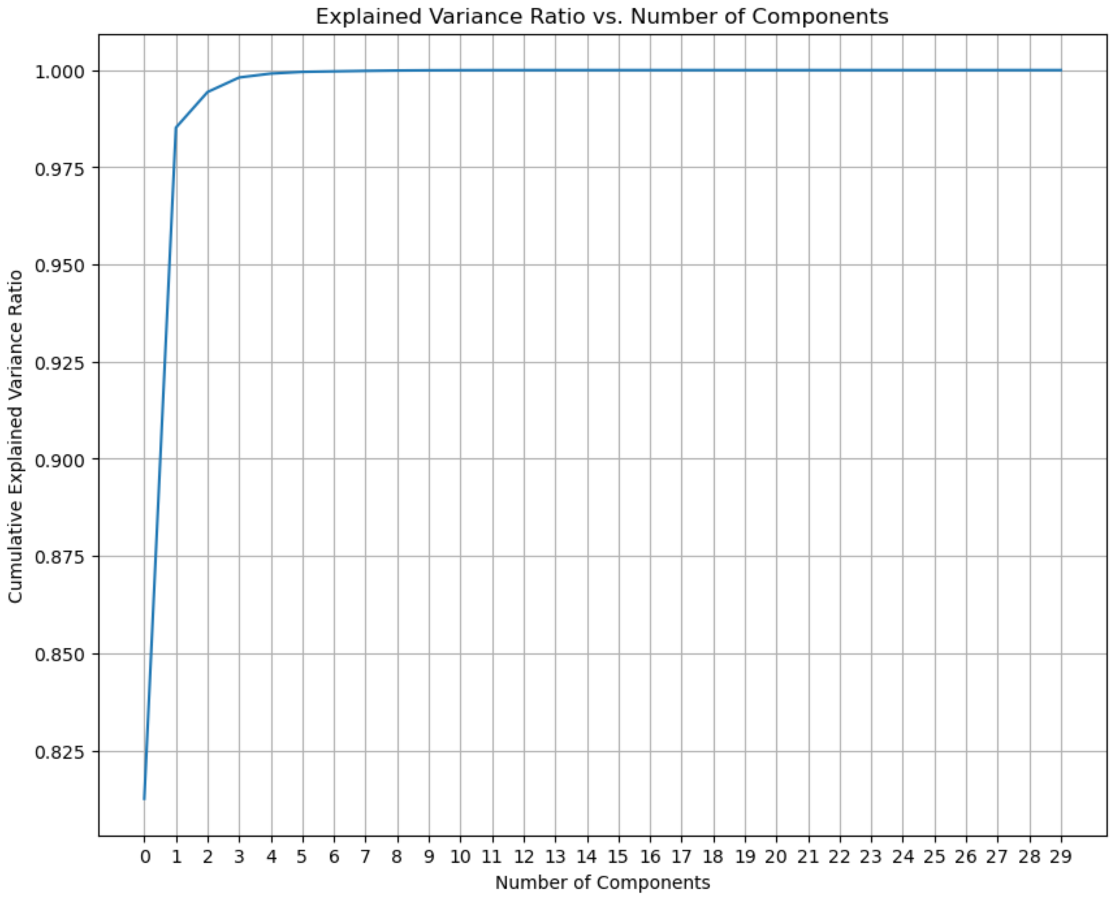


[116]: 12

[117]:
```
distances_pca_ocu = pca_df(12,distances_ocu, 'distances')
```

6.2.3 Demographics PCA

[118]:
```
demographics_ocu = ocuppier[groups_dfs['demographics'].filter(like='dens').
  ↪columns]
plotPCA(demographics_ocu, 0.95)
```



[118]: 2

[119]:
```
demographics_pca_ocu = pca_df(2,demographics_ocu,'demographic')
```

[120]:
```
# Combine the pca data frames:

# Demographics, Surroundings, Distances

pca_df_ocu = pd.concat([cafe_ocu_pca, green_prom_pca_ocu,␣
↪demographics_pca_ocu,distances_pca_ocu],axis=1)
surr_floats = list(groups_dfs['surroundings'].select_dtypes("float64").columns)
demographics = list(groups_dfs['demographics'].filter(like='dens').columns)
distances = list(groups_dfs['distances'].select_dtypes('float64').columns)
new_ocu = ocuppier.drop(columns=surr_floats+demographics+distances).
↪reset_index(drop=True)
new_ocu[pca_df_ocu.columns] = pca_df_ocu
```

7 Preparing Data For Model Building

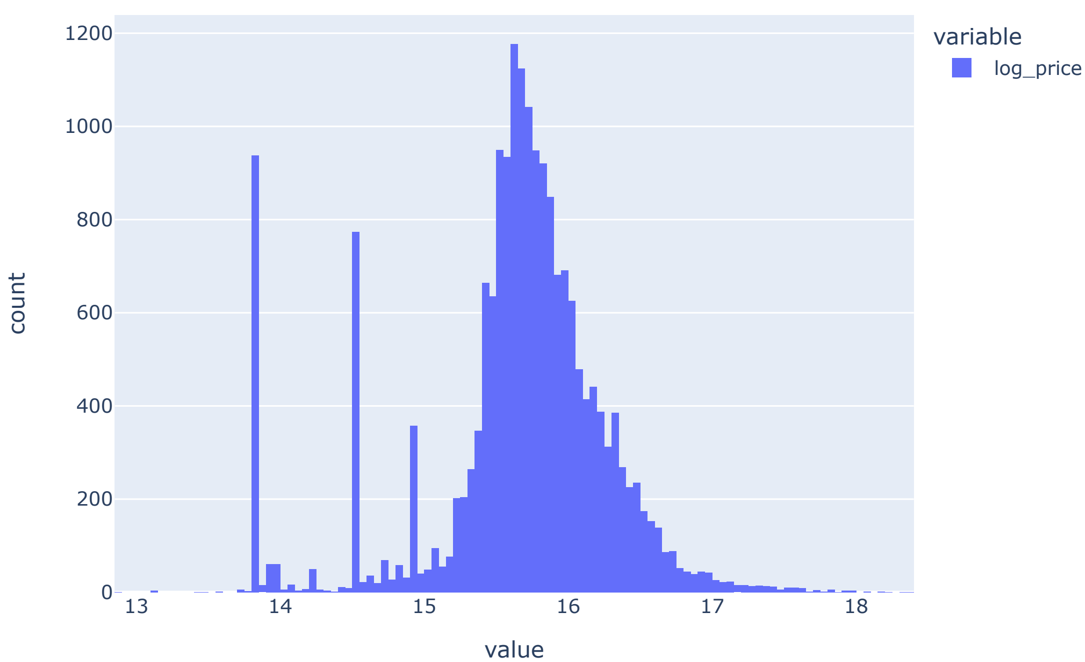
[121]:
```
# Part 5
new_inv = new_inv[(new_inv.price_sq <= 600000) & (new_inv.price_sq >= 10000) |␣
↪new_inv.price_sq.isnull()] # reduces errors...
new_ocu = new_ocu[(new_ocu.price_sq <= 600000) & (new_ocu.price_sq >= 10000) |␣
↪new_ocu.price_sq.isnull()] # reduces errors...
```

7.1 Investment

Before building and tuning the models we first need to handle some bad prices values in our investment data.

RECALL:

[122]:
```
px.histogram(new_inv['log_price'])
```

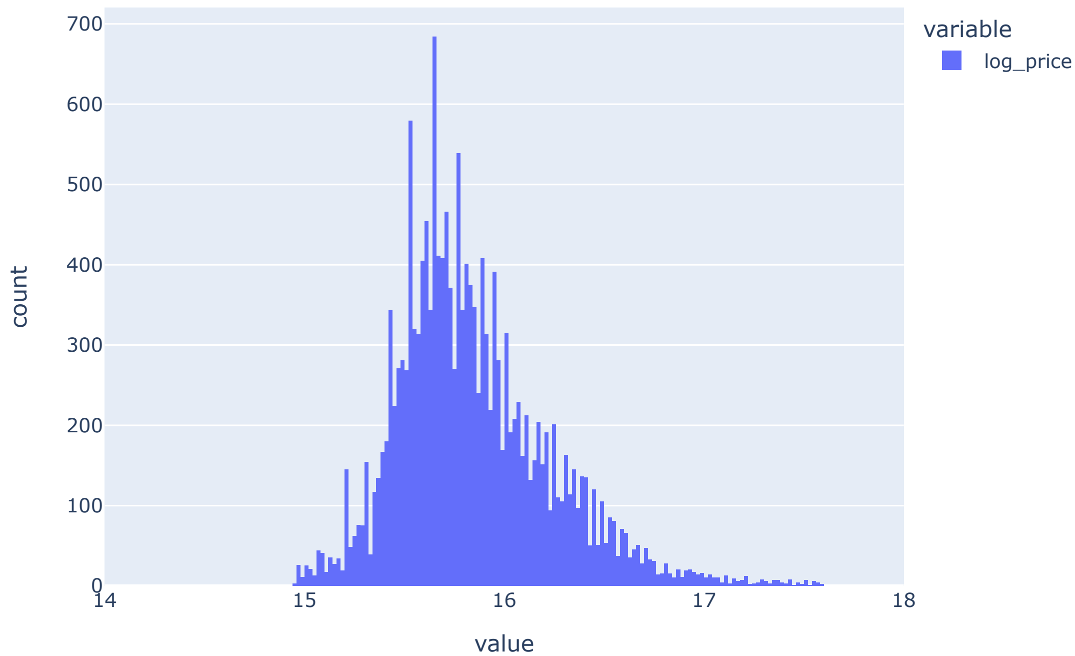


Bad Values:

1. 14.9 - 14.95
2. 14.5 - 14.55
3. 13.8 - 13.85

[123]:

```
fig = px.histogram(new_inv[~(((new_inv['log_price']>=12) &
↪(new_inv['log_price'] <= 14.95)) | (new_inv['log_price'] > 17.
↪6))]['log_price'])
fig.update_layout(xaxis_range = [14,18])
```



Seems a little bit better…

[124]:

```
inv = new_inv[~(((new_inv['log_price']>=12) & (new_inv['log_price'] <= 14.95))
↪| (new_inv['log_price'] > 17.6))].set_index('timestamp').drop(columns = 'id')
```

[125]:

```
cat = inv.select_dtypes('object').columns
inv[cat] = inv[cat].astype('int')
response_vars = inv.columns[inv.columns.
↪isin(['price_doc','price_sq','log_price'])] # get response variables
inv_response_values = inv[inv.columns[inv.columns.isin(response_vars)]]
inv_features = inv[inv.columns[~inv.columns.isin(inv_response_values.columns)]]
↪# get features
```

7.2 OwnerOcuppier

[126]:
```
cat = new_ocu.select_dtypes('object').columns
new_ocu[cat] = new_ocu[cat].astype('int')
ocu = new_ocu.set_index('timestamp').drop(columns = 'id')
response_vars = ocu.columns[ocu.columns.
 ↪isin(['price_doc','price_sq','log_price'])] # get response variables
ocu_response_values = ocu[ocu.columns[ocu.columns.isin(response_vars)]]
ocu_features = ocu[ocu.columns[~ocu.columns.isin(ocu_response_values.columns)]]␣
 ↪# get features
```

[127]:
```
# Splitting data - train.csv, test.csv
ocu_test = ocu[ocu.price_doc.isnull()].drop(columns=ocu_response_values.columns)
inv_test = inv[inv.price_doc.isnull()].drop(columns=inv_response_values.columns)
ocu_train = ocu[~ocu.price_doc.isnull()]
inv_train = inv[~inv.price_doc.isnull()]
```

[128]:
```
#  Get response variables values
ocu_price = ocu_train.price_doc
ocu_logprice = ocu_train.log_price
ocu_pricesq = ocu_train.price_sq

inv_price = inv_train.price_doc
inv_logprice = inv_train.log_price
inv_pricesq = inv_train.price_sq
```

[129]:
```
final_ocu_train = ocu_train.drop(columns = ['price_doc','log_price','price_sq'])
final_inv_train = inv_train.drop(columns = ['price_doc','log_price','price_sq'])
```

[130]:
```
from_float_to_int =␣
 ↪['material','state','build_year','floor','max_floor','num_room','hospital_beds_raion','floo
 ↪+ list(groups_dfs['buildings'].columns) + list(full_data.
 ↪filter(like='missing').columns)
```

[131]:
```
final_inv_train[from_float_to_int] = final_inv_train[from_float_to_int].
 ↪astype(int)
final_ocu_train[from_float_to_int] = final_ocu_train[from_float_to_int].
 ↪astype(int)
```

8 Model Building and Hyperparameter Tuning

There are several methods for hyperparameter tuning so in order to choose which method to use we read about the ones we know. While researching for reading materials we encountered bayesian optimization method for tuning. (Putatunda, S. et al) Showed that hyperopt (a library for bayesian optimization in python) gave the best results in terms of time complexitiy and minimizing the loss function compared to random seach and grid search thus we decided to use it in our project as well.

[238]:
```
def trainTestSplit(x,y):
    x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3,␣
↪random_state=22)
    x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=0.
↪5, random_state=22)
    return x_train, x_val, x_test, y_train, y_val, y_test
```

[239]:
```
ocu_x_train, ocu_x_val, ocu_x_test, ocu_y_train, ocu_y_val, ocu_y_test =␣
↪trainTestSplit(final_ocu_train, ocu_logprice)
inv_x_train, inv_x_val, inv_x_test, inv_y_train, inv_y_val, inv_y_test =␣
↪trainTestSplit(final_inv_train, inv_logprice)
```

[240]:
```
def plotImportance(n_features, importance, features, model,Product):
    sorted_imp = importance.argsort()[::-1]
    df_imp = pd.DataFrame(dict(features = features[sorted_imp], imp =␣
↪importance[sorted_imp]))
    fig = px.bar(df_imp[:n_features], y='features', x='imp')
    fig.update_layout(title = f'Feature Importance - {model} {Product}')
    fig.show()
    print(features[importance.argmax()])
```

8.1 XGBoost

We created an objective function for each type of apartments.

[241]:
```
def xgb_inv_tuning_allfeatures(parameters):
    model = XGBRegressor(**parameters)
    evaluation = [(inv_x_train,inv_y_train),(inv_x_val, inv_y_val)]
    model.fit(inv_x_train,inv_y_train,eval_set=evaluation,verbose=False)
    preds = model.predict(inv_x_val)
    rmse = mean_squared_error(inv_y_val, preds, squared=False)
    print("Score:",rmse)
    return {'loss':rmse,'status':STATUS_OK,'model':model}
```

[242]:
```
def xgb_ocu_tuning_allfeatures(parameters):
    model = XGBRegressor(**parameters)
    evaluation = [(ocu_x_train,ocu_y_train),(ocu_x_val, ocu_y_val)]
    model.fit(ocu_x_train, ocu_y_train,eval_set=evaluation,verbose=False)
    preds = model.predict(ocu_x_val)
    rmse = mean_squared_error(ocu_y_val, preds, squared=False)
    print("Score:",rmse)
    return {'loss':rmse,'status':STATUS_OK,'model':model}
```

we defined xgboost parameter space as (Kapoor & Perrone, 2021) did

[243]:
```
xgb_parameters = {'max_depth':hp.randint("max_depth", 2,8),
                  'eta':hp.loguniform('eta',np.log((1/10)**3), np.log(1)),
                  'colsample_bytree': hp.uniform('colsample_bytree',0.3,1),
```

```
                    'subsample': hp.uniform('subsample',0.5,1),
                    'reg_lambda': hp.loguniform('reg_lambda',np.log((1/10)**6),np.
↪log(20)),
                    'reg_alpha': hp.loguniform('reg_alpha',np.log((1/10)**6),np.
↪log(20)),
                    'gamma': hp.loguniform('gamma', np.log((1/10)**6),np.log(64)),
                    'n_estimators': hp.randint('n_estimators',100,1024),
                    'eval_metric':'rmse',
                    'objective':'reg:squarederror'
                    }
```

[244]:
```
xgb_trials_ocu = Trials()
```

[245]:
```
xgb_trials_inv = Trials()
```

[246]:
```
xgb_ocu_best_parameters = fmin(
    fn=xgb_ocu_tuning_allfeatures,
    space = xgb_parameters,
    algo=tpe.suggest,
    max_evals = 50,
    trials=xgb_trials_ocu
)

print(xgb_ocu_best_parameters)
```

```
Score:
0.1058841780318957
Score:
0.11760002354235286
Score:
0.11243087575672941
Score:
0.24060875364947543
Score:
0.24511412104881367
Score:
0.14878641873855952
Score:
0.10953223382234713
Score:
0.1835038071362695
Score:
0.12745476681769424
Score:
0.29803440732223524
Score:
0.23810987577761583
Score:
```

```
0.15848260810087186
Score:
0.18837042037773216
Score:
0.2813977395594886
Score:
0.19923688360831746
Score:
0.11654816048142522
Score:
0.12395860184795071
Score:
0.37200025166924333
Score:
0.12089973907258454
Score:
0.21334093942880827
Score:
0.1112088978961913
Score:
0.12312344756530295
Score:
0.11236894022987384
Score:
0.11335735138611501
Score:
0.19391468129846934
Score:
0.13596687735197857
Score:
0.20883406337506294
Score:
0.11510706838921458
Score:
0.1077578131701131
Score:
0.10978600129542317
Score:
0.11415945882246847
Score:
0.11503978225490993
Score:
0.11668929415623612
Score:
0.10830222249519537
Score:
0.10895546970934925
Score:
```

```
0.127058205756004
Score:
0.11613781442191491
Score:
0.2303979501671179
Score:
0.12039804630252057
Score:
0.12257767236801981
Score:
0.11837344983528544
Score:
0.11335939746853689
Score:
0.17798761424432913
Score:
0.13528101929824526
Score:
0.1518207944514198
Score:
0.11645348245217892
Score:
0.11375553251210391
Score:
0.13601846248070645
Score:
0.12151958031341414
Score:
0.13146835281146416
100%|      | 50/50 [01:41<00:00,  2.03s/trial, best loss: 0.1058841780318957]
{'colsample_bytree': 0.9372904286503072, 'eta': 0.1623697585579469, 'gamma':
0.0008233655894597945, 'max_depth': 6, 'n_estimators': 134, 'reg_alpha':
0.21573418575839792, 'reg_lambda': 0.0005802138188644974, 'subsample':
0.5570964152767883}
```

