

תשובה לשאלת 2: פירוט המחברת Basic Time Series Analysis & Feature Selection • שינוי price ללוג שלו, פעולה זאת בעצם מקלה על הקריאה של המחיר ווהופכת אותה לפешטה וגורמת למשתנה להתקרב לנורמלית. • בדיקת שינוי מחירים לאחר זמן, יש שלושה פיצ'רים של זמן (כל החודשים, ממוצע חדשים וכל הימים) על מנת לבדוק את הטrndים נוצר `zbox` לכל פיצ'ר וכן לבדוק וייזואלית האם אפשר להסיק מסקנות על טrndים לאחר זמן מסוים. ניתן להסיק מגרפים אלו היא אין טrnd שנייה להבין לפי התפלגות המחיר לאחר הזמן. • מחיקת ערכים חסרים, מחיקת כל הפיצ'רים עם יותר מ-20% NA. פעולה זאת באה לצמצם את כמות הפיצ'רים בשכיב לפשט את המודל. • בחירת פיצ'רים: (1) קורלציה – בדיקת קורלציה בין משתנה המטרה לשאר הפיצ'רים על מנת למצוא את 50 הפיצ'רים עם הקורלציה הגבוהה. נוסף חסם תחתון לקורלציה, ככלומר כל משתנה עם קורלציה גבוהה מ-0.01 נשאר. פעולה זאת צמצמה את הפיצ'רים ל-40 והשיאה רק את הרלוונטיים ביותר. לאחר מכן, בוצעה בדיקת קורלציה בין הפיצ'רים לעצמם, זה בדק האם יש פיצ'רים עם התאמה מלאה, ככלומר פיצ'ר שמתאר את אותו דבר. לאחר בדיקה זאת הtagלו 5 פיצ'רים אשר נמחקו ולבסוף נשארו 35 פיצ'רים חשובים. (2) XGBoost – בוצעה רגסיטיון XGBoost על הדאטה וממנה נבחרו 99 פיצ'רים החשובים ביותר. התגלה נקודה מעניינת שלמרות שלא ראיינו טrndים ברורים לאחר זמן, הפיצ'רים של הזמן הtagלו כפיצ'ר חשוב. (3) לבסוף נוצרה רשימה שהיא חיבור בין הפיצ'רים של הקורלציה לבין הפיצ'רים של המודל XGBoost. שילוב הרשימה בעצם נותן מידע על קשרים ליניאריים (קורלצייה) ומידע על קשרים יותר מסובכים (xgboost). • פעולה זאת נעשתה על מנת לשפר את המודל הסופי ע"י הוספה חמימדים שנוצרו מקשרים. • הכתנת הדאטה הסופי עם הפיצ'רים הנבחרים: (1) טיפול בערכים חסרים – מותך כל אחד מ-126 הפיצ'רים שנבחרו והחלפו הערכים החסרים בערכים עם כמות ההופעות הגבוהה ביותר (most_frequent). (2) פיתוח דאטה פריים חדש עם הפיצ'רים המלאים והפיכתם לדאטה מנורמל בעזרת רגולציה L2. המטרה בפעולה זו היא הקטנת הסיבוכיות וגורמת לפיצ'רים מסוימים להיות פחות דומיננטיים ובכך מוביל לביצועים טובים יותר של המודל. • ביצוע `kfold`, שהוא טכניקת חלוקה של סט האימון לביצוע cross validation. תהליך זה לפי-h-k מחלק את הסט המלא לא סטים באופן רנדומלי – 1-k סטים של אימון אחד לוולידציה ומבצע זאת k פעמים. לאחר מכן המודל מאמת את סט האימון והערכתו ניתנת על ידי הוולידציה, את הערכה עושים לפי מzdse. בנוסף לכך המודל מתאמן 1000 סבבים או שנוצר מוקדם יותר אם ההערכה של האימון לא משתפרת במשך 50 סיבוכים. פעולה העצירה נוצרה בשכיב לcker את תהליכי ההערכתה ולמנוע overfitting. ולטיסום כל תהליכי בוצע על מנת שנוכל להעריך את ביצוע המודל על נתונים שהוא עדין לא ראה. נקודות לשיפור וביקורת: • גרפ' מחיר לאחר זמן לפי ימים – הגרף לא אינטואיטיבי בגלל כמות הימים הגדולה בציר-x והוא גם הוסיף שאין לו מהימנות בגרף לאחר מכן בו הוא הראה שאין מספיק כמות של מכירות במספר רב של ימים. וכך ניתן היה להוויד אותו. • הורדת הפיצ'רים שיש בהם מעל 20 אחוז ערכים חסרים – ניתן היה לבחור בשיטות אחרות על מנת להתרמודד עם הערכים החסרים כמו הוספה חציון או סוג של imputer regression שיכלו למלא את הערכים החסרים. יכול להיות שבקבוקות הורדת הפיצ'רים הורדו פיצ'רים שיכלו לתרום למודל.

פירוט מחברת Sberbank – Simple Exploration Notebook: • התבוננות על משתנה המטרה: תחילת נוצר גרפ' scatter plot על מנת לבדוק outliers ווגראה שאין נקודות חריגות. לאחר מכן בוצעה היסטוגרמה אך הוא היה מוטה עם זנב ימני ולכן נעשה פונקצייתelog שבסוצותה נוח לראות את ההתפלגות. היה פחות מוטה ונראית כמו ההתפלגות נורמלית. • גרפ' של עליית המחיר: גרפ' זה מראה את ההתפלגות המוחרים לאחר החציון בכל חדש. ניתן ללמידה שישנה מגמה של עלייה לינארית מחודש מסויים ועד הסוף. • התבוננות על הדאטה: תחילת נבחנו סוגי הפיצ'רים ורוכם היי מספריים. בהמשך נבדקו כמות הערכים מכל הפיצ'רים המספריים. תהליך זה הוא למעשה הבנה בפעולה עתידית. בהמשך שונו המשתנים הקטגוריאליים לערכים מספריים בעזרת קידוד ומיפוי כדי שהמודול יוכל לעבוד עם משתנים אלה. • אימון מודל `gbm` על הדאטה: תהליכי זה בוצע על מנת למצוא את הפיצ'רים החשובים ביותר. מפיצ'רים אלו נלקחו ארבעת החשובים

באותה. • חקירת ארבעת הפיצ'רים הנחשבים: לפיצ'ר `qs_full` בוצעה בדיקה scatter plot על מנת לבדוק האם יש קשר ויזואלי והאם אפשר לצפות outliers.outliers. לפיצ'ר `qs_life` בוצעה בדיקה עם `kde` על מנת ליזור גרפף צפיפות. ובדרך זאת ניתן לראות קורלוציה בין שטח הדירה למחיר. לפיצ'ר `floor` בוצעה בדיקה של חציון המחיר לפי קומה וניתן היה לראות שכאש מס' הקומה גדול מ'מחיר עולה. ולפי'צ'ר `max_floor` בוצעה בדיקת boxplot על מנת לראות ויזואלית את האחוזונים. שני הגרפים הראשונים בוצעה הורדת outliers של כל פיצ'ר לפי אחוזון מסוים שנבחר. בסה"כ גראפים אלו בוצעו על מנת לראות קשר ומגמות בין פיצ'רים אלה למשתנה המטרה. נקודות לשיפור וביקורת: • הchèלת משתנים קטגוריאליים בערכים מסוימים – דרך זאת יכולת לייצר בעיה בכך שהיא מייצרת סדר למשתנים הקטגוריאליים אך הסדר הוא לא בהכרח קיים. דרך אשר יכולה לשפר היא שימוש בOne-HotEncoding כדי שהמודול יבין שלא קיים סדר בכל קטגוריה וכי רוב הפיצ'רים הקטגוריאליים הם איכוטיים בדעתה הנתון מה שפחות מתאים לLabelEncoder שבוצע. • בבדיקה הNA התגללה כמות גדולה של ערכים חסרים, למורות שמודל XGBOOST יודע לטפל בערכים אלה, עדיף לטפל בערכים אלה לפני הכנסתם למודל בעזרות שיטות שונות כמו מחיקת הפיצ'רים עם כמות גבוהה של ערכים חסרים, הכנסת החציון או הממוצע במקום ועוד. • הגרפים שייצאו `mbq_life` לא הראו מגמות מסוימת דבר זה נראה נוצר בכך שאת הערכים החסרים החליפו באפס ומספר הערכים החסרים היה גבוה מה שהראה קורלציה גבוהה בערך 0. בנוסף לכך של `max_floor` היה טיפול לקיי ב outliers מה שגרם לכך שהוא מגמות שונות לאורך הזמן וכיוון המגמה החזוי של עליית המחיר נהפר ללא אישר גובה הקומה המקורי עלה.