[247]:
```
best_model_ocu = xgb_trials_ocu.results[np.argmin([r['loss'] for r in
↪xgb_trials_ocu.results if 'loss' in r])]['model'] # gets the best model
```

[248]:
```
top_models = 5 # get the top 5 models... we will use average on the predictions.
↪.
```

[249]:
```
top_xgb_ocu = sorted(xgb_trials_ocu.results, key= lambda x: x['loss'] if 'loss'
↪in x else 9999)[:top_models] # gets the top 5 models
```

[250]:
```
xgb_inv_best_parameters = fmin(
    fn=xgb_inv_tuning_allfeatures,
    space = xgb_parameters,
    algo=tpe.suggest,
```

```
    max_evals = 50,
    trials=xgb_trials_inv
)

print(xgb_inv_best_parameters)
```

```
Score:
0.2644836297439095
Score:
0.237984461666739
Score:
0.210273106002578
Score:
0.1913569946088281
Score:
0.20951332148997517
Score:
0.28066070348554417
Score:
0.22007190133446491
Score:
0.19764589327468923
Score:
0.1913246020769075
Score:
0.24152545107071222
Score:
0.20332354595408839
Score:
0.19377021365863262
Score:
0.19011617733955652
Score:
0.19446846784182495
Score:
0.1891460588945189
Score:
0.20454553558655522
Score:
0.3287878977157227
Score:
0.19548133299062423
Score:
0.21539542026975106
Score:
0.3376580378296252
Score:
0.19929156283175697
```

```
Score:
0.190075905765267
Score:
0.18874587798440692
Score:
0.1881638907524976
Score:
0.188457930114363
Score:
0.2755012072676936
Score:
0.19089195515664698
Score:
0.27687410051956995
Score:
0.18979205129823307
Score:
0.20307321240006596
Score:
0.1922816831163264
Score:
0.28715084202275964
 64%|      | 32/50 [01:58<01:30,  5.02s/trial, best loss:
0.1881638907524976][CV 2/5] END gamma=0.4, learning_rate=0.01, max_depth=3,
n_estimators=500;, score=-5127318584929.354 total time=   1.4s
[CV 4/5] END gamma=0.4, learning_rate=0.01, max_depth=8, n_estimators=600;,
score=-4317272443258.196 total time=   2.9s
[CV 2/5] END gamma=0.0, learning_rate=0.01, max_depth=7, n_estimators=700;,
score=-4192620222329.378 total time=   2.9s
[CV 4/5] END gamma=0.4, learning_rate=0.01, max_depth=3, n_estimators=500;,
score=-5232050735579.723 total time=   1.5s
[CV 5/5] END gamma=0.4, learning_rate=0.01, max_depth=8, n_estimators=600;,
score=-4580326896855.904 total time=   3.0s
[CV 3/5] END gamma=0.0, learning_rate=0.01, max_depth=7, n_estimators=700;,
score=-4508570745843.917 total time=   2.8s
[CV 1/5] END gamma=0.4, learning_rate=0.01, max_depth=3, n_estimators=500;,
score=-5164109389688.033 total time=   1.5s
[CV 3/5] END gamma=0.4, learning_rate=0.01, max_depth=6, n_estimators=900;,
score=-4504080967896.182 total time=   3.1s
[CV 5/5] END gamma=0.0, learning_rate=0.01, max_depth=7, n_estimators=700;,
score=-4595175520303.857 total time=   2.8s
[CV 1/5] END gamma=0.4, learning_rate=0.01, max_depth=8, n_estimators=600;,
score=-4321104468889.524 total time=   3.0s
[CV 1/5] END gamma=0.0, learning_rate=0.01, max_depth=7, n_estimators=700;,
score=-4331478831631.282 total time=   3.0s
[CV 2/5] END gamma=0.0, learning_rate=0.01, max_depth=3, n_estimators=500;,
score=-5127318584929.354 total time=   1.4s
[CV 3/5] END gamma=0.4, learning_rate=0.01, max_depth=3, n_estimators=500;,
```

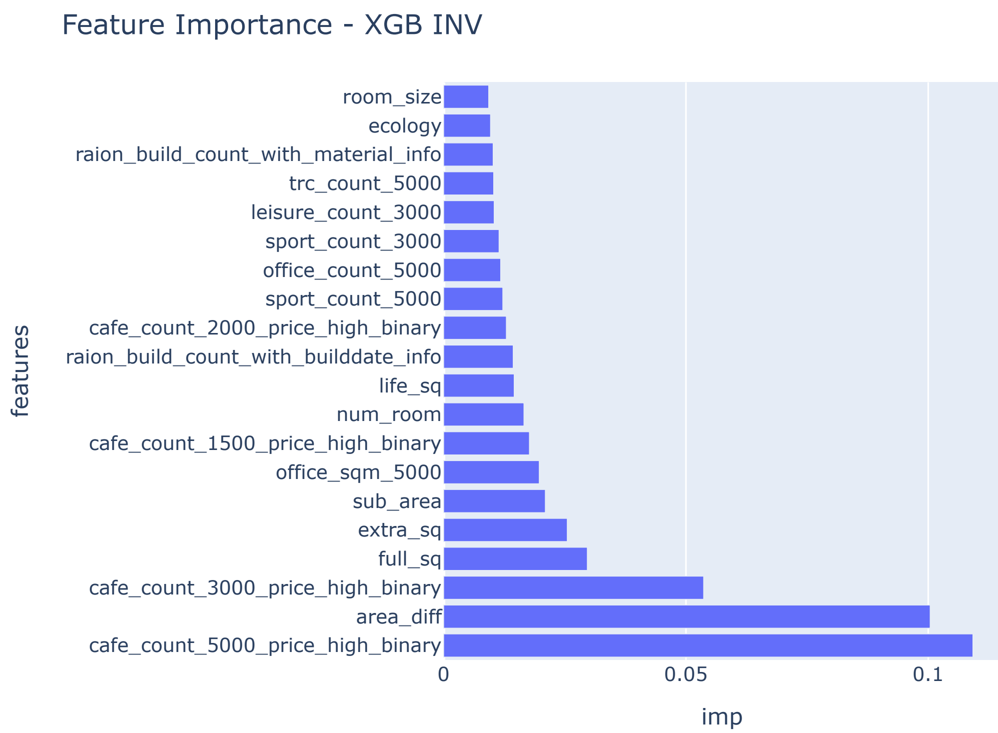
```
score=-5360790269415.222 total time=   1.5s
[CV 1/5] END gamma=0.4, learning_rate=0.01, max_depth=6, n_estimators=900;,
score=-4303690138607.251 total time=   3.2s
[CV 4/5] END gamma=0.0, learning_rate=0.01, max_depth=7, n_estimators=700;,
score=-4312570188166.866 total time=   2.8s
[CV 5/5] END gamma=0.4, learning_rate=0.01, max_depth=3, n_estimators=500;,
score=-5426181289294.336 total time=   1.5s
[CV 2/5] END gamma=0.4, learning_rate=0.01, max_depth=6, n_estimators=900;,
score=-4191622197420.055 total time=   3.2s
[CV 1/5] END gamma=0.0, learning_rate=0.01, max_depth=3, n_estimators=500;,
score=-5164109389688.033 total time=   1.5s
[CV 3/5] END gamma=0.0, learning_rate=0.01, max_depth=3, n_estimators=500;,
score=-5360790269415.222 total time=   1.3s
[CV 2/5] END gamma=0.4, learning_rate=0.01, max_depth=8, n_estimators=600;,
score=-4176254383141.411 total time=   3.0s
[CV 5/5] END gamma=0.4, learning_rate=0.01, max_depth=6, n_estimators=900;,
score=-4613396917786.215 total time=   3.2s
[CV 4/5] END gamma=0.0, learning_rate=0.01, max_depth=3, n_estimators=500;,
score=-5232050735579.723 total time=   1.3s
[CV 3/5] END gamma=0.4, learning_rate=0.01, max_depth=8, n_estimators=600;,
score=-4488427717458.747 total time=   2.9s
[CV 4/5] END gamma=0.4, learning_rate=0.01, max_depth=6, n_estimators=900;,
score=-4321196746451.036 total time=   3.2s
[CV 5/5] END gamma=0.0, learning_rate=0.01, max_depth=3, n_estimators=500;,
score=-5426181289294.336 total time=   1.3s
Score:
0.19445228859836627
Score:
0.19794270711307135
Score:
0.19028979628278397
Score:
0.22037142354119835
Score:
0.19148575567196613
Score:
0.19284961295365233
Score:
0.22837141806007372
Score:
0.19147764565405398
Score:
0.22101366067907519
Score:
0.2704537532194982
Score:
0.22802909721355305
Score:
```

```
0.21628700098937664
Score:
0.1918366088728885
Score:
0.1893375517304326
Score:
0.21708204548126495
Score:
0.1906910116265093
Score:
0.2410008190273595
Score:
0.18866423954997255
100%|     | 50/50 [03:01<00:00,  3.62s/trial, best loss: 0.1881638907524976]
{'colsample_bytree': 0.4017398986910317, 'eta': 0.014062305138921366, 'gamma':
1.5298470665441234e-05, 'max_depth': 7, 'n_estimators': 698, 'reg_alpha':
0.0013533958007616535, 'reg_lambda': 9.091311069282547e-06, 'subsample':
0.9522562407615086}
```

[251]:
```
best_model_inv = xgb_trials_inv.results[np.argmin([r['loss'] for r in␣
↪xgb_trials_inv.results if 'loss' in r ])]['model']
```

[252]:
```
top_xgb_inv = sorted(xgb_trials_inv.results, key= lambda x: x['loss'] if 'loss'␣
↪in x else 9999)[:top_models]
```

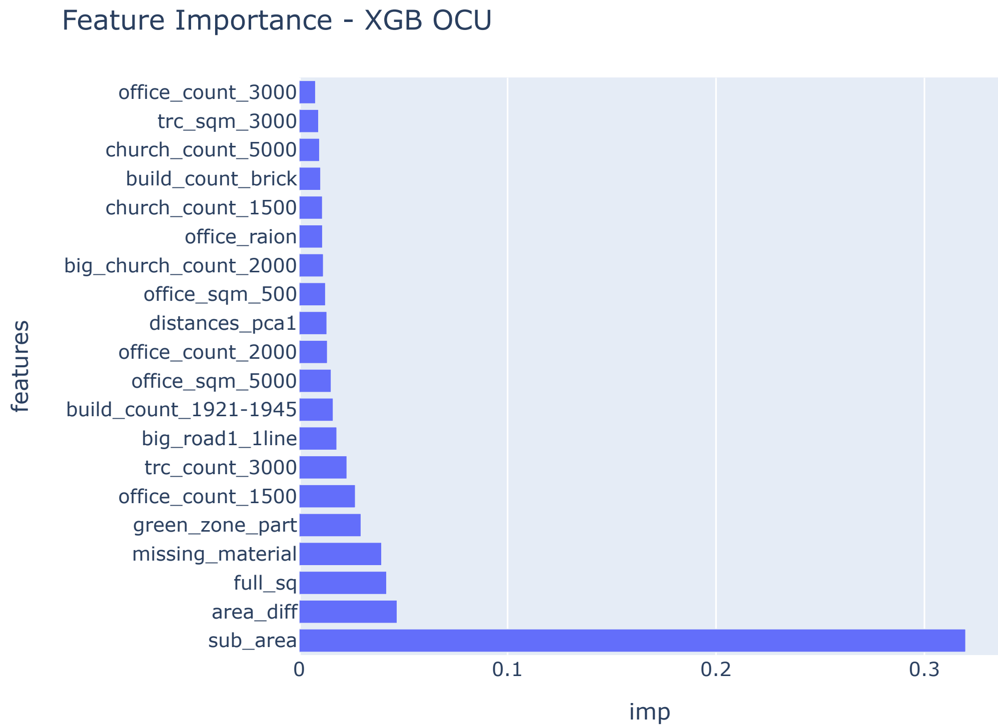
[253]:
```
plotImportance(n_features=20, importance=best_model_inv.feature_importances_,␣
↪features=inv_x_train.columns, model="XGB", Product="INV")
```




```
cafe_count_5000_price_high_binary
```

[254]:

```
plotImportance(n_features=20, importance=best_model_ocu.feature_importances_,
    features= ocu_x_train.columns, model="XGB", Product="OCU")
```




```
sub_area
```

8.2 Random Forest

[255]:
```
rf_parameters = {
    'n_estimators': hp.choice('n_estimators', np.arange(50,1050,50)),
    'max_features':hp.choice('max_features', ['log2','sqrt']),
    'criterion': hp.choice('criterion', ['friedman_mse', 'squared_error']),
    'max_depth': scope.int(hp.uniform('max_depth', 5,20)),
    'max_samples': hp.uniform('max_samples', 0,0.8),
    'min_samples_leaf': scope.int(hp.uniform('min_samples_leaf',1,5)),
    'min_samples_split': scope.int(hp.uniform('min_samples_split',2,6))
}
```

[256]:
```
def rf_inv_tuning_allfeatures(parameters):
    model = RandomForestRegressor(**parameters)
    model.fit(inv_x_train,inv_y_train)
    preds = model.predict(inv_x_val)
    rmse = mean_squared_error(inv_y_val, preds, squared=False)
    print("Score:",rmse)
    return {'loss':rmse,'status':STATUS_OK,'model':model}
```

[257]:
```
def rf_ocu_tuning_allfeatures(parameters):
    model = RandomForestRegressor(**parameters)
    model.fit(ocu_x_train, ocu_y_train)
    preds = model.predict(ocu_x_val)
    rmse = mean_squared_error(ocu_y_val, preds, squared=False)
    print("Score:",rmse)
    return {'loss':rmse,'status':STATUS_OK,'model':model}
```

[258]:
```
rf_trials_inv = Trials()
```

[259]:
```
rf_inv_best_parameters = fmin(
    fn=rf_inv_tuning_allfeatures,
    space = rf_parameters,
    algo=tpe.suggest,
    max_evals = 20,
    trials=rf_trials_inv
)
print(rf_inv_best_parameters)
```

```
Score:
0.22299355804043655
Score:
0.25935196368644237
Score:
0.21612811558531436
Score:
0.2089327336511747
Score:
0.26499524273401126
Score:
0.20532434921530165
Score:
0.2633114956032697
Score:
0.2252921786079209
Score:
0.20292189754500922
Score:
0.2380140243122244
Score:
0.2821369101963713
Score:
0.20042785505870644
Score:
0.21054567611959404
Score:
0.21399065674874168
Score:
```

```
0.2311431083680959
Score:
0.25176518915744345
Score:
0.21538544204616264
Score:
0.26868475006811027
Score:
0.24079665126419023
Score:
0.21764865318343715
100%|     | 20/20 [02:40<00:00,  8.04s/trial, best loss: 0.20042785505870644]
{'criterion': 0, 'max_depth': 15.874326197235035, 'max_features': 1,
'max_samples': 0.658117314106804, 'min_samples_leaf': 1.8732209888405222,
'min_samples_split': 2.676734090766692, 'n_estimators': 12}
```

[260]:

```
rf_trials_ocu = Trials()
```

[261]:

```
rf_ocu_best_parameters = fmin(
    fn=rf_ocu_tuning_allfeatures,
    space = rf_parameters,
    algo=tpe.suggest,
    max_evals = 20,
    trials=rf_trials_ocu
)
print(rf_ocu_best_parameters)
```

```
Score:
0.18007082690660559
Score:
0.18707279774761876
Score:
0.19483020273773885
Score:
0.1329407121782871
Score:
0.1498122550728186
Score:
0.15415476376610596
Score:
0.17535495731279
Score:
0.1565775068465129
Score:
0.15084188954129624
Score:
0.13779421928847332
Score:
```

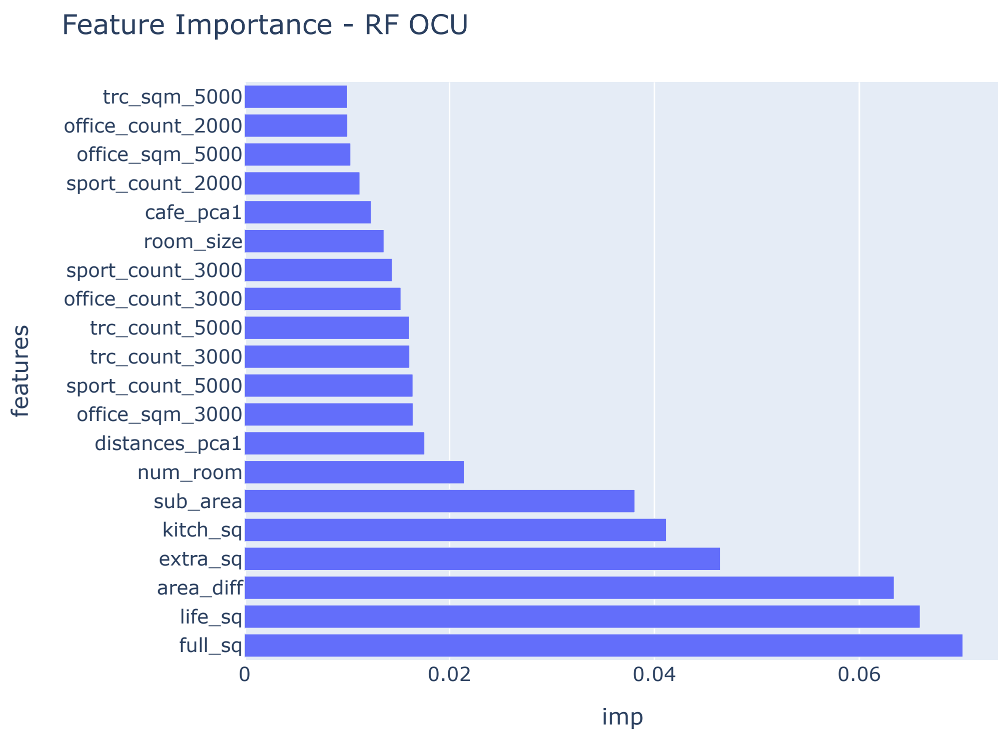
```
0.22146974159414087
Score:
0.16374522672300335
Score:
0.16102651536475218
Score:
0.12958525507623309
Score:
0.17348097021807224
Score:
0.21786415805731146
Score:
0.13635373388878674
Score:
0.14317843043125195
Score:
0.16021760062243592
Score:
0.20190852630409437
100%|     | 20/20 [00:40<00:00,  2.02s/trial, best loss: 0.12958525507623309]
{'criterion': 0, 'max_depth': 15.768681850617753, 'max_features': 0,
'max_samples': 0.7134946183782201, 'min_samples_leaf': 1.7281239019674,
'min_samples_split': 5.03356610318302, 'n_estimators': 14}
```