התחלת שאלת 3

```
In [ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor
from scipy import stats
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder
import category_encoders as ce
from sklearn.metrics import mean_squared_error
import pickle
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
```

Uploading the data

```
In [ ]: dtrain = pd.read_csv('C:\\Users\\odedw\\OneDrive\\סטטיסטיקה\\למידה\\פרויקט סיום\\dataset.csv')
```

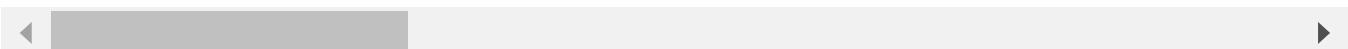
תחילת חקירת הדטה - Exploratory Data Analysis

```
In [ ]: # אוחזונים למשתנים מסוימים
numerical_stats = dtrain.describe()
numerical_stats
```

Out[]:

	id	full_sq	life_sq	floor	max_floor	material	bu
count	30471.000000	30471.000000	24088.000000	30304.000000	20899.000000	20899.000000	1.6866
mean	15237.917397	54.214269	34.403271	7.670803	12.558974	1.827121	3.0680
std	8796.501536	38.031487	52.285733	5.319989	6.756550	1.481154	1.5438
min	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.0000
25%	7620.500000	38.000000	20.000000	3.000000	9.000000	1.000000	1.9670
50%	15238.000000	49.000000	30.000000	6.500000	12.000000	1.000000	1.9790
75%	22855.500000	63.000000	43.000000	11.000000	17.000000	2.000000	2.0050
max	30473.000000	5326.000000	7478.000000	77.000000	117.000000	6.000000	2.0052

8 rows × 276 columns



In []:

```
# שינוי הערך החיריג ממוצע של הפיצ'ר #
dtrain.loc[dtrain['state'] == 33, 'state'] = 4

# שינוי הערך החיריג ממוצע של השנים
dtrain.loc[dtrain['build_year'] == 20052009, 'build_year'] = 2007
```

In []:

```
# הבנה של הדאטה והכרת סוג המשתנים
numerical_columns = dtrain.select_dtypes(exclude=['object']).columns
categorical_columns = dtrain.select_dtypes(include=['object']).columns
print('Number of numerical columns:', len(numerical_columns))
print('Number of categorical columns:', len(categorical_columns))
```

Number of numerical columns: 276

Number of categorical columns: 16

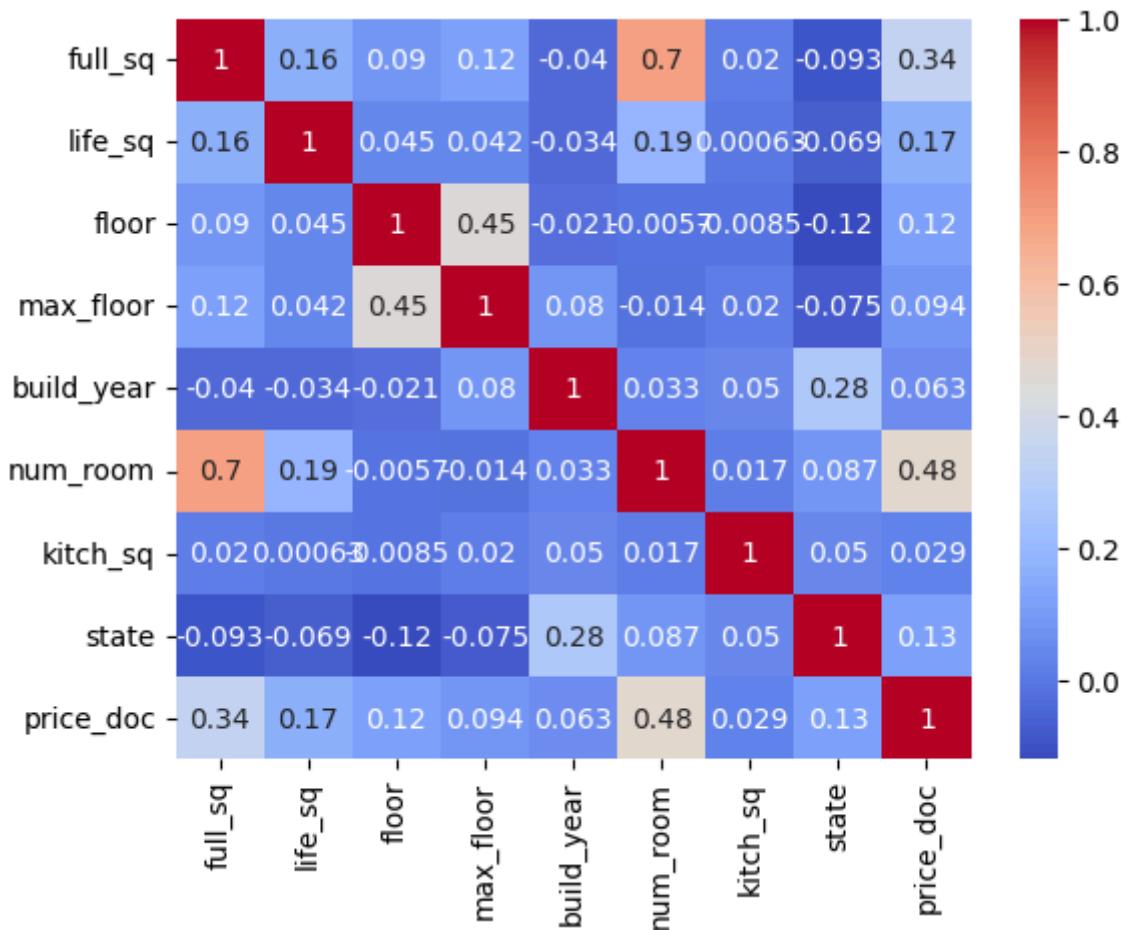
Housing Internal Characteristics

In []:

```
# בדיקת קורלציה לפיצרים ביחסים
internal_chars = ['full_sq', 'life_sq', 'floor', 'max_floor', 'build_year', 'num_rooms']
corr = dtrain[internal_chars].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[]:

<AxesSubplot:>



פלוט קורלציה - ללחנו מפה את הפיצרים עם הקורלציה הגבוהה ביותר עם המחיר ובצם חקמנו אותו בונספ' חקמנו גם היקמה ומקום קומה כי גם להם היה קורלציה גבוהה מאוד

```
In [ ]: #בדיקה האם יש ערכים לא הגיונית
invalid_rows = dtrain[dtrain['life_sq'] > dtrain['full_sq']]
#בדיקה ויזולאות
invalid_rows[['full_sq', 'life_sq']]
```

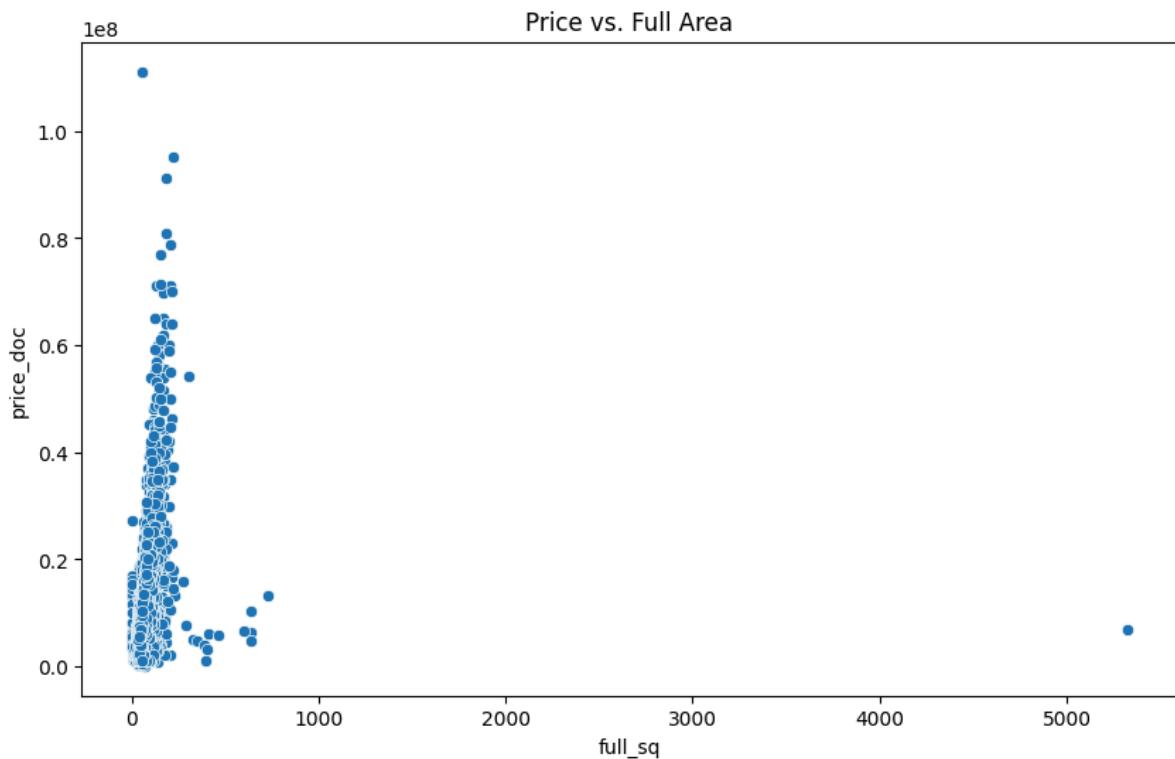
Out[]:

	full_sq	life_sq
1084	44	281.0
1188	9	44.0
1822	18	38.0
1863	30	178.0
2009	5	40.0
4385	73	426.0
6336	37	191.0
6531	80	88.0
6993	73	77.0
7208	31	195.0
8101	37	38.0
9237	47	301.0
9256	77	458.0
9482	52	53.0
9646	82	802.0
11332	1	40.0
11711	56	58.0
11784	46	59.0
12569	56	60.0
13546	79	7478.0
13629	45	259.0
13797	32	163.0
14799	73	77.0
16067	50	52.0
16116	84	85.0
16284	33	62.0
20672	44	45.0
21080	52	349.0
22412	1	47.0
22611	37	38.0
22804	54	55.0
24296	0	77.0
24428	74	78.0
26264	1	60.0
26342	75	435.0
26363	1	64.0

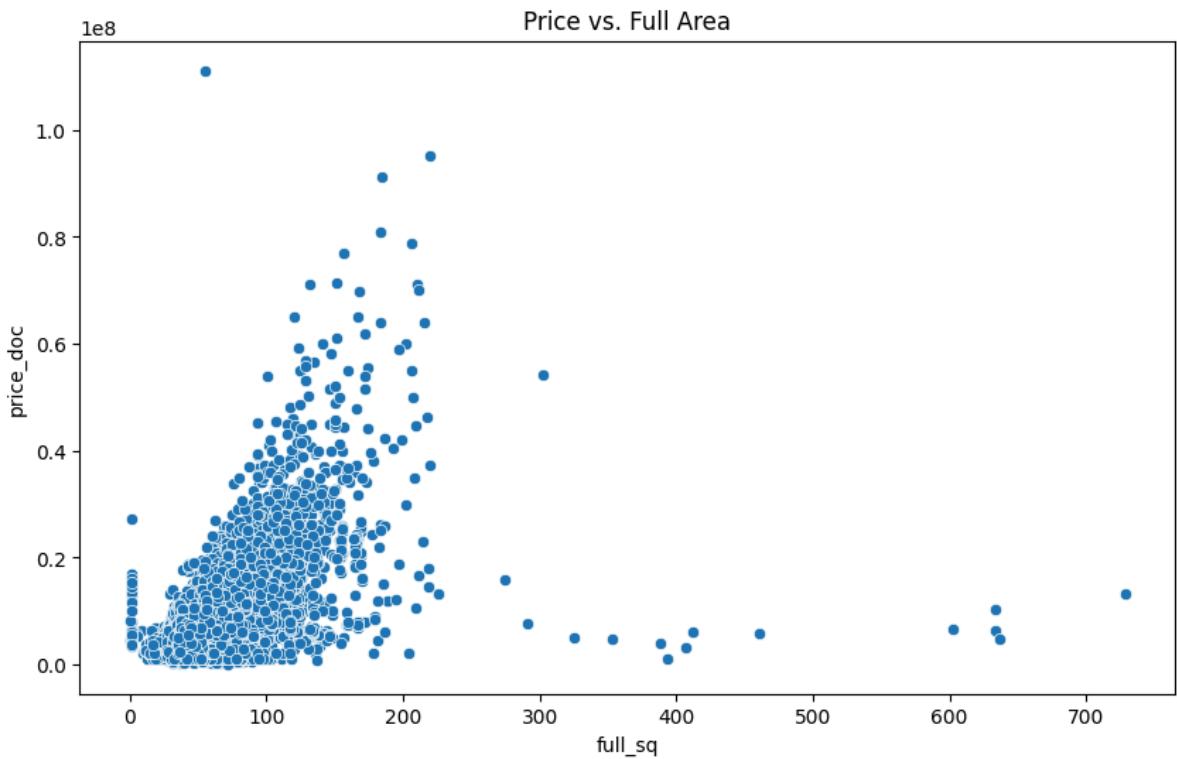
	full_sq	life_sq
29302	56	57.0

```
In [ ]: #להמיר את הערכים האלאה לNA
# Replace 'life_sq' values greater than 'full_sq' with NaN
dtrain.loc[dtrain['life_sq'] > dtrain['full_sq'], 'life_sq'] = np.nan
```

```
In [ ]: # full_sq against price_doc
plt.figure(figsize=(10, 6))
sns.scatterplot(x='full_sq', y='price_doc', data=dtrain)
plt.title('Price vs. Full Area')
plt.show()
```



```
In [ ]: # שיפור הגרף ללא האוטלירים
plt.figure(figsize=(10, 6))
sns.scatterplot(x='full_sq', y='price_doc', data=dtrain[dtrain['full_sq'] < 2000])
plt.title('Price vs. Full Area')
plt.show()
```



```
In [ ]: #משני ערכים חירגים לNA
for col in ['life_sq', 'full_sq']:
    dtrain.loc[dtrain[col] < 5, col] = np.nan

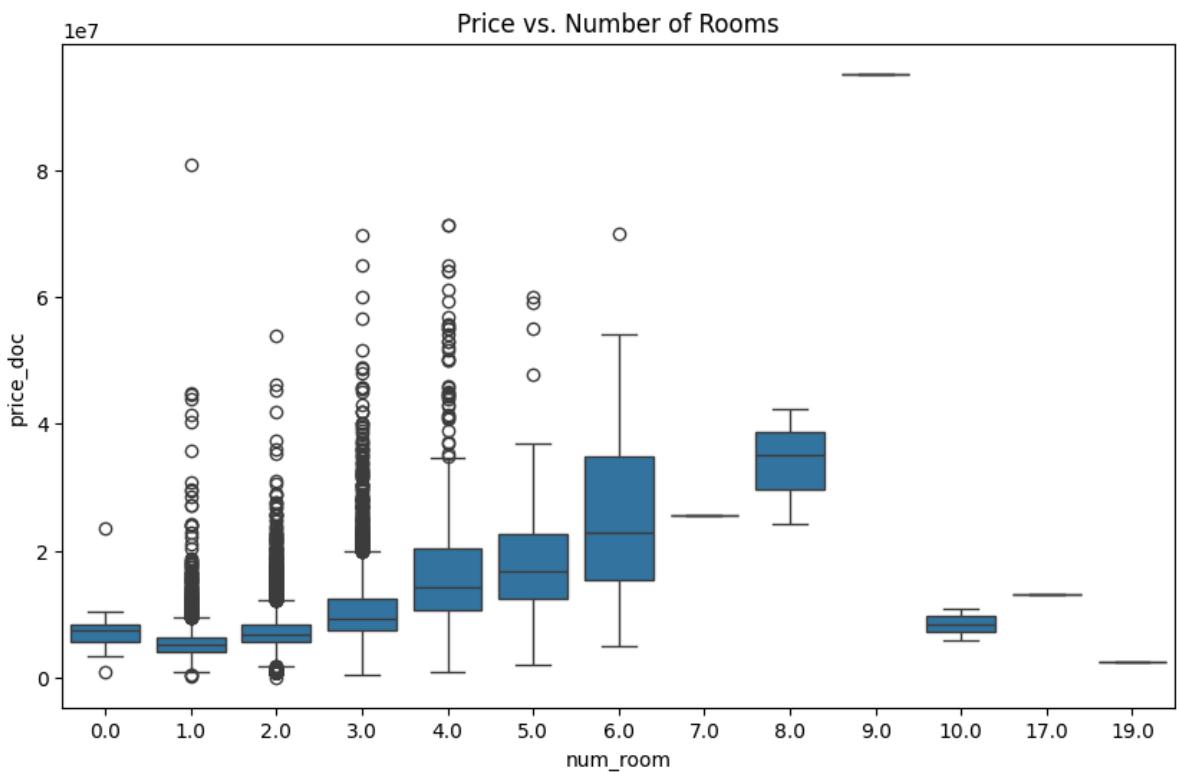
#הערך של הדירה עצמה או האם גודל המטבח היה שווה ל0 או 1 ANהחלפת הערכים החירגים במשתנה גודל המטבח ב
dtrain.loc[(dtrain['kitch_sq'] >= dtrain['life_sq']) | dtrain['kitch_sq'].isin([0, 1]), 'kitch_sq'] = 1