[262]:

```
best_rf_ocu = rf_trials_ocu.results[np.argmin([r['loss'] for r in rf_trials_ocu.
    ↪results if 'loss' in r])]['model']
best_rf_inv = rf_trials_inv.results[np.argmin([r['loss'] for r in rf_trials_inv.
    ↪results if 'loss' in r])]['model']

top5_rf_ocu = sorted(rf_trials_ocu.results, key= lambda x: x['loss'] if 'loss'␣
    ↪in x else 9999)[:top_models]
top5_rf_inv = sorted(rf_trials_inv.results, key= lambda x: x['loss'] if 'loss'␣
    ↪in x else 9999)[:top_models]
```

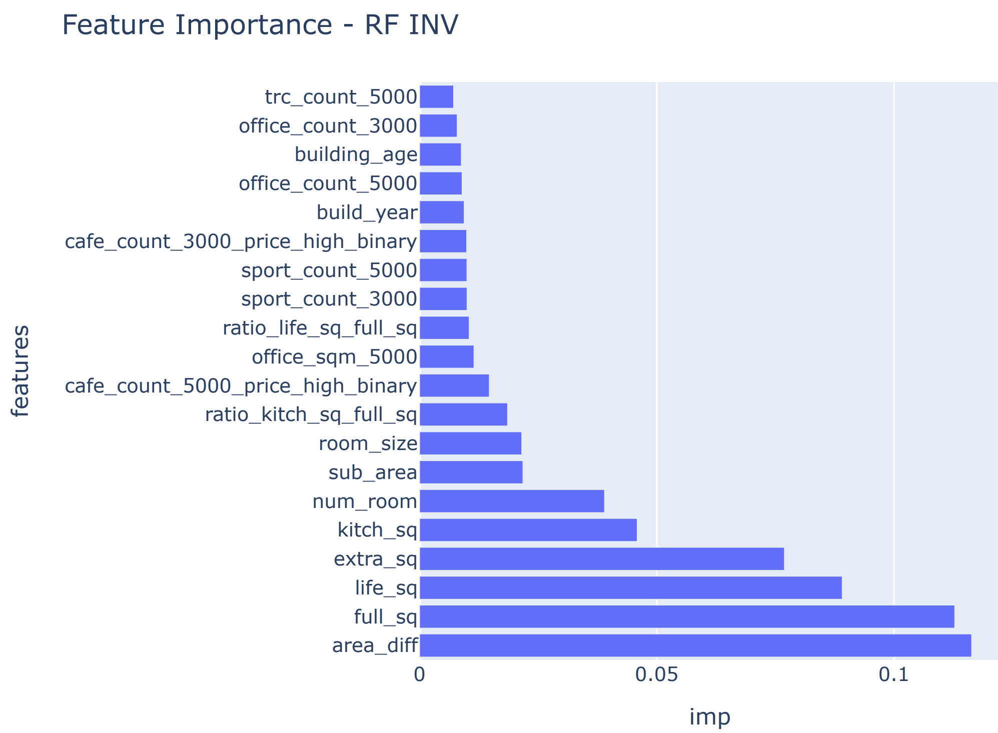
[263]:

```
plotImportance(n_features=20, importance=best_rf_ocu.feature_importances_,␣
    ↪features=ocu_x_train.columns, model="RF", Product="OCU")
```




```
full_sq
```

[264]:
```
plotImportance(n_features=20, importance=best_rf_inv.feature_importances_,
    features=inv_x_train.columns, model="RF", Product="INV")
```



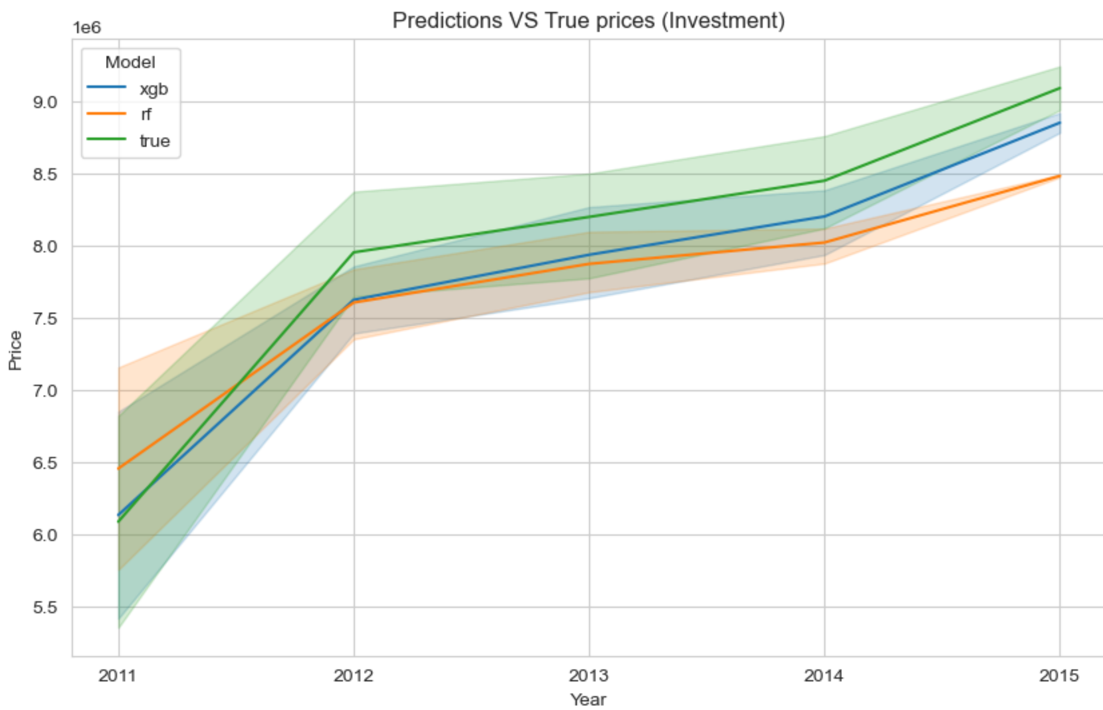

```
area_diff
```

8.3 Test Predictions

[265]:
```
inv_xgb_testpreds = best_model_inv.predict(inv_x_test)
inv_rf_testpreds = best_rf_inv.predict(inv_x_test)

ocu_xgb_testpreds = best_model_ocu.predict(ocu_x_test)
ocu_rf_testpreds = best_rf_ocu.predict(ocu_x_test)

test_inv_preds = pd.DataFrame(dict(year = inv_x_test.year,quarter = inv_x_test.
  ↪quarter,xgb = np.expm1(inv_xgb_testpreds), rf=np.
  ↪expm1(inv_rf_testpreds),true = np.expm1(inv_y_test)))
#test_inv_preds = pd.DataFrame(dict(year = inv_x_test.year,quarter = inv_x_test.
  ↪quarter,xgb = inv_xgb_testpreds*inv_x_test['full_sq'], rf=␣
  ↪inv_rf_testpreds*inv_x_test['full_sq'], true =␣
  ↪inv_y_test*inv_x_test['full_sq']))
test_ocu_preds = pd.DataFrame(dict(year = ocu_x_test.year, quarter = ocu_x_test.
  ↪quarter, xgb = np.expm1(ocu_xgb_testpreds) ,rf=np.expm1(ocu_rf_testpreds),␣
  ↪true = np.expm1(ocu_y_test)))
```

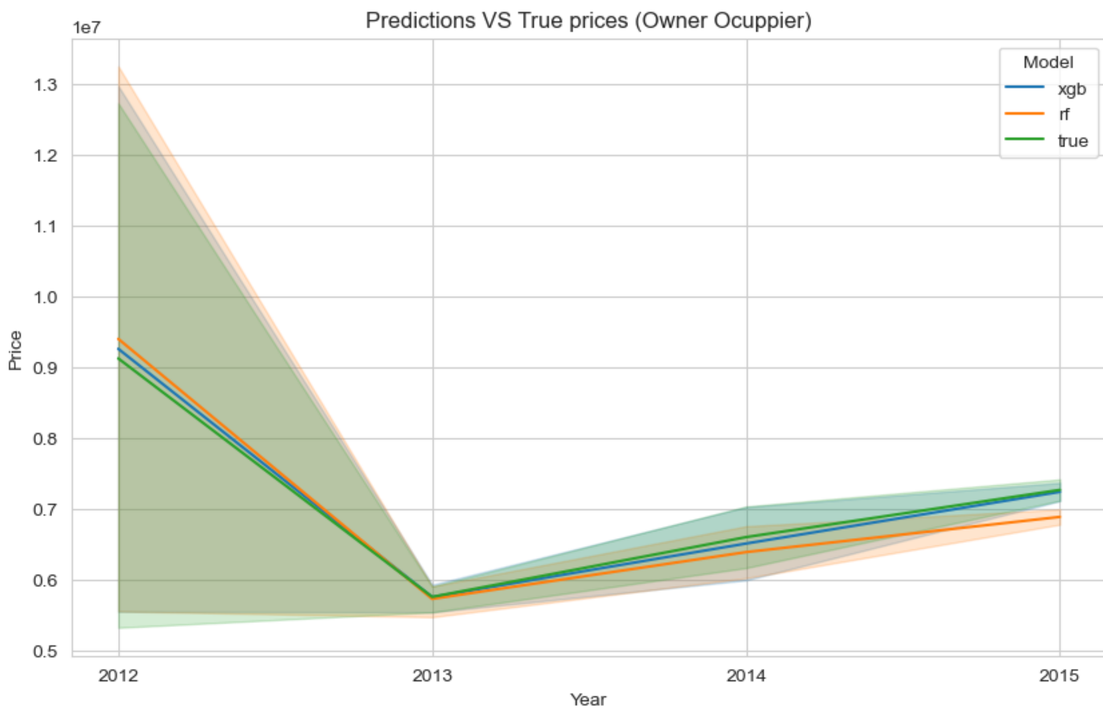
[266]:
```
q = test_inv_preds.groupby(['year','quarter'])
q2 = test_ocu_preds.groupby(['year','quarter'])
```

[267]:
```
melted_q = q.mean().reset_index().melt(id_vars='year',var_name='model',␣
 ↪value_name='prediction')
melted_q['year'] = melted_q['year'].apply(lambda year: str(year))
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))

# Plot the lines for different models
sns.lineplot(data=melted_q[melted_q['model'] !='quarter'], x='year',␣
 ↪y='prediction', hue='model')

plt.xlabel('Year')
plt.ylabel('Price')
plt.title('Predictions VS True prices (Investment)')

plt.legend(title='Model')
plt.show()
```



[268]:
```
melted_q2 = q2.mean().reset_index().melt(id_vars='year',var_name='model',␣
 ↪value_name='prediction')
melted_q2['year'] = melted_q2['year'].apply(lambda year: str(year))
```

```
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))

# Plot the lines for different models
sns.lineplot(data=melted_q2[melted_q2['model'] !='quarter'], x='year',␣
 ↪y='prediction', hue='model')

plt.xlabel('Year')
plt.ylabel('Price')
plt.title('Predictions VS True prices (Owner Ocuppier)')

plt.legend(title='Model')
plt.show()
```



It seems that for both xgboost and random forest models it's easier to predict the prices for the Owner Ocuppier product type.

XGboost - Give better results in both cases.

XGBoost - Builds trees sequentially, optimize a specific objective function.

Random Forest - builds trees independently with no interaction between them.

The sequential nature of xgboost allows it to learn complex patterns as the current tree learns from previous trees mistakes (reduces bias but prone to overfitting) while random forest may capture different pattern due to the randomness during the training phase.

As mentioned above, random forest builds trees independently where each of the trees learn a certain part of the data and at the end the predictions are summarized using average. This method reduces the variance of the model and allows a better generalization to unseen data.

So we believe that the difference between the two models predictions is probably because xgboost can learn complex patterns while a bagging ensemble model (random forest) is good when the model has low bias already.

Though XGBOOST predictions seem to be a little better, we still wanted to try and ensemble them.

9 XGBoost without Hyper Tuning

Though hyper parameter tuning algorithm can help get better results we decided to try choosing the right parameters ourselves. We did some research online, and decided to create 5 parameters sets so we ended up having 5 xgboost models for each product type.

[133]:
```
import xgboost as xgb
def no_tune_train(params, x_train,y_train):
    dtrain = xgb.DMatrix(x_train, y_train)

    boosts = xgb.cv(params, dtrain, num_boost_round = 10000, verbose_eval =␣
 ↪100, early_stopping_rounds = 50)

    xgb_model = xgb.train(params, dtrain,num_boost_round=len(boosts))
    return xgb_model
```