#החלפת ערכי מג'יק נאמרים ב
dtrain.loc[(dtrain['full_sq'] > 210) & (dtrain['life_sq'] / dtrain['full_sq'] < 0.3), 'life_sq'] = 0.3

#ערכים גדולים מ 300 הוחלפו ב
dtrain.loc[dtrain['life_sq'] > 300, ['life_sq', 'full_sq']] = np.nan
```

נראה יותר טוב ורואים קשר בין המחיר לגודל הדירה

```
In [ ]: #num_room against price_doc
plt.figure(figsize=(10, 6))
sns.boxplot(x='num_room', y='price_doc', data=dtrain)
plt.title('Price vs. Number of Rooms')
plt.show()
```



```
In [ ]: #num_room value counts
dtrain['num_room'].value_counts()
```

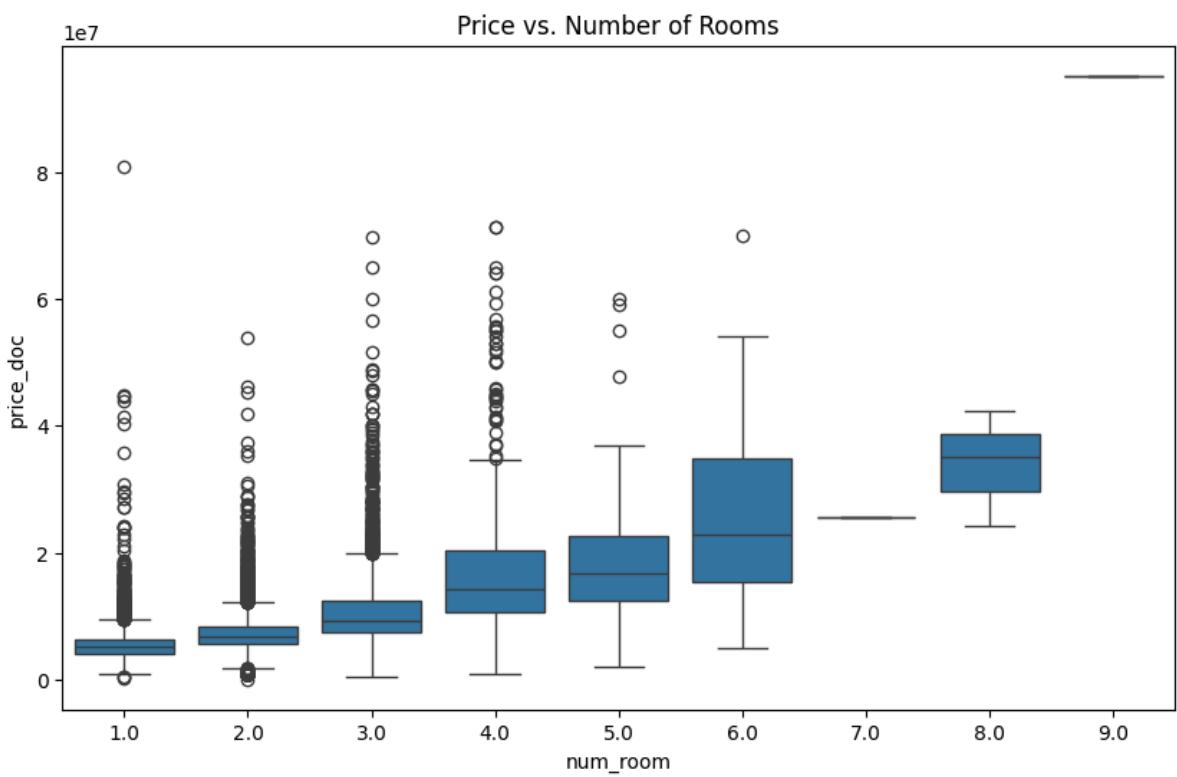
```
Out[ ]: num_room
2.0      8132
1.0      7602
3.0      4675
4.0      418
5.0       40
0.0       14
6.0        9
8.0        3
10.0      2
19.0      1
7.0        1
17.0      1
9.0        1
Name: count, dtype: int64
```

```
In [ ]: #After the outliers were removed, we can see that there is a positive correlation between the number of rooms and the price
dtrain.loc[dtrain['num_room'] == 0, 'num_room'] = np.nan
dtrain.loc[dtrain['num_room'] == 10, 'num_room'] = np.nan
dtrain.loc[dtrain['num_room'] == 17, 'num_room'] = np.nan
dtrain.loc[dtrain['num_room'] == 19, 'num_room'] = np.nan
```

```
In [ ]: #Checking if there is a positive correlation between the number of rooms and the price
rows_with_9_rooms = dtrain[dtrain['num_room'] == 9]
rows_with_9_rooms[['full_sq', 'life_sq', 'num_room']]
```

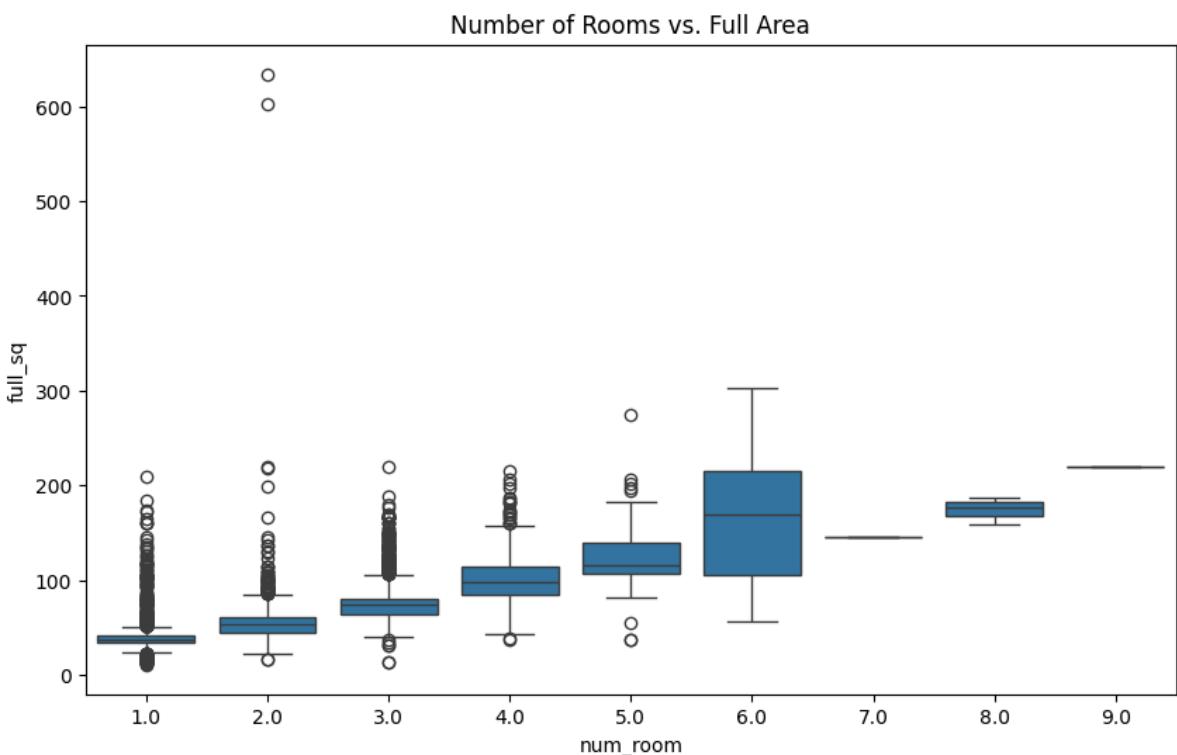
```
Out[ ]: full_sq  life_sq  num_room
28326     220.0    144.0      9.0
```

```
In [ ]: #num_room against price_doc
plt.figure(figsize=(10, 6))
sns.boxplot(x='num_room', y='price_doc', data=dtrain)
plt.title('Price vs. Number of Rooms')
plt.show()
```



פלוט-הגרף בחרור יותר וראהים קשר ישיר בין גודל החדרים לעלייה המהירה

```
In [ ]: # num_room against full_sq
plt.figure(figsize=(10, 6))
sns.boxplot(x='num_room', y='full_sq', data=dtrain)
plt.title('Number of Rooms vs. Full Area')
plt.show()
```



```
In [ ]: # אונטו בדיקה כמו הקיים #
invalid_rows = dtrain[dtrain['floor'] > dtrain['max_floor']]

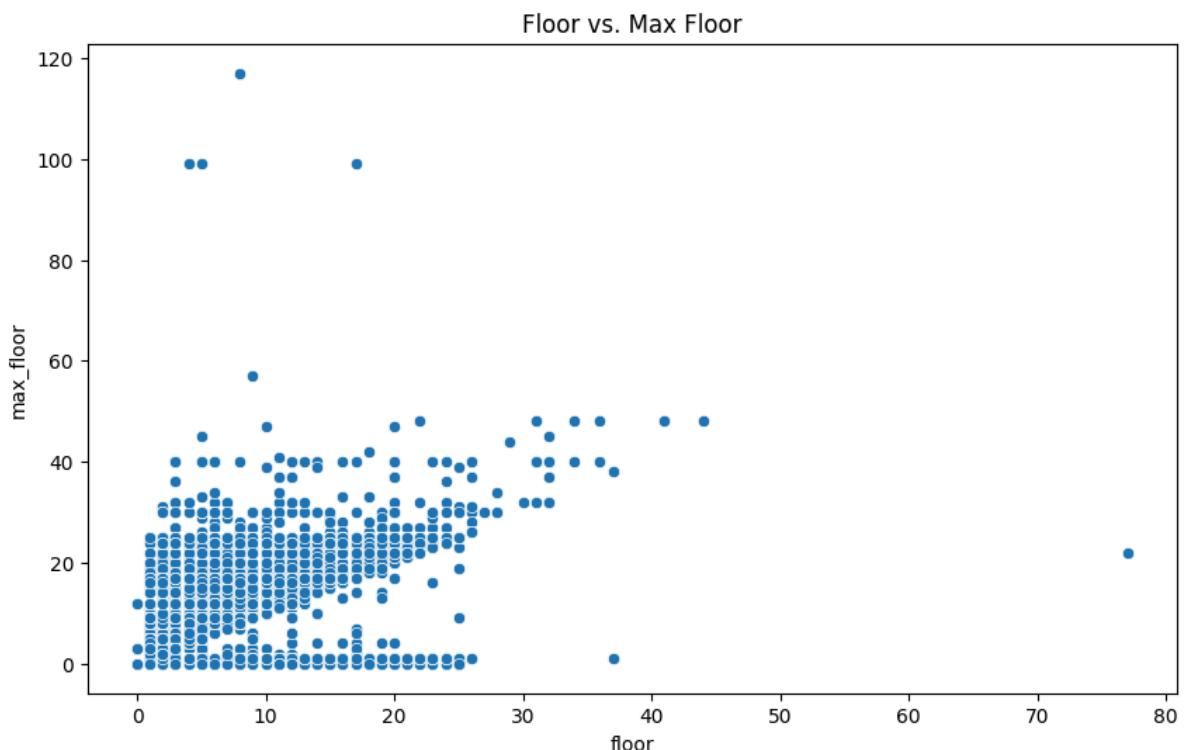
# Print the invalid rows
invalid_rows[['floor', 'max_floor', 'price_doc']]
```

Out[]:

	floor	max_floor	price_doc
8216	13.0	0.0	5813760
8268	3.0	1.0	5427640
8499	2.0	0.0	6000000
8531	7.0	0.0	3842500
8912	5.0	0.0	3850000
...
30398	5.0	1.0	4185400
30400	3.0	0.0	16748512
30426	1.0	0.0	7127255
30439	12.0	0.0	8535937
30450	5.0	0.0	12610000

1493 rows × 3 columns

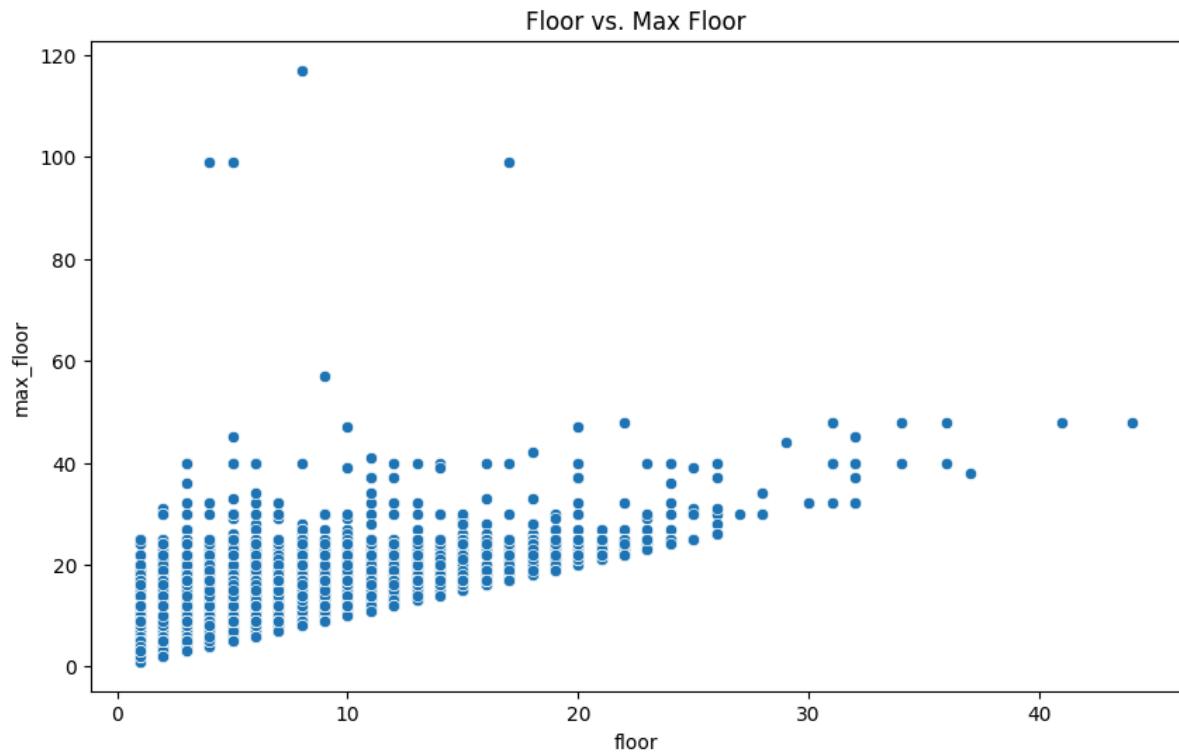
```
In [ ]: # Floor against max_floor
plt.figure(figsize=(10, 6))
sns.scatterplot(x='floor', y='max_floor', data=dtrain)
plt.title('Floor vs. Max Floor')
plt.show()
```



```
In [ ]: שינוי ערכים אלה להרים #
dtrain.loc[dtrain['floor'] > dtrain['max_floor'], 'max_floor'] = np.nan
# Replace 'max_floor' and 'floor' values where both are 0 with NaN
dtrain.loc[(dtrain['floor'] == 0) & (dtrain['max_floor'] == 0), ['max_floor', 'floor']] = np.nan
# Replace 'floor' values where 'floor' is 0 with NaN
dtrain.loc[dtrain['floor'] == 0, 'floor'] = np.nan
```

```
# Replace 'max_floor' values where 'max_floor' is 0 with NaN
dtrain.loc[dtrain['max_floor'] == 0, 'max_floor'] = np.nan
```

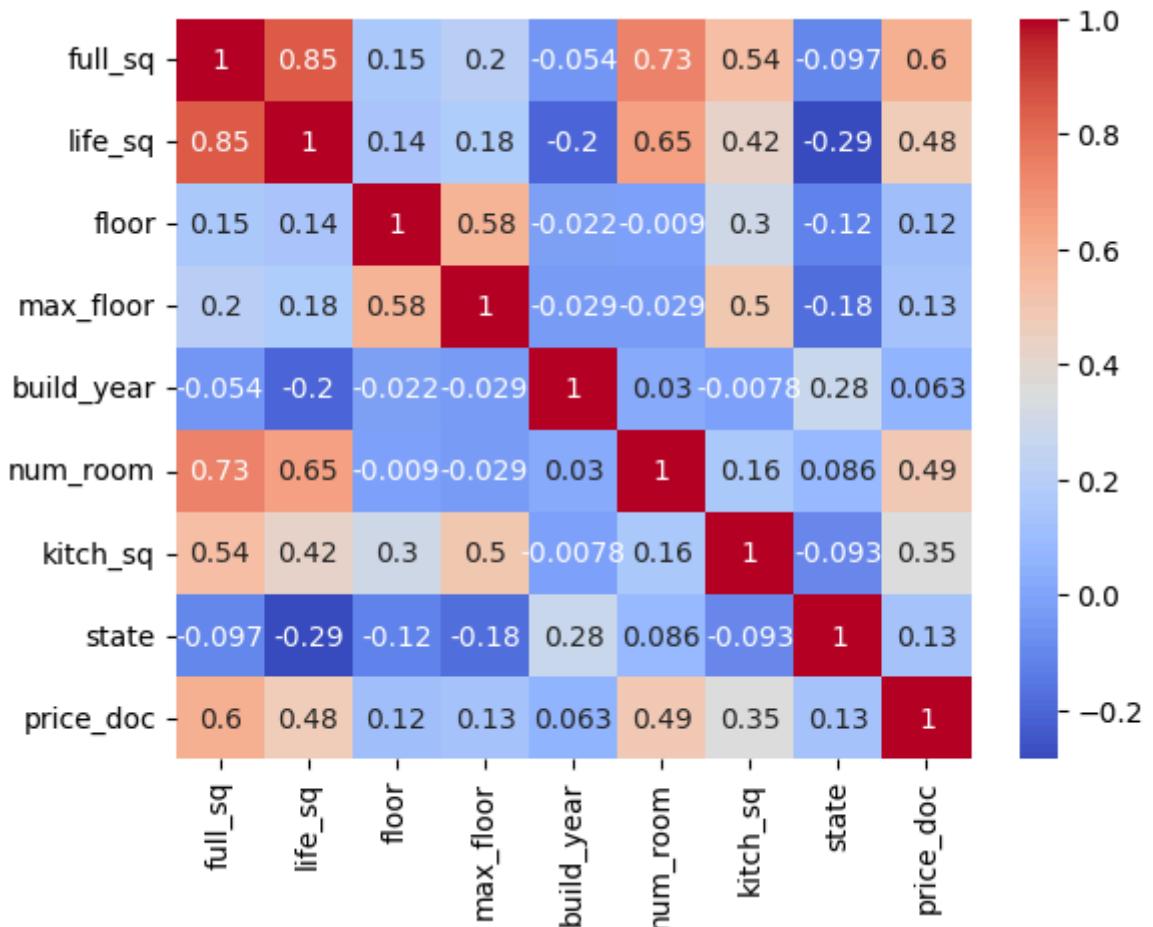
```
In [ ]: # floor against max_floor
plt.figure(figsize=(10, 6))
sns.scatterplot(x='floor', y='max_floor', data=dtrain)
plt.title('Floor vs. Max Floor')
plt.show()
```



פלוט-אפשר לראות קשר לינארי אך קשה יותר בഗל האוטליירם