9.1 Log Price

[270]:
```
ocu_xgb_no_tune_params1 = {
    'subsample':0.7,
    'max_depth': 4,
    'colsample_bytree':0.7,
    'lambda': 200,
    'alpha':8,
    'eta': 0.05
}
ocu_xgb_no_tune_params2 = {
    'subsample':1,
    'max_depth':4,
    'colsample_bytree':0.7,
    'lambda': 200,
    'alpha':8,
    'eta':0.05
}

ocu_xgb_no_tune_params3 = {
    'subsample':1,
```

```
    'max_depth':4,
    'colsample_bytree':1,
    'lambda': 200,
    'alpha':8,
    'eta':0.05,
    'nthread':8
}

ocu_xgb_no_tune_params4 = {
    'subsample':0.9,
    'max_depth':4,
    'colsample_bytree':0.7,
    'lambda': 200,
    'alpha':8,
    'eta':0.05
}

ocu_xgb_no_tune_params5 = {
    'subsample':0.8,
    'max_depth':7,
    'colsample_bytree':0.7,
    'lambda': 200,
    'alpha':8,
    'eta':0.05
}
ocu_notue_xgb1 = no_tune_train(params=ocu_xgb_no_tune_params1,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.log_price)
ocu_notue_xgb2 = no_tune_train(params=ocu_xgb_no_tune_params2,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.log_price)
ocu_notue_xgb3 = no_tune_train(params=ocu_xgb_no_tune_params3,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.log_price)
ocu_notue_xgb4 = no_tune_train(params=ocu_xgb_no_tune_params4,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.log_price)
ocu_notue_xgb5 = no_tune_train(params=ocu_xgb_no_tune_params5,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.log_price)
```

```
[0]     train-rmse:0.43817+0.00228      test-rmse:0.43827+0.00482
[100]   train-rmse:0.15363+0.00092      test-rmse:0.16070+0.00237
[200]   train-rmse:0.13024+0.00113      test-rmse:0.14034+0.00156
[300]   train-rmse:0.12142+0.00133      test-rmse:0.13335+0.00159
[400]   train-rmse:0.11649+0.00152      test-rmse:0.12965+0.00158
[500]   train-rmse:0.11323+0.00163      test-rmse:0.12726+0.00158
[600]   train-rmse:0.11071+0.00168      test-rmse:0.12545+0.00170
[700]   train-rmse:0.10888+0.00170      test-rmse:0.12425+0.00172
[800]   train-rmse:0.10744+0.00173      test-rmse:0.12330+0.00187
[900]   train-rmse:0.10623+0.00173      test-rmse:0.12252+0.00191
[1000]  train-rmse:0.10515+0.00172      test-rmse:0.12179+0.00200
```

```
[1100]  train-rmse:0.10424+0.00173    test-rmse:0.12121+0.00206
[1200]  train-rmse:0.10349+0.00176    test-rmse:0.12071+0.00210
[1300]  train-rmse:0.10280+0.00174    test-rmse:0.12031+0.00212
[1400]  train-rmse:0.10219+0.00175    test-rmse:0.11994+0.00218
[1500]  train-rmse:0.10166+0.00172    test-rmse:0.11967+0.00220
[1600]  train-rmse:0.10120+0.00169    test-rmse:0.11941+0.00225
[1700]  train-rmse:0.10077+0.00164    test-rmse:0.11914+0.00233
[1800]  train-rmse:0.10035+0.00162    test-rmse:0.11892+0.00237
[1900]  train-rmse:0.10001+0.00158    test-rmse:0.11873+0.00243
[2000]  train-rmse:0.09966+0.00157    test-rmse:0.11854+0.00246
[2100]  train-rmse:0.09936+0.00157    test-rmse:0.11838+0.00248
[2200]  train-rmse:0.09905+0.00156    test-rmse:0.11819+0.00250
[2300]  train-rmse:0.09877+0.00155    test-rmse:0.11804+0.00254
[2400]  train-rmse:0.09849+0.00154    test-rmse:0.11787+0.00258
[2500]  train-rmse:0.09821+0.00152    test-rmse:0.11770+0.00259
[2600]  train-rmse:0.09797+0.00153    test-rmse:0.11760+0.00260
[2700]  train-rmse:0.09775+0.00153    test-rmse:0.11748+0.00261
[2800]  train-rmse:0.09752+0.00151    test-rmse:0.11736+0.00265
[2900]  train-rmse:0.09728+0.00151    test-rmse:0.11723+0.00268
[3000]  train-rmse:0.09706+0.00151    test-rmse:0.11709+0.00271
[3100]  train-rmse:0.09685+0.00149    test-rmse:0.11698+0.00272
[3200]  train-rmse:0.09667+0.00150    test-rmse:0.11690+0.00274
[3300]  train-rmse:0.09645+0.00150    test-rmse:0.11679+0.00275
[3400]  train-rmse:0.09629+0.00148    test-rmse:0.11671+0.00276
[3500]  train-rmse:0.09610+0.00148    test-rmse:0.11664+0.00278
[3600]  train-rmse:0.09593+0.00149    test-rmse:0.11657+0.00276
[3700]  train-rmse:0.09577+0.00147    test-rmse:0.11648+0.00279
[3800]  train-rmse:0.09560+0.00146    test-rmse:0.11639+0.00281
[3900]  train-rmse:0.09545+0.00147    test-rmse:0.11632+0.00282
[4000]  train-rmse:0.09530+0.00146    test-rmse:0.11624+0.00284
[4100]  train-rmse:0.09516+0.00146    test-rmse:0.11617+0.00282
[4200]  train-rmse:0.09503+0.00144    test-rmse:0.11610+0.00285
[4300]  train-rmse:0.09489+0.00145    test-rmse:0.11603+0.00284
[4400]  train-rmse:0.09477+0.00144    test-rmse:0.11596+0.00288
[4500]  train-rmse:0.09466+0.00144    test-rmse:0.11590+0.00287
[4600]  train-rmse:0.09452+0.00143    test-rmse:0.11584+0.00288
[4700]  train-rmse:0.09441+0.00142    test-rmse:0.11578+0.00288
[4800]  train-rmse:0.09430+0.00140    test-rmse:0.11573+0.00291
[4900]  train-rmse:0.09419+0.00139    test-rmse:0.11569+0.00292
[5000]  train-rmse:0.09408+0.00140    test-rmse:0.11564+0.00293
[5100]  train-rmse:0.09397+0.00140    test-rmse:0.11560+0.00293
[5200]  train-rmse:0.09387+0.00139    test-rmse:0.11556+0.00294
[5300]  train-rmse:0.09376+0.00139    test-rmse:0.11551+0.00295
[5400]  train-rmse:0.09366+0.00139    test-rmse:0.11546+0.00296
[5500]  train-rmse:0.09356+0.00140    test-rmse:0.11542+0.00296
[5600]  train-rmse:0.09346+0.00140    test-rmse:0.11538+0.00296
[5700]  train-rmse:0.09337+0.00141    test-rmse:0.11532+0.00297
[5800]  train-rmse:0.09328+0.00141    test-rmse:0.11530+0.00297
```

```
[5900]  train-rmse:0.09317+0.00140      test-rmse:0.11524+0.00299
[6000]  train-rmse:0.09309+0.00141      test-rmse:0.11521+0.00299
[6100]  train-rmse:0.09299+0.00140      test-rmse:0.11514+0.00300
[6200]  train-rmse:0.09290+0.00141      test-rmse:0.11513+0.00300
[6300]  train-rmse:0.09281+0.00142      test-rmse:0.11510+0.00300
[6400]  train-rmse:0.09271+0.00140      test-rmse:0.11505+0.00304
[6500]  train-rmse:0.09263+0.00141      test-rmse:0.11500+0.00305
[6600]  train-rmse:0.09256+0.00141      test-rmse:0.11496+0.00305
[6700]  train-rmse:0.09248+0.00140      test-rmse:0.11492+0.00306
[6751]  train-rmse:0.09244+0.00141      test-rmse:0.11492+0.00306
[0]     train-rmse:0.43740+0.00235      test-rmse:0.43751+0.00472
[100]   train-rmse:0.14626+0.00112      test-rmse:0.15512+0.00156
[200]   train-rmse:0.12483+0.00098      test-rmse:0.13708+0.00176
[300]   train-rmse:0.11613+0.00133      test-rmse:0.13024+0.00191
[400]   train-rmse:0.11141+0.00134      test-rmse:0.12663+0.00221
[500]   train-rmse:0.10841+0.00141      test-rmse:0.12445+0.00229
[595]   train-rmse:0.10766+0.00124      test-rmse:0.12389+0.00243
[0]     train-rmse:0.43685+0.00219      test-rmse:0.43702+0.00485
[100]   train-rmse:0.14571+0.00117      test-rmse:0.15488+0.00162
[200]   train-rmse:0.12427+0.00096      test-rmse:0.13703+0.00194
[300]   train-rmse:0.11555+0.00144      test-rmse:0.13039+0.00201
[400]   train-rmse:0.11079+0.00154      test-rmse:0.12684+0.00213
[500]   train-rmse:0.10772+0.00160      test-rmse:0.12467+0.00220
[581]   train-rmse:0.10743+0.00137      test-rmse:0.12449+0.00240
[0]     train-rmse:0.43758+0.00227      test-rmse:0.43771+0.00473
[100]   train-rmse:0.14824+0.00094      test-rmse:0.15651+0.00217
[200]   train-rmse:0.12622+0.00099      test-rmse:0.13785+0.00190
[300]   train-rmse:0.11766+0.00122      test-rmse:0.13118+0.00212
[400]   train-rmse:0.11271+0.00139      test-rmse:0.12746+0.00218
[500]   train-rmse:0.10951+0.00148      test-rmse:0.12521+0.00222
[600]   train-rmse:0.10721+0.00153      test-rmse:0.12363+0.00230
[700]   train-rmse:0.10556+0.00158      test-rmse:0.12257+0.00236
[800]   train-rmse:0.10441+0.00161      test-rmse:0.12184+0.00239
[900]   train-rmse:0.10356+0.00168      test-rmse:0.12133+0.00243
[1000]  train-rmse:0.10280+0.00168      test-rmse:0.12083+0.00249
[1100]  train-rmse:0.10225+0.00170      test-rmse:0.12050+0.00253
[1200]  train-rmse:0.10181+0.00170      test-rmse:0.12023+0.00260
[1300]  train-rmse:0.10139+0.00170      test-rmse:0.11995+0.00261
[1400]  train-rmse:0.10104+0.00168      test-rmse:0.11975+0.00263
[1500]  train-rmse:0.10072+0.00171      test-rmse:0.11956+0.00265
[1600]  train-rmse:0.10044+0.00169      test-rmse:0.11938+0.00268
[1700]  train-rmse:0.10016+0.00169      test-rmse:0.11924+0.00267
[1800]  train-rmse:0.09994+0.00169      test-rmse:0.11911+0.00270
[1900]  train-rmse:0.09975+0.00166      test-rmse:0.11900+0.00273
[2000]  train-rmse:0.09958+0.00164      test-rmse:0.11890+0.00273
[2100]  train-rmse:0.09940+0.00167      test-rmse:0.11880+0.00275
[2200]  train-rmse:0.09926+0.00164      test-rmse:0.11872+0.00276
[2300]  train-rmse:0.09911+0.00164      test-rmse:0.11862+0.00278
```

```
[2400]	train-rmse:0.09896+0.00164	test-rmse:0.11853+0.00278
[2500]	train-rmse:0.09881+0.00162	test-rmse:0.11844+0.00279
[2600]	train-rmse:0.09868+0.00163	test-rmse:0.11837+0.00279
[2700]	train-rmse:0.09855+0.00163	test-rmse:0.11828+0.00280
[2800]	train-rmse:0.09845+0.00161	test-rmse:0.11822+0.00282
[2900]	train-rmse:0.09834+0.00161	test-rmse:0.11816+0.00282
[3000]	train-rmse:0.09823+0.00160	test-rmse:0.11810+0.00284
[3100]	train-rmse:0.09813+0.00160	test-rmse:0.11803+0.00285
[3200]	train-rmse:0.09803+0.00160	test-rmse:0.11798+0.00286
[3300]	train-rmse:0.09792+0.00160	test-rmse:0.11791+0.00286
[3400]	train-rmse:0.09781+0.00159	test-rmse:0.11785+0.00285
[3500]	train-rmse:0.09771+0.00160	test-rmse:0.11779+0.00287
[3600]	train-rmse:0.09762+0.00159	test-rmse:0.11775+0.00287
[3700]	train-rmse:0.09754+0.00157	test-rmse:0.11770+0.00289
[3800]	train-rmse:0.09746+0.00157	test-rmse:0.11765+0.00290
[3900]	train-rmse:0.09740+0.00158	test-rmse:0.11762+0.00291
[4000]	train-rmse:0.09732+0.00158	test-rmse:0.11756+0.00291
[4100]	train-rmse:0.09721+0.00157	test-rmse:0.11749+0.00292
[4200]	train-rmse:0.09713+0.00160	test-rmse:0.11746+0.00292
[4300]	train-rmse:0.09707+0.00159	test-rmse:0.11742+0.00293
[4400]	train-rmse:0.09700+0.00159	test-rmse:0.11738+0.00294
[4500]	train-rmse:0.09695+0.00160	test-rmse:0.11735+0.00294
[4600]	train-rmse:0.09691+0.00159	test-rmse:0.11733+0.00294
[4700]	train-rmse:0.09684+0.00157	test-rmse:0.11730+0.00295
[4800]	train-rmse:0.09676+0.00156	test-rmse:0.11726+0.00297
[4900]	train-rmse:0.09669+0.00157	test-rmse:0.11722+0.00297
[5000]	train-rmse:0.09663+0.00157	test-rmse:0.11720+0.00299
[5100]	train-rmse:0.09657+0.00157	test-rmse:0.11717+0.00299
[5200]	train-rmse:0.09653+0.00157	test-rmse:0.11714+0.00300
[5300]	train-rmse:0.09648+0.00158	test-rmse:0.11712+0.00300
[5400]	train-rmse:0.09641+0.00157	test-rmse:0.11708+0.00300
[5500]	train-rmse:0.09637+0.00157	test-rmse:0.11706+0.00300
[5600]	train-rmse:0.09631+0.00157	test-rmse:0.11703+0.00300
[5700]	train-rmse:0.09626+0.00155	test-rmse:0.11699+0.00302
[5800]	train-rmse:0.09621+0.00154	test-rmse:0.11697+0.00303
[5900]	train-rmse:0.09616+0.00155	test-rmse:0.11694+0.00303
[6000]	train-rmse:0.09611+0.00155	test-rmse:0.11692+0.00304
[6100]	train-rmse:0.09607+0.00156	test-rmse:0.11690+0.00304
[6200]	train-rmse:0.09601+0.00156	test-rmse:0.11686+0.00304
[6300]	train-rmse:0.09595+0.00156	test-rmse:0.11684+0.00304
[6400]	train-rmse:0.09590+0.00156	test-rmse:0.11681+0.00304
[6500]	train-rmse:0.09586+0.00156	test-rmse:0.11680+0.00305
[6600]	train-rmse:0.09582+0.00155	test-rmse:0.11677+0.00306
[6700]	train-rmse:0.09578+0.00154	test-rmse:0.11675+0.00307
[6800]	train-rmse:0.09573+0.00155	test-rmse:0.11674+0.00307
[6900]	train-rmse:0.09570+0.00155	test-rmse:0.11671+0.00307
[7000]	train-rmse:0.09566+0.00155	test-rmse:0.11670+0.00306
[7100]	train-rmse:0.09563+0.00154	test-rmse:0.11669+0.00306
```

```
[7200]  train-rmse:0.09559+0.00153      test-rmse:0.11667+0.00307
[7300]  train-rmse:0.09556+0.00153      test-rmse:0.11665+0.00308
[7400]  train-rmse:0.09552+0.00152      test-rmse:0.11661+0.00308
[7500]  train-rmse:0.09547+0.00151      test-rmse:0.11660+0.00309
[7600]  train-rmse:0.09545+0.00151      test-rmse:0.11658+0.00309
[7700]  train-rmse:0.09541+0.00151      test-rmse:0.11656+0.00310
[7800]  train-rmse:0.09537+0.00151      test-rmse:0.11654+0.00310
[7900]  train-rmse:0.09535+0.00150      test-rmse:0.11653+0.00310
[8000]  train-rmse:0.09532+0.00150      test-rmse:0.11651+0.00310
[8100]  train-rmse:0.09528+0.00150      test-rmse:0.11649+0.00310
[8200]  train-rmse:0.09525+0.00150      test-rmse:0.11647+0.00311
[8300]  train-rmse:0.09522+0.00149      test-rmse:0.11646+0.00312
[8400]  train-rmse:0.09519+0.00150      test-rmse:0.11644+0.00312
[8500]  train-rmse:0.09517+0.00149      test-rmse:0.11643+0.00312
[8600]  train-rmse:0.09514+0.00149      test-rmse:0.11642+0.00312
[8700]  train-rmse:0.09510+0.00149      test-rmse:0.11639+0.00313
[8800]  train-rmse:0.09507+0.00149      test-rmse:0.11638+0.00313
[8900]  train-rmse:0.09503+0.00148      test-rmse:0.11637+0.00313
[9000]  train-rmse:0.09500+0.00148      test-rmse:0.11635+0.00314
[9100]  train-rmse:0.09498+0.00148      test-rmse:0.11634+0.00314
[9200]  train-rmse:0.09496+0.00148      test-rmse:0.11633+0.00314
[9300]  train-rmse:0.09493+0.00148      test-rmse:0.11632+0.00314
[9400]  train-rmse:0.09490+0.00147      test-rmse:0.11630+0.00314
[9500]  train-rmse:0.09486+0.00147      test-rmse:0.11628+0.00314
[9600]  train-rmse:0.09484+0.00147      test-rmse:0.11627+0.00315
[9700]  train-rmse:0.09481+0.00147      test-rmse:0.11625+0.00315
[9800]  train-rmse:0.09478+0.00146      test-rmse:0.11623+0.00316
[9900]  train-rmse:0.09476+0.00146      test-rmse:0.11622+0.00316
[9999]  train-rmse:0.09473+0.00145      test-rmse:0.11621+0.00316
[0]     train-rmse:0.43751+0.00223      test-rmse:0.43769+0.00477
[100]   train-rmse:0.13752+0.00116      test-rmse:0.15028+0.00192
[200]   train-rmse:0.11485+0.00122      test-rmse:0.13210+0.00147
[300]   train-rmse:0.10650+0.00156      test-rmse:0.12609+0.00176
[400]   train-rmse:0.10206+0.00168      test-rmse:0.12315+0.00179
[500]   train-rmse:0.09932+0.00172      test-rmse:0.12147+0.00192
[600]   train-rmse:0.09748+0.00172      test-rmse:0.12039+0.00202
[700]   train-rmse:0.09615+0.00184      test-rmse:0.11967+0.00203
[800]   train-rmse:0.09518+0.00180      test-rmse:0.11912+0.00212
[900]   train-rmse:0.09449+0.00183      test-rmse:0.11876+0.00214
[1000]  train-rmse:0.09379+0.00177      test-rmse:0.11836+0.00219
[1100]  train-rmse:0.09327+0.00178      test-rmse:0.11807+0.00219
[1200]  train-rmse:0.09282+0.00181      test-rmse:0.11783+0.00223
[1300]  train-rmse:0.09237+0.00180      test-rmse:0.11762+0.00224
[1400]  train-rmse:0.09199+0.00180      test-rmse:0.11742+0.00227
[1500]  train-rmse:0.09168+0.00182      test-rmse:0.11728+0.00225
[1600]  train-rmse:0.09135+0.00185      test-rmse:0.11712+0.00227
[1700]  train-rmse:0.09105+0.00181      test-rmse:0.11696+0.00234
[1800]  train-rmse:0.09076+0.00180      test-rmse:0.11681+0.00236
```

```
[1900]  train-rmse:0.09054+0.00179      test-rmse:0.11673+0.00237
[2000]  train-rmse:0.09030+0.00180      test-rmse:0.11661+0.00237
[2100]  train-rmse:0.09008+0.00178      test-rmse:0.11650+0.00240
[2200]  train-rmse:0.08989+0.00176      test-rmse:0.11641+0.00241
[2300]  train-rmse:0.08973+0.00177      test-rmse:0.11634+0.00243
[2400]  train-rmse:0.08955+0.00176      test-rmse:0.11627+0.00245
[2500]  train-rmse:0.08940+0.00174      test-rmse:0.11618+0.00246
[2600]  train-rmse:0.08925+0.00173      test-rmse:0.11611+0.00246
[2700]  train-rmse:0.08910+0.00174      test-rmse:0.11603+0.00249
[2800]  train-rmse:0.08893+0.00173      test-rmse:0.11597+0.00248
[2900]  train-rmse:0.08880+0.00175      test-rmse:0.11589+0.00251
[3000]  train-rmse:0.08867+0.00173      test-rmse:0.11582+0.00251
[3100]  train-rmse:0.08853+0.00176      test-rmse:0.11575+0.00251
[3200]  train-rmse:0.08840+0.00175      test-rmse:0.11570+0.00250
[3300]  train-rmse:0.08827+0.00178      test-rmse:0.11564+0.00251
[3400]  train-rmse:0.08817+0.00178      test-rmse:0.11560+0.00251
[3500]  train-rmse:0.08805+0.00178      test-rmse:0.11555+0.00251
[3600]  train-rmse:0.08792+0.00177      test-rmse:0.11550+0.00249
[3700]  train-rmse:0.08779+0.00175      test-rmse:0.11545+0.00251
[3800]  train-rmse:0.08767+0.00173      test-rmse:0.11539+0.00252
[3900]  train-rmse:0.08756+0.00171      test-rmse:0.11535+0.00253
[4000]  train-rmse:0.08745+0.00172      test-rmse:0.11531+0.00252
[4100]  train-rmse:0.08732+0.00173      test-rmse:0.11525+0.00252
[4200]  train-rmse:0.08724+0.00173      test-rmse:0.11521+0.00254
[4300]  train-rmse:0.08716+0.00172      test-rmse:0.11516+0.00253
[4400]  train-rmse:0.08707+0.00171      test-rmse:0.11513+0.00255
[4500]  train-rmse:0.08698+0.00170      test-rmse:0.11509+0.00254
[4600]  train-rmse:0.08690+0.00169      test-rmse:0.11507+0.00254
[4700]  train-rmse:0.08681+0.00167      test-rmse:0.11502+0.00256
[4800]  train-rmse:0.08674+0.00166      test-rmse:0.11498+0.00256
[4900]  train-rmse:0.08668+0.00165      test-rmse:0.11495+0.00257
[5000]  train-rmse:0.08659+0.00165      test-rmse:0.11492+0.00258
[5100]  train-rmse:0.08652+0.00165      test-rmse:0.11489+0.00256
[5200]  train-rmse:0.08646+0.00166      test-rmse:0.11487+0.00257
[5300]  train-rmse:0.08638+0.00165      test-rmse:0.11483+0.00258
[5400]  train-rmse:0.08631+0.00165      test-rmse:0.11480+0.00258
[5500]  train-rmse:0.08625+0.00164      test-rmse:0.11476+0.00258
[5600]  train-rmse:0.08618+0.00163      test-rmse:0.11473+0.00258
[5700]  train-rmse:0.08609+0.00162      test-rmse:0.11468+0.00260
[5800]  train-rmse:0.08604+0.00161      test-rmse:0.11466+0.00260
[5900]  train-rmse:0.08596+0.00160      test-rmse:0.11462+0.00261
[6000]  train-rmse:0.08588+0.00160      test-rmse:0.11460+0.00260
[6100]  train-rmse:0.08580+0.00160      test-rmse:0.11456+0.00259
[6200]  train-rmse:0.08573+0.00160      test-rmse:0.11452+0.00259
[6300]  train-rmse:0.08566+0.00162      test-rmse:0.11450+0.00260
[6400]  train-rmse:0.08559+0.00162      test-rmse:0.11447+0.00260
[6500]  train-rmse:0.08554+0.00161      test-rmse:0.11444+0.00260
[6600]  train-rmse:0.08548+0.00161      test-rmse:0.11443+0.00260
```

```
[6700]  train-rmse:0.08541+0.00161      test-rmse:0.11440+0.00259
[6800]  train-rmse:0.08535+0.00161      test-rmse:0.11437+0.00260
[6900]  train-rmse:0.08529+0.00163      test-rmse:0.11435+0.00259
[7000]  train-rmse:0.08524+0.00162      test-rmse:0.11433+0.00260
[7100]  train-rmse:0.08518+0.00162      test-rmse:0.11430+0.00260
[7200]  train-rmse:0.08513+0.00161      test-rmse:0.11427+0.00260
[7300]  train-rmse:0.08509+0.00160      test-rmse:0.11425+0.00261
[7400]  train-rmse:0.08504+0.00159      test-rmse:0.11423+0.00261
[7500]  train-rmse:0.08498+0.00158      test-rmse:0.11421+0.00261
[7600]  train-rmse:0.08493+0.00158      test-rmse:0.11419+0.00260
[7700]  train-rmse:0.08488+0.00158      test-rmse:0.11416+0.00261
[7800]  train-rmse:0.08483+0.00157      test-rmse:0.11414+0.00261
[7900]  train-rmse:0.08477+0.00157      test-rmse:0.11411+0.00262
[8000]  train-rmse:0.08474+0.00156      test-rmse:0.11410+0.00262
[8100]  train-rmse:0.08469+0.00154      test-rmse:0.11408+0.00262
[8200]  train-rmse:0.08465+0.00153      test-rmse:0.11406+0.00263
[8300]  train-rmse:0.08461+0.00153      test-rmse:0.11404+0.00264
[8400]  train-rmse:0.08457+0.00153      test-rmse:0.11403+0.00264
[8500]  train-rmse:0.08453+0.00153      test-rmse:0.11401+0.00265
[8600]  train-rmse:0.08449+0.00153      test-rmse:0.11400+0.00265
[8700]  train-rmse:0.08445+0.00153      test-rmse:0.11397+0.00265
[8800]  train-rmse:0.08441+0.00152      test-rmse:0.11395+0.00266
[8900]  train-rmse:0.08436+0.00152      test-rmse:0.11393+0.00266
[8944]  train-rmse:0.08435+0.00153      test-rmse:0.11393+0.00266
```