```
In [ ]: internal_chars = ['full_sq', 'life_sq', 'floor', 'max_floor', 'build_year', 'num_rms']
corr = dtrain(internal_chars).corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

```
Out[ ]: <AxesSubplot:>
```



עשינו עוד פעם את אותן קורלציות לראות אם השינויים שעשינו באמת השפיעו על הדברים ראיין שacademic השינויים שעשינו שיפורו את הקורלציות והסקנו מפה דברים חשובים

```
In [ ]: # בדיקה האם יש שינוי בין החזינן של המהירים לפי סוג הנכס
dtrain.groupby('product_type')['price_doc'].median()
```

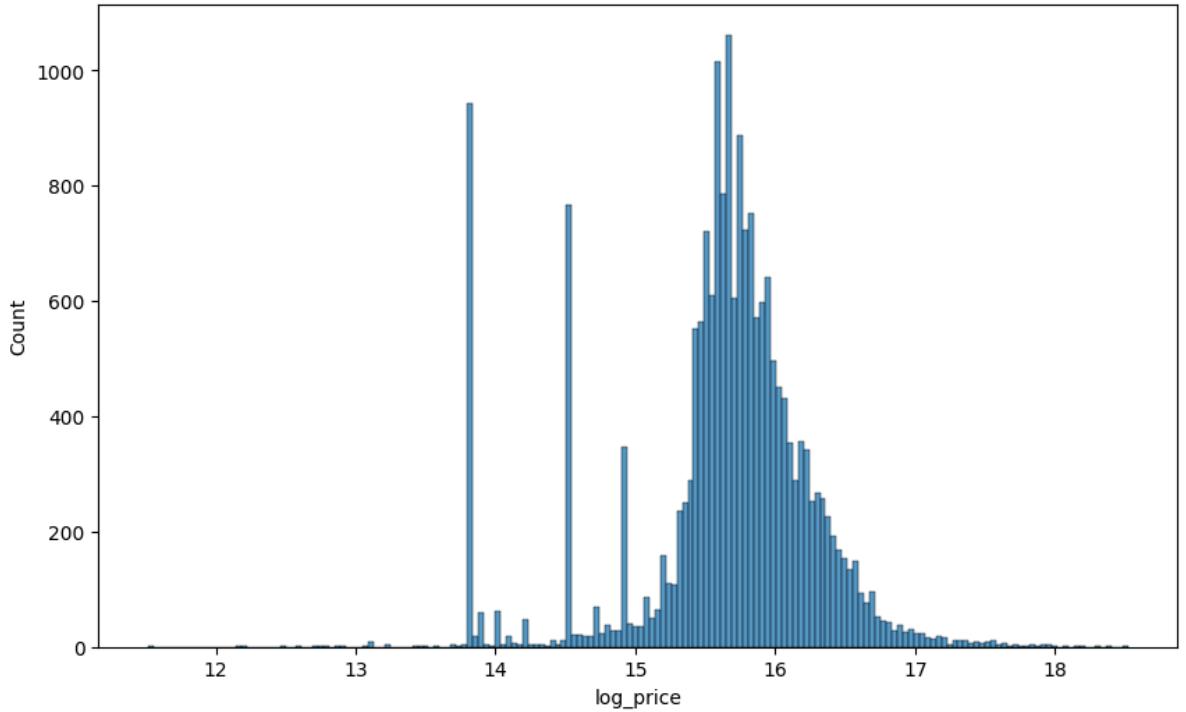
```
Out[ ]: product_type
Investment      6670000.0
OwnerOccupier   5564090.0
Name: price_doc, dtype: float64
```

```
In [ ]: # Create a new column 'log_price'
dtrain['log_price'] = np.log(dtrain['price_doc'])

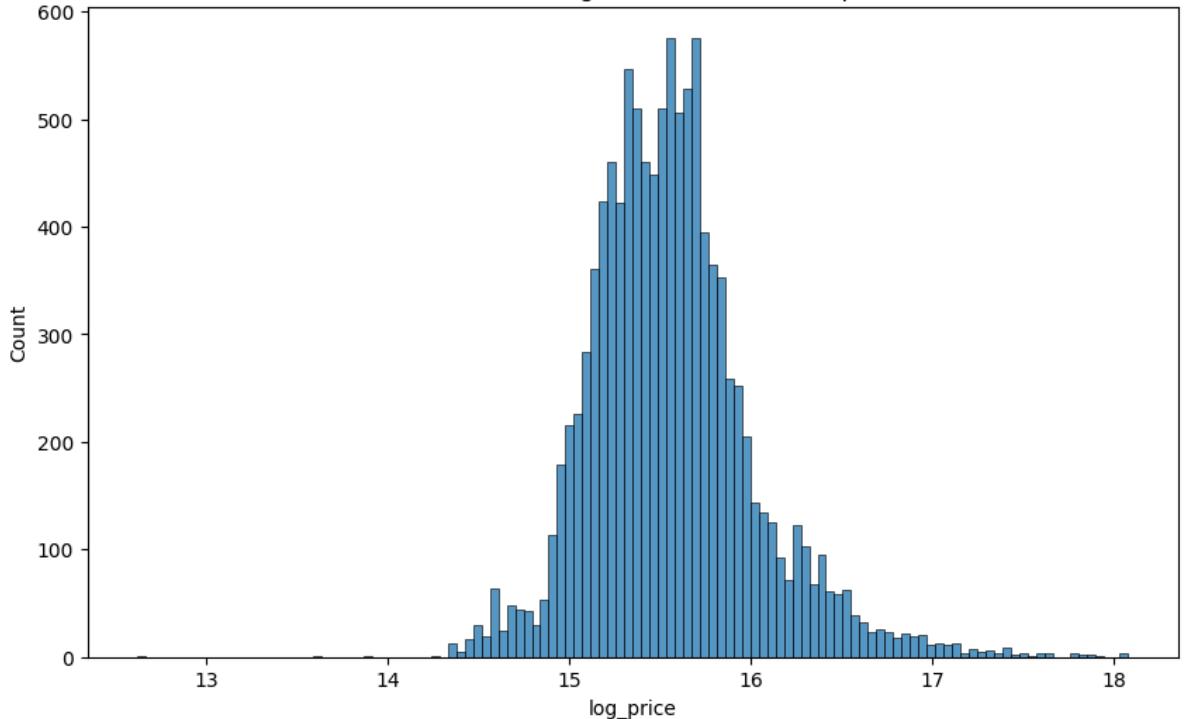
# Get unique product types
product_types = dtrain['product_type'].unique()

# Create a separate histogram for each product type
for product_type in product_types:
    plt.figure(figsize=(10,6))
    sns.histplot(data=dtrain[dtrain['product_type'] == product_type], x='log_price')
    plt.title(f'Distribution of Log Price for {product_type}')
    plt.show()
```

Distribution of Log Price for Investment

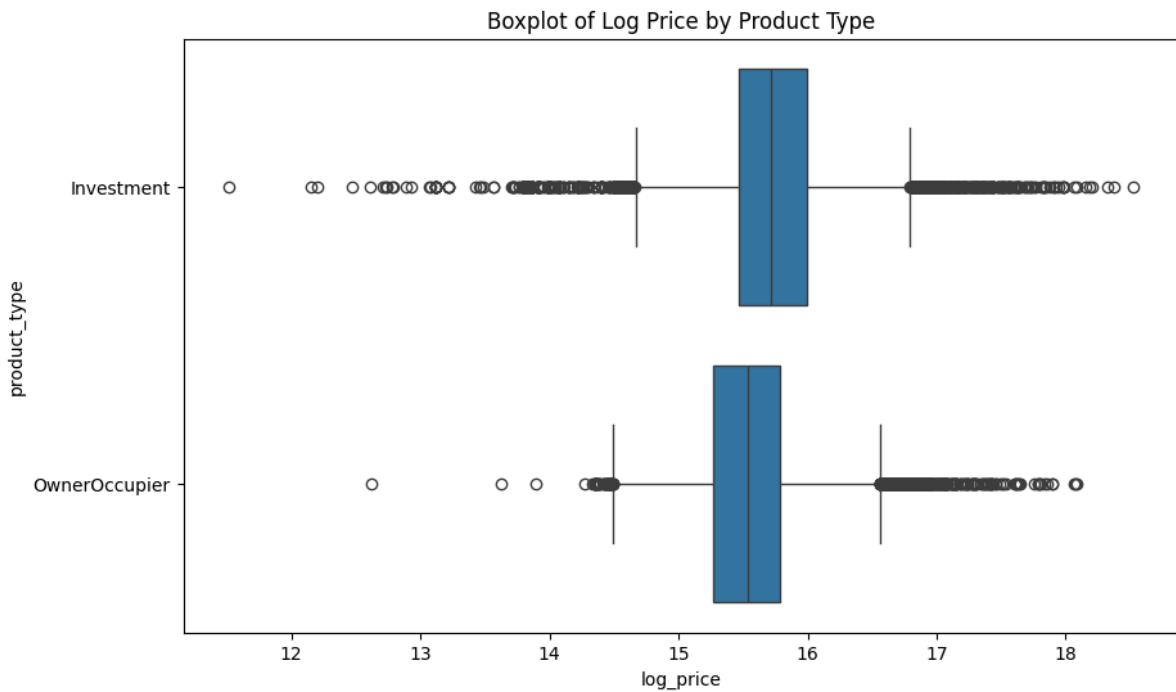


Distribution of Log Price for OwnerOccupier

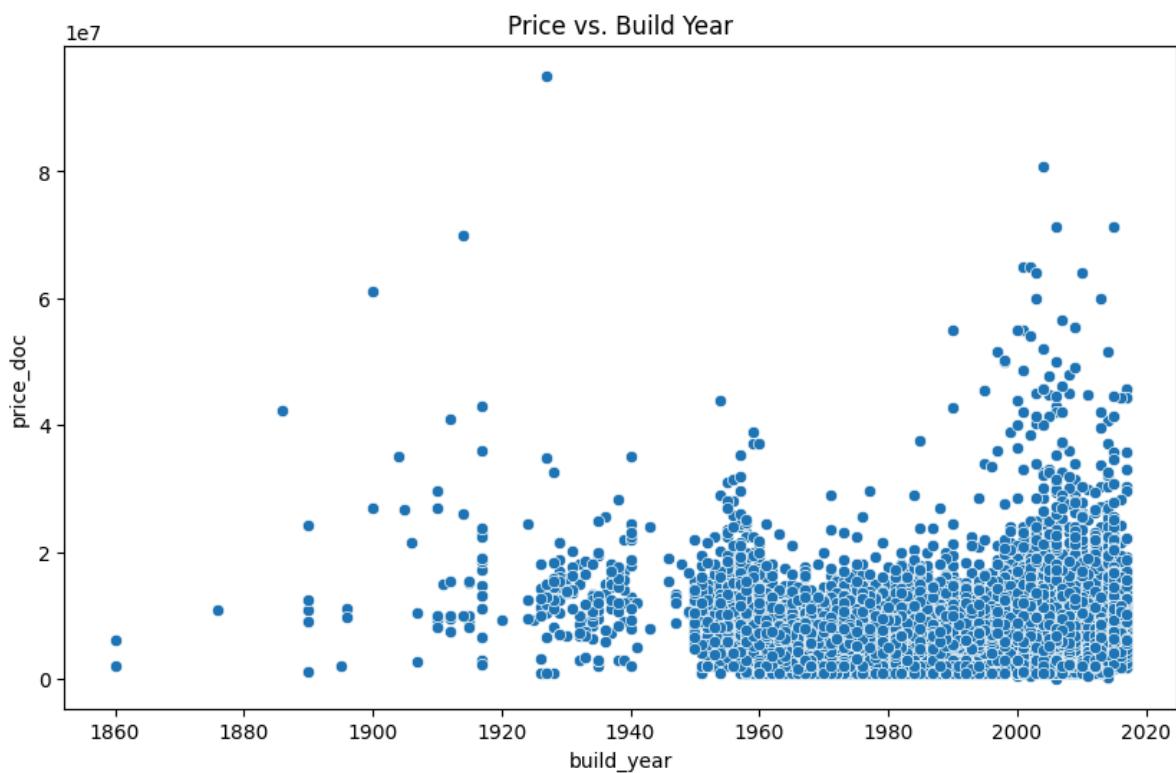


```
In [ ]: plt.figure(figsize=(10,6))
sns.boxplot(data=dtrain, x='log_price', y='product_type')
plt.title('Boxplot of Log Price by Product Type')
plt.show()

dtrain = dtrain.drop('log_price', axis=1)
```



```
In [ ]: #build year against price_doc
#without the problematic years
plt.figure(figsize=(10, 6))
sns.scatterplot(x='build_year', y='price_doc', data=dtrain[(dtrain['build_year'] >
plt.title('Price vs. Build Year')
plt.show()
```



```
In [ ]: # List of problematic years
invalid_years = [0, 1, 3, 20, 71, 215, 2018, 4965]

# נסמן ינאיות
for year in invalid_years:
    dtrain.loc[dtrain['build_year'] == year, 'build_year'] = np.nan
```

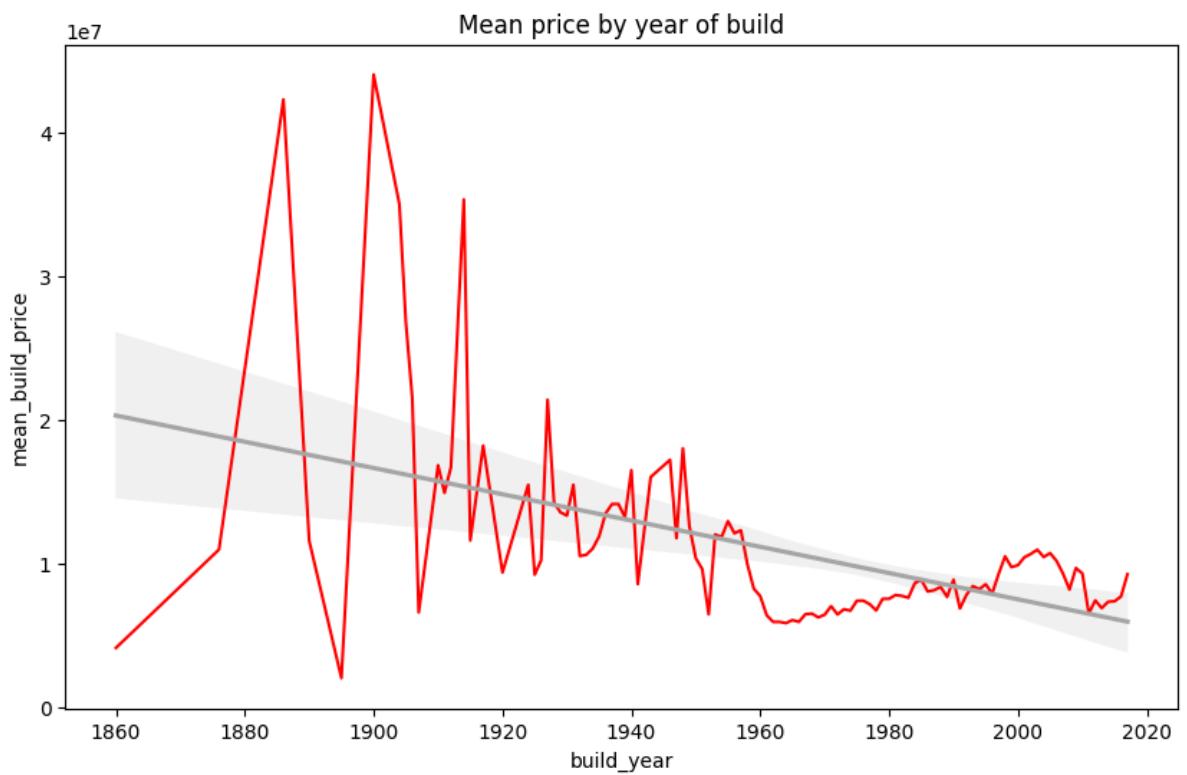
```
In [ ]: # Filter rows where build_year is between 1691 and 2018
filtered_dtrain = dtrain[(dtrain['build_year'] > 1691) & (dtrain['build_year'] < 2018)]
```

```

# Group by build_year and calculate mean price
grouped_dtrain = filtered_dtrain.groupby('build_year')['price_doc'].mean().reset_index()
grouped_dtrain.columns = ['build_year', 'mean_build_price']

# Plot mean price by year of build
plt.figure(figsize=(10, 6))
sns.lineplot(x='build_year', y='mean_build_price', data=grouped_dtrain, color='red')
sns.regplot(x='build_year', y='mean_build_price', data=grouped_dtrain, scatter=False)
plt.title('Mean price by year of build')
plt.show()