[271]:

```
inv_xgb_no_tune_params1 = {
    'subsample':0.7,
    'max_depth': 4,
    'colsample_bytree':0.6,
    'lambda': 100,
    'alpha':8,
    'eta': 0.05
}
inv_xgb_no_tune_params2 = {
    'subsample':1,
    'max_depth':4,
    'colsample_bytree':0.5,
    'lambda': 100,
    'alpha':8,
    'eta':0.05
}

inv_xgb_no_tune_params3 = {
    'subsample':1,
    'max_depth':4,
    'colsample_bytree':1,
    'lambda': 100,
```

```
    'alpha':8,
    'eta':0.05
}

inv_xgb_no_tune_params4 = {
    'subsample':0.9,
    'max_depth':4,
    'colsample_bytree':0.8,
    'lambda': 100,
    'alpha':8,
    'eta':0.05
}

inv_xgb_no_tune_params5 = {
    'subsample':0.7,
    'max_depth':6,
    'colsample_bytree':1,
    'lambda': 100,
    'alpha':8,
    'eta':0.05
}


inv_notue_xgb1 = no_tune_train(params=inv_xgb_no_tune_params1,␣
 ↪x_train=final_inv_train,y_train=inv_train.log_price)
inv_notue_xgb2 = no_tune_train(params=inv_xgb_no_tune_params2,␣
 ↪x_train=final_inv_train,y_train=inv_train.log_price)
inv_notue_xgb3 = no_tune_train(params=inv_xgb_no_tune_params3,␣
 ↪x_train=final_inv_train,y_train=inv_train.log_price)
inv_notue_xgb4 = no_tune_train(params=inv_xgb_no_tune_params4,␣
 ↪x_train=final_inv_train,y_train=inv_train.log_price)
inv_notue_xgb5 = no_tune_train(params=inv_xgb_no_tune_params5,␣
 ↪x_train=final_inv_train,y_train=inv_train.log_price)
```

```
[0]     train-rmse:0.37664+0.00145      test-rmse:0.37672+0.00315
[100]   train-rmse:0.19010+0.00260      test-rmse:0.19851+0.00594
[200]   train-rmse:0.17815+0.00263      test-rmse:0.19003+0.00638
[300]   train-rmse:0.17254+0.00256      test-rmse:0.18758+0.00650
[400]   train-rmse:0.16838+0.00252      test-rmse:0.18633+0.00642
[500]   train-rmse:0.16497+0.00249      test-rmse:0.18569+0.00646
[600]   train-rmse:0.16198+0.00244      test-rmse:0.18511+0.00644
[700]   train-rmse:0.15934+0.00238      test-rmse:0.18488+0.00639
[800]   train-rmse:0.15695+0.00237      test-rmse:0.18468+0.00637
[900]   train-rmse:0.15475+0.00242      test-rmse:0.18445+0.00633
[1000]  train-rmse:0.15270+0.00241      test-rmse:0.18428+0.00632
[1051]  train-rmse:0.15171+0.00240      test-rmse:0.18428+0.00632
[0]     train-rmse:0.37625+0.00147      test-rmse:0.37639+0.00313
```

```
[100]   train-rmse:0.18728+0.00264   test-rmse:0.19715+0.00600
[200]   train-rmse:0.17591+0.00280   test-rmse:0.18924+0.00620
[300]   train-rmse:0.17054+0.00273   test-rmse:0.18692+0.00619
[400]   train-rmse:0.16648+0.00275   test-rmse:0.18581+0.00614
[500]   train-rmse:0.16299+0.00268   test-rmse:0.18523+0.00619
[600]   train-rmse:0.16008+0.00270   test-rmse:0.18486+0.00614
[700]   train-rmse:0.15745+0.00270   test-rmse:0.18464+0.00618
[800]   train-rmse:0.15563+0.00307   test-rmse:0.18448+0.00615
[900]   train-rmse:0.15493+0.00404   test-rmse:0.18446+0.00612
[946]   train-rmse:0.15486+0.00415   test-rmse:0.18446+0.00612
[0]     train-rmse:0.37614+0.00152   test-rmse:0.37628+0.00306
[100]   train-rmse:0.18636+0.00275   test-rmse:0.19686+0.00598
[200]   train-rmse:0.17476+0.00262   test-rmse:0.18916+0.00633
[300]   train-rmse:0.16912+0.00253   test-rmse:0.18695+0.00649
[400]   train-rmse:0.16486+0.00250   test-rmse:0.18596+0.00649
[500]   train-rmse:0.16124+0.00259   test-rmse:0.18541+0.00641
[600]   train-rmse:0.15812+0.00265   test-rmse:0.18515+0.00645
[700]   train-rmse:0.15529+0.00265   test-rmse:0.18496+0.00645
[800]   train-rmse:0.15311+0.00240   test-rmse:0.18487+0.00647
[866]   train-rmse:0.15296+0.00237   test-rmse:0.18485+0.00648
[0]     train-rmse:0.37632+0.00151   test-rmse:0.37646+0.00312
[100]   train-rmse:0.18730+0.00275   test-rmse:0.19750+0.00596
[200]   train-rmse:0.17543+0.00293   test-rmse:0.18933+0.00628
[300]   train-rmse:0.16971+0.00281   test-rmse:0.18697+0.00639
[400]   train-rmse:0.16531+0.00272   test-rmse:0.18580+0.00633
[500]   train-rmse:0.16168+0.00269   test-rmse:0.18525+0.00633
[600]   train-rmse:0.15841+0.00274   test-rmse:0.18489+0.00634
[700]   train-rmse:0.15556+0.00263   test-rmse:0.18466+0.00633
[800]   train-rmse:0.15302+0.00263   test-rmse:0.18456+0.00631
[900]   train-rmse:0.15066+0.00255   test-rmse:0.18443+0.00628
[1000]  train-rmse:0.14847+0.00250   test-rmse:0.18435+0.00630
[1100]  train-rmse:0.14640+0.00247   test-rmse:0.18432+0.00631
[1200]  train-rmse:0.14449+0.00246   test-rmse:0.18422+0.00632
[1261]  train-rmse:0.14339+0.00244   test-rmse:0.18423+0.00633
[0]     train-rmse:0.37640+0.00143   test-rmse:0.37663+0.00311
[100]   train-rmse:0.17997+0.00286   test-rmse:0.19515+0.00638
[200]   train-rmse:0.16605+0.00277   test-rmse:0.18813+0.00661
[300]   train-rmse:0.15812+0.00263   test-rmse:0.18626+0.00649
[400]   train-rmse:0.15205+0.00244   test-rmse:0.18541+0.00645
[500]   train-rmse:0.14694+0.00247   test-rmse:0.18506+0.00642
[600]   train-rmse:0.14255+0.00233   test-rmse:0.18490+0.00651
[700]   train-rmse:0.13864+0.00221   test-rmse:0.18473+0.00647
[800]   train-rmse:0.13521+0.00214   test-rmse:0.18468+0.00652
[900]   train-rmse:0.13213+0.00212   test-rmse:0.18457+0.00655
[1000]  train-rmse:0.12938+0.00203   test-rmse:0.18453+0.00660
[1034]  train-rmse:0.12850+0.00200   test-rmse:0.18458+0.00660
```

9.2 Price SQ

[303]:
```
ocu_xgb_no_tune_params1 = {
    'subsample':0.7,
    'max_depth': 9,
    'colsample_bytree':0.7,
    'lambda': 200,
    'eta': 0.1,
    'nthread':8
}

ocu_xgb_no_tune_params2 = {
    'subsample':1,
    'max_depth':9,
    'colsample_bytree':0.7,
    'lambda': 200,
    'eta':0.1,
    'nthread':8
}

ocu_xgb_no_tune_params3 = {
    'subsample':1,
    'max_depth':9,
    'colsample_bytree':1,
    'lambda': 200,
    'eta':0.1,
    'nthread':8
}

ocu_xgb_no_tune_params4 = {
    'subsample':0.9,
    'max_depth':7,
    'colsample_bytree':0.7,
    'lambda': 200,
    'eta':0.1,
    'nthread':8
}

ocu_xgb_no_tune_params5 = {
    'subsample':0.8,
    'max_depth':5,
    'colsample_bytree':0.7,
    'lambda': 200,
    'eta':0.1,
    'nthread':8
}
```

```python
ocu_xgb_no_tune_params5 = {
    'subsample':0.8,
    'max_depth':5,
    'colsample_bytree':0.7,
    'lambda': 200,
    'eta':0.1,
    'nthread':8
}


ocu_pricesq_xgb1 = no_tune_train(params=ocu_xgb_no_tune_params1,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.price_sq)
ocu_pricesq_xgb2 = no_tune_train(params=ocu_xgb_no_tune_params2,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.price_sq)
ocu_pricesq_xgb3 = no_tune_train(params=ocu_xgb_no_tune_params3,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.price_sq)
#ocu_pricesq_xgb4 = no_tune_train(params=ocu_xgb_no_tune_params4,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.price_sq)
#ocu_pricesq_xgb5 = no_tune_train(params=ocu_xgb_no_tune_params5,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.price_sq)
```

```
[0]     train-rmse:38484.04351+82.81214 test-rmse:38511.40089+121.35604
[100]   train-rmse:13424.62413+254.73118        test-rmse:16532.09987+654.36717
[200]   train-rmse:10627.66035+241.24923        test-rmse:15392.24064+817.18197
[300]   train-rmse:9016.62751+248.22539 test-rmse:14983.03044+855.50944
[400]   train-rmse:7917.29103+222.79735 test-rmse:14828.63538+832.76186
[500]   train-rmse:7065.66106+185.11194 test-rmse:14743.64086+827.03857
[600]   train-rmse:6350.22851+155.12067 test-rmse:14715.67248+815.60607
[700]   train-rmse:5761.05577+140.79376 test-rmse:14696.74167+809.29647
[800]   train-rmse:5252.27029+125.26369 test-rmse:14686.02123+797.07021
[900]   train-rmse:4816.80461+112.31727 test-rmse:14675.51928+789.44450
[1000]  train-rmse:4437.63322+100.90364 test-rmse:14672.16563+781.59906
[1029]  train-rmse:4338.73054+104.52579 test-rmse:14672.98742+781.00732
[0]     train-rmse:38325.34227+82.76809 test-rmse:38361.97021+125.98146
[100]   train-rmse:12264.10733+282.85695        test-rmse:15941.27286+640.49545
[200]   train-rmse:9571.51963+262.57019 test-rmse:14960.51730+765.37213
[300]   train-rmse:7977.62319+78.87495  test-rmse:14679.75808+794.12570
[400]   train-rmse:6926.04498+103.97775 test-rmse:14598.29623+798.35386
[500]   train-rmse:6111.25220+106.54485 test-rmse:14566.52595+799.86332
[600]   train-rmse:5467.41191+118.65159 test-rmse:14546.56774+787.89674
[663]   train-rmse:5131.79120+141.73843 test-rmse:14549.51370+789.06929
[0]     train-rmse:38285.06338+75.46018 test-rmse:38323.41696+129.52488
[100]   train-rmse:12269.69857+199.75133        test-rmse:16083.18546+670.70088
[200]   train-rmse:9638.59525+247.09952 test-rmse:15068.71348+793.10939
[300]   train-rmse:7991.22889+57.25974  test-rmse:14767.46078+800.76465
[400]   train-rmse:6832.73762+30.87685  test-rmse:14696.33936+766.84479
[500]   train-rmse:5967.88760+118.60616 test-rmse:14690.46550+742.77208
```

```
[530]	train-rmse:5750.46650+136.85066 test-rmse:14693.80237+744.51364
```

[309]:

```
inv_xgb_no_tune_params1 = {
    'subsample':0.7,
    'max_depth': 6,
    'colsample_bytree':0.6,
    'lambda': 8,
    'eta': 0.02,
    'nthread':8
}

inv_xgb_no_tune_params2 = {
    'subsample':0.9,
    'max_depth':6,
    'colsample_bytree':0.5,
    'lambda': 8,
    'eta':0.02,
    'nthread':8
}

inv_xgb_no_tune_params3 = {
    'subsample':1,
    'max_depth':6,
    'colsample_bytree':1,
    'lambda': 8,
    'eta':0.02,
    'nthread':8
}

inv_xgb_no_tune_params4 = {
    'subsample':0.9,
    'max_depth':6,
    'colsample_bytree':0.8,
    'lambda': 8,
    'eta':0.02,
    'nthread':8
}

inv_xgb_no_tune_params5 = {
    'subsample':0.7,
    'max_depth':6,
    'colsample_bytree':1,
    'lambda': 8,
    'eta':0.02,
    'nthread':8
}
```

```
inv_pricesq_xgb1 = no_tune_train(params=inv_xgb_no_tune_params1,␣
↪x_train=final_inv_train,y_train=inv_train.price_sq)
inv_pricesq_xgb2 = no_tune_train(params=inv_xgb_no_tune_params2,␣
↪x_train=final_inv_train,y_train=inv_train.price_sq)
inv_pricesq_xgb3 = no_tune_train(params=inv_xgb_no_tune_params3,␣
↪x_train=final_inv_train,y_train=inv_train.price_sq)
inv_pricesq_xgb4 = no_tune_train(params=inv_xgb_no_tune_params4,␣
↪x_train=final_inv_train,y_train=inv_train.price_sq)
inv_pricesq_xgb5 = no_tune_train(params=inv_xgb_no_tune_params5,␣
↪x_train=final_inv_train,y_train=inv_train.price_sq)
```

```
[0]     train-rmse:40112.21781+175.26007        test-rmse:40151.52418+345.31237
[100]   train-rmse:25713.19561+231.26029        test-rmse:28243.93444+570.67381
[200]   train-rmse:22325.83311+231.46293        test-rmse:26547.64555+654.96826
[300]   train-rmse:20572.27552+226.64285        test-rmse:26035.94167+685.93400
[400]   train-rmse:19274.49162+218.20516        test-rmse:25815.40726+690.83347
[500]   train-rmse:18143.59007+220.51647        test-rmse:25690.77861+706.13558
[600]   train-rmse:17098.61699+194.43773        test-rmse:25602.38414+700.71122
[700]   train-rmse:16191.43047+183.59547        test-rmse:25568.85820+708.72245
[800]   train-rmse:15317.22658+194.48421        test-rmse:25544.64502+701.07093
[900]   train-rmse:14523.65541+184.46306        test-rmse:25524.58302+697.47492
[1000]  train-rmse:13764.89153+192.50943        test-rmse:25513.95836+701.97912
[1096]  train-rmse:13066.95545+183.99071        test-rmse:25512.94185+708.13624
[0]     train-rmse:40104.87528+181.34007        test-rmse:40144.53793+344.42132
[100]   train-rmse:25573.80319+211.94962        test-rmse:28225.29001+546.58794
[200]   train-rmse:22125.25758+225.48297        test-rmse:26526.03713+616.55481
[300]   train-rmse:20356.30941+192.95062        test-rmse:25993.02056+670.48326
[400]   train-rmse:19055.85392+209.78975        test-rmse:25773.97448+676.99983
[500]   train-rmse:17975.95951+202.84791        test-rmse:25653.16361+673.04108
[600]   train-rmse:17017.54720+193.38738        test-rmse:25581.96372+676.04512
[700]   train-rmse:16109.07425+213.92915        test-rmse:25539.27983+671.51846
[800]   train-rmse:15304.90073+221.86642        test-rmse:25515.50266+675.53331
[900]   train-rmse:14528.64375+211.31627        test-rmse:25497.50124+665.35837
[1000]  train-rmse:13840.42151+210.31587        test-rmse:25483.04309+669.26762
[1100]  train-rmse:13187.12322+202.90460        test-rmse:25478.29638+662.99636
[1123]  train-rmse:13045.78137+193.52022        test-rmse:25479.14973+664.86461
[0]     train-rmse:40091.48104+176.62077        test-rmse:40142.81723+349.31856
[100]   train-rmse:25230.82412+237.48703        test-rmse:28210.36184+476.21528
[200]   train-rmse:21806.41111+268.46419        test-rmse:26626.35423+537.39232
[300]   train-rmse:20043.92496+256.15611        test-rmse:26142.33494+544.08696
[400]   train-rmse:18827.14995+239.72498        test-rmse:25938.31806+556.64564
[500]   train-rmse:17851.84147+248.96391        test-rmse:25838.92680+561.28374
[600]   train-rmse:17012.78311+245.99652        test-rmse:25791.86855+567.33443
[700]   train-rmse:16210.58251+252.21551        test-rmse:25765.34613+564.67262
[800]   train-rmse:15479.54069+267.64126        test-rmse:25736.91844+566.50680
[900]   train-rmse:14802.79544+253.15374        test-rmse:25725.68236+568.74879
[1000]  train-rmse:14213.70850+275.44110        test-rmse:25716.35282+574.22306
```

```
[1023]  train-rmse:14081.39975+284.61799        test-rmse:25716.26264+573.64021
[0]     train-rmse:40100.30738+179.80512        test-rmse:40142.23219+343.76530
[100]   train-rmse:25338.60307+227.13645        test-rmse:28167.47178+584.69803
[200]   train-rmse:21843.27759+223.35781        test-rmse:26537.58469+652.09021
[300]   train-rmse:20045.65537+211.10064        test-rmse:26045.88003+661.40152
[400]   train-rmse:18729.16594+213.14176        test-rmse:25848.98584+647.93477
[500]   train-rmse:17598.80925+193.07936        test-rmse:25729.37333+647.64883
[600]   train-rmse:16618.91879+193.68059        test-rmse:25670.73440+641.47182
[700]   train-rmse:15705.15004+176.72161        test-rmse:25639.43614+642.70056
[800]   train-rmse:14901.33047+180.09017        test-rmse:25618.40493+645.15552
[900]   train-rmse:14104.36939+178.64508        test-rmse:25598.74759+636.15278
[1000]  train-rmse:13385.27053+168.18778        test-rmse:25583.91390+630.86253
[1100]  train-rmse:12697.63255+185.21431        test-rmse:25580.00242+629.08821
[1128]  train-rmse:12525.32874+178.89176        test-rmse:25580.82581+631.77730
[0]     train-rmse:40103.01254+180.71323        test-rmse:40138.94820+345.24360
[100]   train-rmse:25462.45520+200.58399        test-rmse:28212.33727+540.62766
[200]   train-rmse:22017.62097+217.78137        test-rmse:26550.50226+639.96330
[300]   train-rmse:20262.26150+202.91240        test-rmse:26064.55233+668.81168
[400]   train-rmse:18879.71051+176.92455        test-rmse:25832.43824+701.68727
[500]   train-rmse:17725.28978+163.96314        test-rmse:25716.77768+693.56615
[600]   train-rmse:16666.77675+134.60030        test-rmse:25657.19211+697.88980
[700]   train-rmse:15684.15170+138.78549        test-rmse:25623.50753+696.17407
[800]   train-rmse:14819.52773+125.28874        test-rmse:25601.33754+690.66189
[900]   train-rmse:13977.16086+118.24544        test-rmse:25592.65241+694.78896
[938]   train-rmse:13670.83779+125.35484        test-rmse:25595.15545+696.90130
```

9.3 Price Doc

[274]:

```
ocu_price_xgb1 = no_tune_train(params=ocu_xgb_no_tune_params1,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.price_doc)
ocu_price_xgb2 = no_tune_train(params=ocu_xgb_no_tune_params2,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.price_doc)
ocu_price_xgb3 = no_tune_train(params=ocu_xgb_no_tune_params3,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.price_doc)
ocu_price_xgb4 = no_tune_train(params=ocu_xgb_no_tune_params4,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.price_doc)
ocu_price_xgb5 = no_tune_train(params=ocu_xgb_no_tune_params5,␣
 ↪x_train=final_ocu_train,y_train=ocu_train.price_doc)
```

```
[0]     train-rmse:3997561.35034+105907.26534   test-
rmse:3997081.92588+227108.23764
[100]   train-rmse:1394180.32318+86822.30984    test-
rmse:1673751.15972+266801.50298
[200]   train-rmse:1051343.93733+71406.55071    test-
rmse:1497363.61191+245541.14749
[300]   train-rmse:869121.88629+62188.59237     test-
rmse:1436081.40288+231210.73280
[400]   train-rmse:746920.60417+54914.22926     test-
```

```
rmse:1407370.71659+222621.85112
[500]   train-rmse:657828.42981+46388.24218     test-
rmse:1392457.41027+217308.31632
[600]   train-rmse:587418.87263+38347.58312     test-
rmse:1386128.96860+213790.19186
[700]   train-rmse:529879.66017+32439.42607     test-
rmse:1380192.71594+212187.48130
[800]   train-rmse:483781.76497+29393.89820     test-
rmse:1375431.63458+209616.37949
[900]   train-rmse:443424.80433+25430.11737     test-
rmse:1372541.69579+207543.18844
[1000]  train-rmse:407289.65558+22673.81195     test-
rmse:1370148.80911+205875.23850
[1100]  train-rmse:375589.71539+19566.97559     test-
rmse:1368918.56551+204935.82996
[1200]  train-rmse:349257.13216+15859.37876     test-
rmse:1366730.11108+203297.71671
[1300]  train-rmse:325308.27184+13025.27191     test-
rmse:1365600.23417+202121.56421
[1400]  train-rmse:302809.34006+11362.30371     test-
rmse:1364136.23722+200970.25694
[1500]  train-rmse:282005.92583+9487.32808      test-
rmse:1363208.78678+199406.67977
[1600]  train-rmse:263749.76846+7819.41576      test-
rmse:1362316.06255+198625.90066
[1610]  train-rmse:262018.96954+7742.22473      test-
rmse:1362382.12922+198567.95347
[0]     train-rmse:3978031.79744+106207.97446   test-
rmse:3977753.44184+226479.15722
[100]   train-rmse:1248745.58872+92606.70357    test-
rmse:1646493.33832+239105.42166
[200]   train-rmse:904591.02401+73968.62705     test-
rmse:1505622.38333+211284.00806
[300]   train-rmse:727679.84730+66292.46162     test-
rmse:1462294.76952+202792.85029
[400]   train-rmse:616156.73383+61664.51860     test-
rmse:1447805.11636+197219.92882
[500]   train-rmse:535950.80763+56300.16763     test-
rmse:1440216.01910+196990.70706
[600]   train-rmse:471992.74527+53704.25628     test-
rmse:1439148.71753+194216.39226
[626]   train-rmse:458183.64695+52270.28194     test-
rmse:1439232.00450+193360.81779
[0]     train-rmse:3975060.63967+106008.51476   test-
rmse:3974454.77499+228557.03379
[100]   train-rmse:1247946.14588+86255.35580    test-
rmse:1665299.74475+222225.38388
[200]   train-rmse:901868.40513+75082.27595     test-
```

```
rmse:1513483.83563+197845.33536
[300]	train-rmse:725082.21264+68646.16521	test-
rmse:1466005.43412+197298.04604
[400]	train-rmse:617047.47961+64738.15837	test-
rmse:1450227.88181+191227.39511
[500]	train-rmse:538284.40295+59413.04001	test-
rmse:1440286.98212+192480.34926
[600]	train-rmse:476692.25396+57885.64957	test-
rmse:1436645.82945+191843.26492
[700]	train-rmse:427307.67592+54636.27095	test-
rmse:1435350.04289+191429.16404
[737]	train-rmse:411263.14789+52587.23798	test-
rmse:1435956.65036+191096.84349
[0]	train-rmse:3984757.12494+106809.38134	test-
rmse:3985838.44425+226177.01859
[100]	train-rmse:1331189.64699+83768.44691	test-
rmse:1647793.48111+254199.11299
[200]	train-rmse:997806.40222+66781.86007	test-
rmse:1488428.56847+229620.07790
[300]	train-rmse:820876.59977+57749.60709	test-
rmse:1434257.57585+216740.94759
[400]	train-rmse:709729.59530+53927.52529	test-
rmse:1412130.72759+206956.76367
[500]	train-rmse:627523.84348+49761.25172	test-
rmse:1402497.62110+203519.27867
[600]	train-rmse:563851.25879+45790.70835	test-
rmse:1395961.07597+202246.39288
[700]	train-rmse:509540.10028+40941.08015	test-
rmse:1392063.94834+201005.92191
[800]	train-rmse:464995.89564+37105.48977	test-
rmse:1390047.33828+197339.24772
[900]	train-rmse:426708.10981+34861.87501	test-
rmse:1386681.07121+195064.15872
[1000]	train-rmse:393128.34315+32147.49630	test-
rmse:1385392.41189+192932.16615
[1100]	train-rmse:363571.15845+29416.86011	test-
rmse:1383479.45949+191619.70405
[1200]	train-rmse:337179.09100+27003.40852	test-
rmse:1383271.10611+190477.82924
[1268]	train-rmse:320922.72691+25821.72209	test-
rmse:1382731.02821+190024.99982
[0]	train-rmse:3992403.96377+106915.93132	test-
rmse:3992740.33130+226019.18639
[100]	train-rmse:1428979.74952+90263.31373	test-
rmse:1667233.66481+243986.47294
[200]	train-rmse:1108232.68972+73569.48101	test-
rmse:1492304.20052+217852.33233
[300]	train-rmse:942841.63809+63181.87466	test-
```

```
rmse:1431992.08613+206694.54979
[400]	train-rmse:838206.22716+57201.44562	test-
rmse:1400696.74763+202063.78003
[500]	train-rmse:758287.91466+51587.89689	test-
rmse:1386948.26794+199504.61293
[600]	train-rmse:696563.20358+46119.32101	test-
rmse:1379162.98271+198542.67495
[700]	train-rmse:643826.18608+41276.44584	test-
rmse:1371636.04107+197337.53571
[800]	train-rmse:601242.40969+38817.89750	test-
rmse:1365479.49925+199287.39000
[900]	train-rmse:564054.97774+36152.25326	test-
rmse:1360999.65899+197804.41170
[1000]	train-rmse:530983.91608+32408.56501	test-
rmse:1357040.21418+197366.07974
[1100]	train-rmse:500330.09741+29148.05657	test-
rmse:1354163.73257+197442.53700
[1200]	train-rmse:473446.11305+26681.67241	test-
rmse:1351172.95591+196829.87273
[1300]	train-rmse:449622.32073+24139.51927	test-
rmse:1348660.11430+195789.33249
[1400]	train-rmse:426752.22542+22186.84452	test-
rmse:1346112.74600+195905.78086
[1500]	train-rmse:406594.07579+20640.65933	test-
rmse:1344299.24611+195096.84050
[1600]	train-rmse:387985.98385+18832.22095	test-
rmse:1342151.83547+194244.79270
[1700]	train-rmse:370899.51843+17513.54765	test-
rmse:1340552.93239+194364.65011
[1782]	train-rmse:357566.62267+16522.56775	test-
rmse:1340465.62127+193425.69979
```

[275]:

```
inv_price_xgb1 = no_tune_train(params=inv_xgb_no_tune_params1,␣
 ↪x_train=final_inv_train,y_train=inv_train.price_doc)
inv_price_xgb2 = no_tune_train(params=inv_xgb_no_tune_params2,␣
 ↪x_train=final_inv_train,y_train=inv_train.price_doc)
inv_price_xgb3 = no_tune_train(params=inv_xgb_no_tune_params3,␣
 ↪x_train=final_inv_train,y_train=inv_train.price_doc)
inv_price_xgb4 = no_tune_train(params=inv_xgb_no_tune_params4,␣
 ↪x_train=final_inv_train,y_train=inv_train.price_doc)
inv_price_xgb5 = no_tune_train(params=inv_xgb_no_tune_params5,␣
 ↪x_train=final_inv_train,y_train=inv_train.price_doc)
```

```
[0]	train-rmse:4043486.19608+36468.84202	test-
rmse:4044329.11215+75340.12220
[100]	train-rmse:2249147.07222+23808.76128	test-
rmse:2381206.05804+76191.10618
[200]	train-rmse:1845647.60587+25251.86052	test-
```

```
rmse:2105554.70170+75403.99085
[300]   train-rmse:1665873.98608+25838.22730    test-
rmse:2021312.20021+72613.03314
[400]   train-rmse:1551986.08521+26007.90318    test-
rmse:1985194.00300+72208.09522
[500]   train-rmse:1463856.55551+25284.63476    test-
rmse:1964009.74085+72673.81490
[600]   train-rmse:1390000.65129+22823.39715    test-
rmse:1950926.57437+72698.40867
[700]   train-rmse:1326289.33063+20527.95404    test-
rmse:1941020.80665+74043.51078
[800]   train-rmse:1268927.08281+18281.34543    test-
rmse:1933900.38222+75395.58635
[900]   train-rmse:1217075.57841+17523.84799    test-
rmse:1928562.42327+75680.89146
[1000]  train-rmse:1168848.58893+16602.98446    test-
rmse:1924649.44471+75254.87038
[1100]  train-rmse:1125195.02139+17243.66110    test-
rmse:1922216.16729+75060.90572
[1200]  train-rmse:1083337.20123+16731.74384    test-
rmse:1919814.65123+75143.44725
[1300]  train-rmse:1044385.35173+15069.48272    test-
rmse:1918121.53693+74582.63198
[1400]  train-rmse:1008416.55665+14662.74867    test-
rmse:1917356.21028+74053.97044
[1500]  train-rmse:973388.73458+14571.86667     test-
rmse:1915639.89278+73864.41274
[1600]  train-rmse:940774.30977+14457.74543     test-
rmse:1915332.06271+73891.79529
[1700]  train-rmse:909161.28372+13439.87048     test-
rmse:1914032.87467+73727.51454
[1758]  train-rmse:891251.96502+13158.31824     test-
rmse:1914355.49274+73534.23542
[0]     train-rmse:4041471.41179+36545.53622    test-
rmse:4041871.39023+75163.62067
[100]   train-rmse:2180124.46646+23672.12454    test-
rmse:2343486.30162+75334.89217
[200]   train-rmse:1770879.89749+26434.86679    test-
rmse:2088696.62701+70151.31055
[300]   train-rmse:1594265.41564+26510.19340    test-
rmse:2021775.02678+70177.46108
[400]   train-rmse:1484458.46467+24560.78909    test-
rmse:1994530.70672+67368.51788
[500]   train-rmse:1402576.48665+25066.42280    test-
rmse:1980329.67462+65518.93756
[600]   train-rmse:1332983.15808+23742.16278    test-
rmse:1967989.74936+65259.13694
[700]   train-rmse:1277335.09354+21306.34515    test-
```

```
rmse:1960454.57306+67227.13531
[800]   train-rmse:1228226.04951+23214.15579    test-
rmse:1954142.82927+68029.48527
[900]   train-rmse:1186654.93469+23537.91964    test-
rmse:1949768.98062+69676.23347
[1000]  train-rmse:1149611.49689+23524.92588    test-
rmse:1946091.92848+71399.21782
[1100]  train-rmse:1115209.25906+26115.29020    test-
rmse:1943029.39941+71585.39492
[1200]  train-rmse:1080604.25419+27548.32816    test-
rmse:1940840.26185+72499.12017
[1300]  train-rmse:1048709.88782+28171.65430    test-
rmse:1938668.02241+72928.39297
[1400]  train-rmse:1021457.85193+31470.77282    test-
rmse:1937199.00448+73202.71581
[1500]  train-rmse:993718.34810+34352.60862     test-
rmse:1936021.80372+73648.59426
[1600]  train-rmse:965323.75885+36255.84834     test-
rmse:1934426.70583+73757.97814
[1700]  train-rmse:940535.13217+36819.36237     test-
rmse:1933278.27178+74201.38760
[1800]  train-rmse:917688.00554+37384.27542     test-
rmse:1932573.03721+74345.45829
[1900]  train-rmse:893951.71080+36788.34627     test-
rmse:1932148.92171+74343.92994
[2000]  train-rmse:870540.14355+35485.32042     test-
rmse:1931916.94930+74327.35828
[2005]  train-rmse:869668.32329+35648.72730     test-
rmse:1931878.97248+74262.25697
[0]     train-rmse:4041226.53873+36792.39114    test-
rmse:4042016.89226+74635.88386
[100]   train-rmse:2147387.76784+24249.80386    test-
rmse:2328734.07918+77243.79879
[200]   train-rmse:1733676.90007+27554.46697    test-
rmse:2082331.03388+73905.70690
[300]   train-rmse:1560548.05884+28406.66973    test-
rmse:2024539.83973+70593.55976
[400]   train-rmse:1452841.41044+25564.32407    test-
rmse:2001491.24006+66866.38264
[500]   train-rmse:1376052.44596+21962.81536    test-
rmse:1989048.14714+64405.62733
[600]   train-rmse:1313167.39070+19351.46242    test-
rmse:1981140.79211+63804.30732
[700]   train-rmse:1256474.96026+19755.16100    test-
rmse:1974529.85863+64606.73918
[800]   train-rmse:1209634.13501+17179.12316    test-
rmse:1969359.11122+66762.24200
[900]   train-rmse:1167652.46646+17089.40011    test-
```

```
rmse:1966219.20757+67356.59756
[1000]  train-rmse:1124627.47855+16742.56655    test-
rmse:1963054.65878+68078.35713
[1100]  train-rmse:1085194.11627+19453.75139    test-
rmse:1960317.06427+67625.94408
[1200]  train-rmse:1046791.20419+23744.55959    test-
rmse:1958135.99043+67150.89346
[1300]  train-rmse:1012028.63218+25252.97531    test-
rmse:1956874.38948+65752.36160
[1400]  train-rmse:982789.27433+23248.05093     test-
rmse:1955373.82594+64979.56943
[1500]  train-rmse:956043.48882+25576.63589     test-
rmse:1953972.27009+64989.37629
[1600]  train-rmse:931886.12399+27086.56076     test-
rmse:1952778.04328+64898.10389
[1700]  train-rmse:907490.03355+27734.03570     test-
rmse:1951976.37321+65107.49833
[1770]  train-rmse:890952.28457+27016.52955     test-
rmse:1952453.24044+65214.26672
[0]     train-rmse:4041984.42475+36868.35073    test-
rmse:4042764.12688+75021.88277
[100]   train-rmse:2175149.44773+24481.00680    test-
rmse:2339045.26282+76054.41297
[200]   train-rmse:1764710.81164+26031.60538    test-
rmse:2083766.72698+74729.70154
[300]   train-rmse:1588183.88334+24810.60626    test-
rmse:2015922.29775+74631.16914
[400]   train-rmse:1475264.71278+20932.07205    test-
rmse:1988053.93699+74137.51349
[500]   train-rmse:1388662.34406+19175.83817    test-
rmse:1970995.86367+73731.79423
[600]   train-rmse:1315728.30461+17071.22357    test-
rmse:1959296.49250+74657.90874
[700]   train-rmse:1253868.28067+15735.59066    test-
rmse:1951499.29773+75217.57252
[800]   train-rmse:1199866.25213+14479.32097    test-
rmse:1944826.47649+75851.14457
[900]   train-rmse:1149549.09536+14738.72244    test-
rmse:1940257.42801+76298.52679
[1000]  train-rmse:1101754.40605+15346.50042    test-
rmse:1936436.03673+76624.09108
[1100]  train-rmse:1058772.42677+15347.14280    test-
rmse:1933766.21159+76683.62407
[1200]  train-rmse:1017500.60026+16556.81313    test-
rmse:1931016.23058+76884.53109
[1300]  train-rmse:979042.80579+16947.82591     test-
rmse:1929333.04791+76346.96563
[1343]  train-rmse:962730.65365+16515.36384     test-
```

```
rmse:1929400.83698+76142.40635
[0]	train-rmse:4043522.05654+36766.05716	test-
rmse:4044544.21878+75053.53737
[100]	train-rmse:2224504.60118+24339.71964	test-
rmse:2368560.70577+76465.27177
[200]	train-rmse:1816567.09249+26049.02661	test-
rmse:2094882.35156+75024.60426
[300]	train-rmse:1636782.75636+27360.25282	test-
rmse:2014820.61809+72343.32953
[400]	train-rmse:1521394.28778+26136.25626	test-
rmse:1982381.55533+70592.63641
[500]	train-rmse:1431080.85251+24504.66126	test-
rmse:1962757.48940+71704.99363
[600]	train-rmse:1357123.29865+22810.52549	test-
rmse:1949418.15580+72945.04377
[700]	train-rmse:1292604.42354+19765.62509	test-
rmse:1941151.67791+74068.21655
[800]	train-rmse:1235670.29448+18664.83472	test-
rmse:1935334.65777+75699.22464
[900]	train-rmse:1183329.53238+18687.00806	test-
rmse:1930773.31688+76831.76962
[1000]	train-rmse:1135287.95086+18013.08607	test-
rmse:1927799.27970+76296.92360
[1100]	train-rmse:1090039.31820+17119.13112	test-
rmse:1924822.16660+76986.08356
[1200]	train-rmse:1046774.60390+17441.20928	test-
rmse:1922503.80137+76730.09554
[1300]	train-rmse:1007038.42853+16750.53477	test-
rmse:1921082.15892+76400.12381
[1400]	train-rmse:969586.98041+16741.59316	test-
rmse:1919998.46766+76617.30364
[1500]	train-rmse:934117.79187+16267.62226	test-
rmse:1919145.43514+77261.02644
[1600]	train-rmse:900575.23520+15485.12903	test-
rmse:1918614.26699+77186.80492
[1700]	train-rmse:868028.32141+15662.44445	test-
rmse:1918059.08445+77011.90647
[1800]	train-rmse:836743.47058+15422.11641	test-
rmse:1917894.21762+76857.20362
[1900]	train-rmse:807782.98975+14863.04531	test-
rmse:1916901.22863+77562.87533
[1957]	train-rmse:791963.81313+14666.68188	test-
rmse:1916835.61480+78059.77616
```

9.4 No Tune Top Features Models

[276]:
```python
def getTopFeatures(model,  n_features=100):
    scores = model.get_score()
    scores_items = scores.items()
    sorted_imp = sorted(list(scores_items), key=lambda x:x[1], reverse=True)
    return [f for f, v in sorted_imp][:n_features]
```

[277]:
```python
ocu_topf_xgb1 = no_tune_train(params=ocu_xgb_no_tune_params1,␣
 ↪x_train=final_ocu_train[getTopFeatures(ocu_pricesq_xgb1)],y_train=ocu_train.
 ↪price_sq)
ocu_topf_xgb2 = no_tune_train(params=ocu_xgb_no_tune_params2,␣
 ↪x_train=final_ocu_train[getTopFeatures(ocu_pricesq_xgb2)],y_train=ocu_train.
 ↪price_sq)
ocu_topf_xgb3 = no_tune_train(params=ocu_xgb_no_tune_params3,␣
 ↪x_train=final_ocu_train[getTopFeatures(ocu_pricesq_xgb3)],y_train=ocu_train.
 ↪price_sq)
#ocu_topf_xgb4 = no_tune_train(params=ocu_xgb_no_tune_params4,␣
 ↪x_train=final_ocu_train[getTopFeatures(ocu_pricesq_xgb4)],y_train=ocu_train.
 ↪price_sq)
#ocu_topf_xgb5 = no_tune_train(params=ocu_xgb_no_tune_params5,␣
 ↪x_train=final_ocu_train[getTopFeatures(ocu_pricesq_xgb5)],y_train=ocu_train.
 ↪price_sq)
```

```
[0]     train-rmse:38480.25836+81.13618 test-rmse:38509.61063+127.08556
[100]   train-rmse:13450.93157+236.11228        test-rmse:16551.25581+712.61958
[200]   train-rmse:10660.64047+224.24871        test-rmse:15417.84622+891.01862
[300]   train-rmse:9034.54293+216.08831 test-rmse:15002.09528+952.65054
[400]   train-rmse:7923.52501+190.03904 test-rmse:14835.55598+937.81474
[500]   train-rmse:7076.68285+153.40698 test-rmse:14772.91208+929.39184
[600]   train-rmse:6357.57283+124.30423 test-rmse:14746.82396+925.44247
[700]   train-rmse:5760.42348+111.06358 test-rmse:14718.41485+923.41680
[800]   train-rmse:5253.35127+91.99815  test-rmse:14708.35794+914.98522
[900]   train-rmse:4811.45094+81.62272  test-rmse:14703.31453+912.89991
[984]   train-rmse:4491.67784+81.37224  test-rmse:14701.78423+903.22484
[0]     train-rmse:38313.47596+72.78976 test-rmse:38342.36822+144.02281
[100]   train-rmse:12316.29503+250.01243        test-rmse:16048.47925+660.59775
[200]   train-rmse:9512.16455+238.50445 test-rmse:15043.77187+772.75257
[300]   train-rmse:7944.55545+116.26590 test-rmse:14748.35372+820.66088
[400]   train-rmse:6874.55676+58.06755  test-rmse:14656.63145+809.46041
[500]   train-rmse:6085.15487+39.49892  test-rmse:14609.27512+818.50555
[600]   train-rmse:5457.29092+53.22647  test-rmse:14595.59835+807.53000
[631]   train-rmse:5301.29442+64.90939  test-rmse:14592.37970+808.34383
[0]     train-rmse:38288.36350+76.36158 test-rmse:38323.12761+128.15257
[100]   train-rmse:12372.90869+255.93006        test-rmse:16099.24765+655.62374
[200]   train-rmse:9689.96468+311.59243 test-rmse:15051.00492+771.45411
[300]   train-rmse:8030.22770+119.78028 test-rmse:14729.33478+755.18571
[398]   train-rmse:6939.96994+83.70394  test-rmse:14710.85477+741.51071
```

[278]:

```
inv_topf_xgb1 = no_tune_train(params=inv_xgb_no_tune_params1,␣
 ↪x_train=final_inv_train[getTopFeatures(inv_pricesq_xgb1)],y_train=inv_train.
 ↪price_sq)
inv_topf_xgb2 = no_tune_train(params=inv_xgb_no_tune_params2,␣
 ↪x_train=final_inv_train[getTopFeatures(inv_pricesq_xgb2)],y_train=inv_train.
 ↪price_sq)
inv_topf_xgb3 = no_tune_train(params=inv_xgb_no_tune_params3,␣
 ↪x_train=final_inv_train[getTopFeatures(inv_pricesq_xgb3)],y_train=inv_train.
 ↪price_sq)
inv_topf_xgb4 = no_tune_train(params=inv_xgb_no_tune_params4,␣
 ↪x_train=final_inv_train[getTopFeatures(inv_pricesq_xgb4)],y_train=inv_train.
 ↪price_sq)
inv_topf_xgb5 = no_tune_train(params=inv_xgb_no_tune_params5,␣
 ↪x_train=final_inv_train[getTopFeatures(inv_pricesq_xgb5)],y_train=inv_train.
 ↪price_sq)
```

```
[0]     train-rmse:40188.62492+177.40797        test-rmse:40212.88273+347.07107
[100]   train-rmse:28392.54466+216.86992        test-rmse:29643.56770+518.69098
[200]   train-rmse:25253.95135+208.64492        test-rmse:27501.59138+578.58448
[300]   train-rmse:23733.77698+201.55973        test-rmse:26750.27888+614.49033
[400]   train-rmse:22685.10013+211.08082        test-rmse:26393.06148+610.95818
[500]   train-rmse:21798.86978+206.76134        test-rmse:26186.44478+605.41645
[600]   train-rmse:21018.03054+197.75076        test-rmse:26051.76119+599.71006
[700]   train-rmse:20292.82960+190.10698        test-rmse:25961.49427+596.47046
[800]   train-rmse:19624.90386+197.89920        test-rmse:25898.35222+591.47515
[900]   train-rmse:18997.72127+206.65107        test-rmse:25856.91614+594.80101
[1000]  train-rmse:18404.91357+205.97959        test-rmse:25827.42656+596.52622
[1100]  train-rmse:17838.09382+209.15900        test-rmse:25803.19312+608.44313
[1200]  train-rmse:17289.39440+197.81026        test-rmse:25781.71462+618.38652
[1300]  train-rmse:16777.89312+195.15464        test-rmse:25762.74809+619.28543
[1400]  train-rmse:16286.83209+195.65997        test-rmse:25748.90355+616.85970
[1500]  train-rmse:15807.54567+192.79624        test-rmse:25743.09850+615.38244
[1587]  train-rmse:15404.31244+178.15734        test-rmse:25741.66095+620.75626
[0]     train-rmse:40187.93095+174.97001        test-rmse:40214.55040+346.12069
[100]   train-rmse:27971.82087+220.13231        test-rmse:29411.08706+517.91060
[200]   train-rmse:24817.64186+205.57655        test-rmse:27371.62960+561.31726
[300]   train-rmse:23361.85378+156.89010        test-rmse:26668.24135+607.56686
[400]   train-rmse:22382.49977+150.43176        test-rmse:26341.30107+622.81527
[500]   train-rmse:21643.32341+157.54743        test-rmse:26145.00262+608.69736
[600]   train-rmse:21050.87644+151.45903        test-rmse:26030.61498+611.70209
[700]   train-rmse:20478.34004+149.68113        test-rmse:25937.11594+600.42143
[800]   train-rmse:19962.05830+165.21195        test-rmse:25873.23178+602.74874
[900]   train-rmse:19464.35173+172.27927        test-rmse:25839.53285+602.31145
[1000]  train-rmse:18978.42563+223.98332        test-rmse:25801.21838+604.52684
[1100]  train-rmse:18546.94747+219.40645        test-rmse:25780.87241+603.65471
[1200]  train-rmse:18133.13932+219.05227        test-rmse:25762.65042+607.65149
[1300]  train-rmse:17744.62011+225.07041        test-rmse:25750.76321+603.99949
```

```
[1400]  train-rmse:17365.01708+231.02701      test-rmse:25743.91109+601.96341
[1500]  train-rmse:16994.48046+226.72679      test-rmse:25735.96554+605.35267
[1600]  train-rmse:16630.78477+227.45289      test-rmse:25732.68458+606.77851
[1700]  train-rmse:16287.15180+219.07090      test-rmse:25726.22624+605.10600
[1800]  train-rmse:15954.08343+247.47148      test-rmse:25720.46310+602.05059
[1900]  train-rmse:15620.54710+245.79497      test-rmse:25717.95941+601.33518
[1960]  train-rmse:15392.44470+239.53008      test-rmse:25719.75958+603.33606
[0]     train-rmse:40170.65943+175.97934      test-rmse:40199.03484+346.54597
[100]   train-rmse:27756.86964+184.54660      test-rmse:29330.72921+562.90134
[200]   train-rmse:24570.25212+202.90736      test-rmse:27350.36134+564.50490
[300]   train-rmse:23077.73266+202.69012      test-rmse:26688.37384+586.75233
[400]   train-rmse:22120.46195+204.63214      test-rmse:26397.20994+594.67928
[500]   train-rmse:21347.00612+167.70847      test-rmse:26218.96759+586.35482
[600]   train-rmse:20718.28297+169.54591      test-rmse:26117.49068+584.63736
[700]   train-rmse:20151.09661+207.12199      test-rmse:26056.17381+602.39327
[800]   train-rmse:19616.32382+253.15304      test-rmse:26000.49501+600.71442
[900]   train-rmse:19106.41110+304.33337      test-rmse:25977.33688+600.20992
[1000]  train-rmse:18601.27208+329.12612      test-rmse:25950.42745+596.18063
[1100]  train-rmse:18131.82159+351.74902      test-rmse:25938.31369+608.93124
[1200]  train-rmse:17718.93856+372.16970      test-rmse:25923.79128+615.06662
[1300]  train-rmse:17254.15139+363.94660      test-rmse:25911.54838+617.77232
[1400]  train-rmse:16820.38421+361.56976      test-rmse:25908.86748+623.39152
[1500]  train-rmse:16381.06445+372.70024      test-rmse:25902.67643+629.53789
[1596]  train-rmse:16019.69431+367.42542      test-rmse:25900.33207+634.90301
[0]     train-rmse:40175.63966+178.41277      test-rmse:40205.92439+344.81681
[100]   train-rmse:27935.51560+200.47362      test-rmse:29421.82850+521.07143
[200]   train-rmse:24723.62504+193.45018      test-rmse:27343.74825+551.34975
[300]   train-rmse:23190.22962+173.88321      test-rmse:26640.06179+584.36887
[400]   train-rmse:22148.36304+179.04470      test-rmse:26311.72839+580.19395
[500]   train-rmse:21272.83296+216.44463      test-rmse:26122.11669+562.89769
[600]   train-rmse:20519.79434+240.15753      test-rmse:25998.86927+558.64119
[700]   train-rmse:19832.91416+269.24801      test-rmse:25912.42447+556.49551
[800]   train-rmse:19200.64319+254.82880      test-rmse:25852.46725+559.45630
[900]   train-rmse:18610.67244+268.17517      test-rmse:25815.32478+567.56869
[1000]  train-rmse:18026.14861+244.01982      test-rmse:25785.06333+573.92962
[1100]  train-rmse:17459.88193+240.07534      test-rmse:25761.43337+573.40287
[1200]  train-rmse:16925.49182+217.20829      test-rmse:25744.31568+578.00609
[1300]  train-rmse:16428.48231+209.37279      test-rmse:25730.98595+582.98151
[1400]  train-rmse:15925.97466+192.67954      test-rmse:25723.72613+582.24616
[1500]  train-rmse:15452.08746+162.71861      test-rmse:25719.40868+582.72074
[1599]  train-rmse:14994.87538+162.73872      test-rmse:25715.55738+580.02934
[0]     train-rmse:40186.19248+176.82888      test-rmse:40217.10260+346.45892
[100]   train-rmse:28190.51441+202.27347      test-rmse:29548.13205+558.61784
[200]   train-rmse:25021.05225+212.05065      test-rmse:27437.54462+577.76091
[300]   train-rmse:23494.40959+215.08911      test-rmse:26706.16056+595.33185
[400]   train-rmse:22421.10088+202.95816      test-rmse:26374.68863+592.99432
[500]   train-rmse:21519.00599+212.02507      test-rmse:26170.10374+590.25967
[600]   train-rmse:20709.72781+218.14082      test-rmse:26045.07051+598.49376
```

```
[700]	train-rmse:19966.02672+202.07606	test-rmse:25960.20179+597.80586
[800]	train-rmse:19254.93749+208.85360	test-rmse:25897.12068+590.44165
[900]	train-rmse:18606.31104+202.90210	test-rmse:25860.18920+589.13329
[1000]	train-rmse:18000.08812+193.48419	test-rmse:25827.76936+595.37427
[1100]	train-rmse:17413.36043+190.54787	test-rmse:25805.45991+605.92677
[1200]	train-rmse:16842.71905+169.44983	test-rmse:25792.03178+609.55155
[1300]	train-rmse:16313.13698+166.42451	test-rmse:25778.57865+614.42751
[1400]	train-rmse:15805.52165+156.78438	test-rmse:25775.89303+611.91107
[1410]	train-rmse:15753.86174+158.32541	test-rmse:25775.32894+611.60849
```

10 Predictions for Kaggle

[304]:
```
mn = 84/90
mn_inv =  77/90
```

10.1 No Tune Price SQ

[305]:
```
inv_pricesq_preds = [inv_pricesq_xgb1.predict(xgb.
 ↪DMatrix(inv_test))*inv_test['full_sq'],
inv_pricesq_xgb2.predict(xgb.DMatrix(inv_test))*inv_test['full_sq'],
inv_pricesq_xgb3.predict(xgb.DMatrix(inv_test))*inv_test['full_sq'],
inv_pricesq_xgb4.predict(xgb.DMatrix(inv_test))*inv_test['full_sq'],
inv_pricesq_xgb5.predict(xgb.DMatrix(inv_test))*inv_test['full_sq']
]


ocu_pricesq_preds = [ocu_pricesq_xgb1.predict(xgb.
 ↪DMatrix(ocu_test))*ocu_test['full_sq'],
ocu_pricesq_xgb2.predict(xgb.DMatrix(ocu_test))*ocu_test['full_sq'],
ocu_pricesq_xgb3.predict(xgb.DMatrix(ocu_test))*ocu_test['full_sq'],
#ocu_pricesq_xgb4.predict(xgb.DMatrix(ocu_test))*ocu_test['full_sq'],
#ocu_pricesq_xgb5.predict(xgb.DMatrix(ocu_test))*ocu_test['full_sq']
]
```

[306]:
```
# Average/ Median the predictions
ocu_pricesq_preds = np.median(np.array(ocu_pricesq_preds),axis=0)
inv_pricesq_preds = np.median(np.array(inv_pricesq_preds),axis=0)
```

[307]:
```
# Create the data frame of the predictions.
inv_pricesq_notune = pd.DataFrame(dict(id=new_inv[new_inv['price_doc'].
 ↪isnull()]['id'], price_doc = inv_pricesq_preds*mn_inv))
ocu_pricesq_notune = pd.DataFrame(dict(id=new_ocu[new_ocu['price_doc'].
 ↪isnull()]['id'], price_doc = ocu_pricesq_preds*mn))
xgb_pricesq_finals = pd.concat([ocu_pricesq_notune,inv_pricesq_notune])
```

10.2 No Tune Log Price

[283]:
```python
# Get the predictions for each model
inv_notunepreds = [np.expm1(inv_notue_xgb1.predict(xgb.DMatrix(inv_test))),
                   np.expm1(inv_notue_xgb2.predict(xgb.DMatrix(inv_test))),
                     np.expm1(inv_notue_xgb3.predict(xgb.DMatrix(inv_test))),
                     np.expm1(inv_notue_xgb4.predict(xgb.DMatrix(inv_test))),
                     np.expm1(inv_notue_xgb5.predict(xgb.DMatrix(inv_test)))]

ocu_notunepreds = [np.expm1(ocu_notue_xgb1.predict(xgb.DMatrix(ocu_test))),
                   np.expm1(ocu_notue_xgb2.predict(xgb.DMatrix(ocu_test))),
                   np.expm1(ocu_notue_xgb3.predict(xgb.DMatrix(ocu_test))),
                   np.expm1(ocu_notue_xgb4.predict(xgb.DMatrix(ocu_test))),
                   np.expm1(ocu_notue_xgb5.predict(xgb.DMatrix(ocu_test)))]
```

[284]:
```python
# Average the predictions
ocu_notune_preds = np.mean(np.array(ocu_notunepreds),axis=0)
inv_notune_preds = np.mean(np.array(inv_notunepreds),axis=0)
```

[285]:
```python
# Create the data frame of the predictions.
inv_xgb_notune = pd.DataFrame(dict(id=new_inv[new_inv['price_doc'].
 ↪isnull()]['id'], price_doc = inv_notune_preds*mn_inv))
ocu_xgb_notune = pd.DataFrame(dict(id=new_ocu[new_ocu['price_doc'].
 ↪isnull()]['id'], price_doc = ocu_notune_preds*mn))
xgb_notune_finals = pd.concat([ocu_xgb_notune,inv_xgb_notune])
```

10.3 No Tune Price Doc

[286]:
```python
inv_price_preds = [inv_price_xgb1.predict(xgb.DMatrix(inv_test)),
inv_price_xgb2.predict(xgb.DMatrix(inv_test)),
inv_price_xgb3.predict(xgb.DMatrix(inv_test)),
inv_price_xgb4.predict(xgb.DMatrix(inv_test)),
inv_price_xgb5.predict(xgb.DMatrix(inv_test))]


ocu_price_preds = [ocu_price_xgb1.predict(xgb.DMatrix(ocu_test)),
ocu_price_xgb2.predict(xgb.DMatrix(ocu_test)),
ocu_price_xgb3.predict(xgb.DMatrix(ocu_test)),
ocu_price_xgb4.predict(xgb.DMatrix(ocu_test)),
ocu_price_xgb5.predict(xgb.DMatrix(ocu_test))]
```

[287]:
```python
# Average the predictions
ocu_price_preds = np.mean(np.array(ocu_price_preds),axis=0)
inv_price_preds = np.mean(np.array(inv_price_preds),axis=0)
```

[288]:
```python
# Create the data frame of the predictions.
```

```
inv_xgb_price = pd.DataFrame(dict(id=new_inv[new_inv['price_doc'].
 ↪isnull()]['id'], price_doc = inv_price_preds*mn_inv))
ocu_xgb_price = pd.DataFrame(dict(id=new_ocu[new_ocu['price_doc'].
 ↪isnull()]['id'], price_doc = ocu_price_preds*mn))
xgb_price_finals = pd.concat([ocu_xgb_price,inv_xgb_price])
```

10.4 No Tune Top Features Price SQ

[316]:
```
inv_topf_preds = [inv_topf_xgb1.predict(xgb.
 ↪DMatrix(inv_test[getTopFeatures(inv_pricesq_xgb1)]))*inv_test['full_sq'],
inv_topf_xgb2.predict(xgb.
 ↪DMatrix(inv_test[getTopFeatures(inv_pricesq_xgb2)]))*inv_test['full_sq'],
inv_topf_xgb3.predict(xgb.
 ↪DMatrix(inv_test[getTopFeatures(inv_pricesq_xgb3)]))*inv_test['full_sq'],
inv_topf_xgb4.predict(xgb.
 ↪DMatrix(inv_test[getTopFeatures(inv_pricesq_xgb4)]))*inv_test['full_sq'],
inv_topf_xgb5.predict(xgb.
 ↪DMatrix(inv_test[getTopFeatures(inv_pricesq_xgb5)]))*inv_test['full_sq']]


ocu_topf_preds = [ocu_topf_xgb1.predict(xgb.
 ↪DMatrix(ocu_test[getTopFeatures(ocu_pricesq_xgb1)]))*ocu_test['full_sq'],
ocu_topf_xgb2.predict(xgb.
 ↪DMatrix(ocu_test[getTopFeatures(ocu_pricesq_xgb2)]))*ocu_test['full_sq'],
ocu_topf_xgb3.predict(xgb.
 ↪DMatrix(ocu_test[getTopFeatures(ocu_pricesq_xgb3)]))*ocu_test['full_sq'],
#ocu_topf_xgb4.predict(xgb.
 ↪DMatrix(ocu_test[getTopFeatures(ocu_pricesq_xgb4)]))*ocu_test['full_sq'],
#ocu_topf_xgb5.predict(xgb.
 ↪DMatrix(ocu_test[getTopFeatures(ocu_pricesq_xgb5)]))*ocu_test['full_sq']
]
```

[317]:
```
inv_topf_preds = np.mean(np.array(inv_topf_preds),axis=0)
ocu_topf_preds = np.mean(np.array(ocu_topf_preds),axis=0)
```

[318]:
```
inv_xgb_topf= pd.DataFrame(dict(id=new_inv[new_inv['price_doc'].
 ↪isnull()]['id'], price_doc = inv_topf_preds*mn_inv))
ocu_xgb_topf = pd.DataFrame(dict(id=new_ocu[new_ocu['price_doc'].
 ↪isnull()]['id'], price_doc = ocu_topf_preds*mn))
xgb_topf_finals = pd.concat([ocu_xgb_topf,inv_xgb_topf])
xgb_topf_finals['price_doc'] = xgb_topf_finals['price_doc']
```

10.5 Top 5 models - bayesian optimization

[292]:
```
top_inv_preds = [np.exp(model['model'].predict(inv_test)) - 1 for model in␣
↪top_xgb_inv]
top_ocu_preds = [np.exp(model['model'].predict(ocu_test)) - 1 for model in␣
↪top_xgb_ocu]

top_rf_inv_preds = [np.expm1(model['model'].predict(inv_test)) for model in␣
↪top5_rf_inv]
top_rf_ocu_preds = [np.expm1(model['model'].predict(ocu_test)) for model in␣
↪top5_rf_ocu]
```

[293]:
```
top_avg_inv_price = np.array(top_inv_preds).mean(axis=0)
top_avg_ocu_price = np.array(top_ocu_preds).mean(axis=0)

top_avg_inv_rf = np.array(top_rf_inv_preds).mean(axis=0)
top_avg_ocu_rf = np.array(top_rf_ocu_preds).mean(axis=0)
```

[294]:
```
inv_xgb_top = pd.DataFrame(dict(id = new_inv[new_inv['price_doc'].
↪isnull()]['id'], price_doc = top_avg_inv_price*mn_inv))
ocu_xgb_top = pd.DataFrame(dict(id = new_ocu[new_ocu['price_doc'].
↪isnull()]['id'], price_doc = top_avg_ocu_price*mn))
xgb_top_finals = pd.concat([ocu_xgb_top, inv_xgb_top])


inv_rf_top = pd.DataFrame(dict(id = new_inv[new_inv['price_doc'].
↪isnull()]['id'], price_doc = top_avg_inv_rf*mn_inv))
ocu_rf_top = pd.DataFrame(dict(id = new_ocu[new_ocu['price_doc'].
↪isnull()]['id'], price_doc = top_avg_ocu_rf*mn))
rf_top_finals = pd.concat([ocu_rf_top, inv_rf_top])
```

10.6 Top 1 Model - bayesian optimization

[295]:
```
inv_xgb_preds = np.exp(best_model_inv.predict(inv_test)) - 1
#inv_xgb_preds = np.exp(best_model_inv.predict(inv_test))*inv_test['full_sq']
ocu_xgb_preds = np.exp(best_model_ocu.predict(ocu_test)) - 1


inv_xgb = pd.DataFrame(dict(id = new_inv[new_inv['price_doc'].isnull()]['id'],␣
↪price_doc = inv_xgb_preds*mn_inv))
#inv_xgb = pd.DataFrame(dict(id=inv_done[inv_done['price_doc'].isnull()]['id'].
↪reset_index(drop=True), price_doc=inv_xgb_preds.reset_index(drop=True)))
ocu_xgb = pd.DataFrame(dict(id = new_ocu[new_ocu['price_doc'].isnull()]['id'],␣
↪price_doc = ocu_xgb_preds*mn))
xgb_finals = pd.concat([ocu_xgb, inv_xgb])
```

```
rf_inv_preds = np.exp(best_rf_inv.predict(inv_test)) - 1
# rf_inv_preds = best_rf_inv.predict(inv_test)*inv_test['full_sq']
rf_ocu_preds = np.exp(best_rf_ocu.predict(ocu_test)) - 1
inv_rf = pd.DataFrame(dict(id = new_inv[new_inv['price_doc'].isnull()]['id'],␣
 ↪price_doc = rf_inv_preds*mn_inv))
# inv_rf  =pd.DataFrame(dict(id=inv_done[inv_done['price_doc'].isnull()]['id'].
 ↪reset_index(drop=True), price_doc=rf_inv_preds.reset_index(drop=True)))
ocu_rf = pd.DataFrame(dict(id = new_ocu[new_ocu['price_doc'].isnull()]['id'],␣
 ↪price_doc = rf_ocu_preds*mn))
rf_finals = pd.concat([ocu_rf, inv_rf])
```

10.7 Ensemble

Top 1 Model - Investment (Bayesian Optimization) , Pricesq Investment Model

For OwnerOcuppier - pricesq model only.

[313]:
```
ensemble_inv = pd.DataFrame(dict(id=inv_xgb['id'],price_doc =␣
 ↪inv_xgb['price_doc']*0.12 + inv_pricesq_notune['price_doc']*0.88))
ensemble_inv = pd.concat([ocu_pricesq_notune,ensemble_inv])
```

11 Kaggle Submission Files

[308]:
```
xgb_pricesq_finals.to_csv("./price_sq_xgb.csv",index=False)
```

[314]:
```
# Ensemble
ensemble_inv.to_csv("./ensemble_inv_normalocu.csv",index=False)
```

[315]:
```
# Random Forest Predictions
rf_finals.to_csv("./rf_submission.csv", index=False)
# XGboost Predictions
xgb_finals.to_csv("./xgb_submission.csv", index=False)

# # XGboost (without algorithm for tuning)
xgb_notune_finals.to_csv("./xgb_notune.csv", index=False)

# # XGboost (without algorithm for tuning) TOP FEATURES

xgb_topf_finals.to_csv("./xgb_topf_notune.csv", index=False)

# # XGBOOST (without algorithm for tuning) PRICE SQ

xgb_pricesq_finals.to_csv("./price_sq_xgb.csv",index=False)

# # XGboost (without algorithm for tuning) PRICE DOC
```

```
xgb_price_finals.to_csv("./price_doc_xgb.csv", index=False)

# Top 5 xgboost models
xgb_top_finals.to_csv("./xgb_top5_submission.csv", index=False)

# Top 5 random forest models
rf_top_finals.to_csv("./rf_top5_submission.csv",index=False)

# # Weighted Average
pd.DataFrame(dict(id=rf_finals['id'], price_doc=(xgb_finals['price_doc']*0.
↪9999999 + rf_finals['price_doc']*0.01))).to_csv("./top1_rf_xgb_ensemble.
↪csv",index=False)
pd.DataFrame(dict(id=xgb_top_finals['id'],␣
↪price_doc=(xgb_top_finals['price_doc']*0.9999999 +␣
↪rf_top_finals['price_doc']*0.01))).to_csv("./top5_rf_xgb_ensemble.
↪csv",index=False)
pd.DataFrame(dict(id=xgb_notune_finals['id'],␣
↪price_doc=xgb_notune_finals['price_doc']*0.9 + xgb_finals['price_doc']*0.1)).
↪to_csv("./xgb_stack_ensemble.csv",index=False)
pd.DataFrame(dict(id=xgb_notune_finals['id'],␣
↪price_doc=xgb_notune_finals['price_doc']*0.9 + xgb_top_finals['price_doc']*0.
↪1)).to_csv("./xgb_stack_top5_ensemble.csv",index=False)
pd.DataFrame(dict(id=xgb_pricesq_finals['id'],␣
↪price_doc=xgb_pricesq_finals['price_doc']*0.9+␣
↪xgb_notune_finals['price_doc']*(1-0.9))).to_csv("./
↪xgb_pricesq_stack_ensemble.csv",index=False)
```

12 Bibliography

Kapoor, S., & Perrone, V. (2021). A simple and fast baseline for tuning large XGBoost models. arXiv preprint arXiv:2111.06924.

Putatunda, S., & Rama, K. (2018). A Comparative Analysis of Hyperopt as Against Other Approaches for Hyper-Parameter Optimization of XGBoost. International Conference on Signal Processing and Machine Learning.