```



ניסנו לראות את הממוצע של המכירות לפי שנת בניה אך לא הצלחנו להסיק יותר מידי מהגרף לא נראה ממשיכים האלה!

Home State/Material

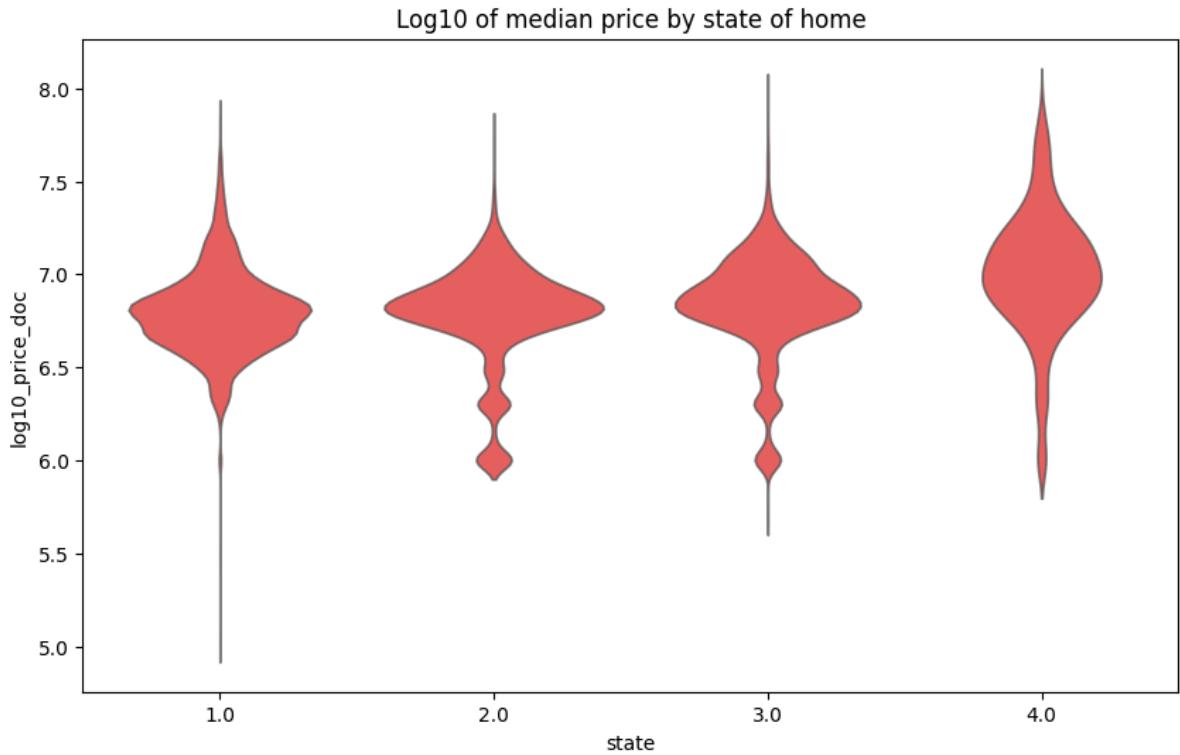
```

In [ ]: # Filter rows where state is not NaN
filtered_dtrain = dtrain[~dtrain['state'].isna()].copy()

# Create a new column for Log10 of price_doc
filtered_dtrain['log10_price_doc'] = np.log10(filtered_dtrain['price_doc'])

# Plot Log10 of price_doc by state
plt.figure(figsize=(10, 6))
sns.violinplot(x='state', y='log10_price_doc', data=filtered_dtrain, inner=None, color='blue')
plt.title('Log10 of median price by state of home')
plt.show()

```



השימוש בלוג היה בשכיל לראות את הגרף יותר טוב לאפשר לראות שאין הבדלים בין האזוריים חוץ מאזור 4 שהמחיר ממוצע גבוי מהשאר

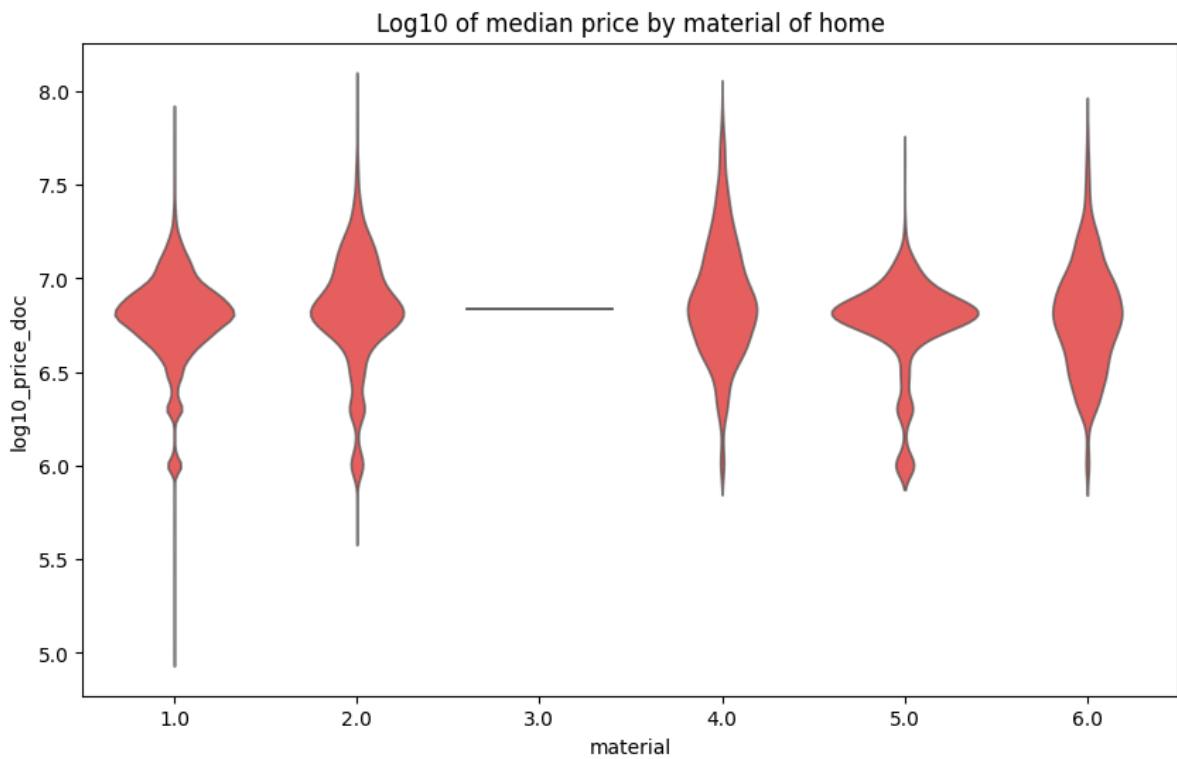
```
In [ ]: # Look at the difference
dtrain.groupby('state')['price_doc'].median()
```

```
Out[ ]: state
1.0    6156040.0
2.0    6650000.0
3.0    7250000.0
4.0    10050000.0
Name: price_doc, dtype: float64
```

```
# Filter rows where state is not NaN
filtered_dtrain = dtrain[~dtrain['material'].isna()].copy()

# Create a new column for Log10 of price_doc
filtered_dtrain['log10_price_doc'] = np.log10(filtered_dtrain['price_doc'])

# Plot Log10 of price_doc by state
plt.figure(figsize=(10, 6))
sns.violinplot(x='material', y='log10_price_doc', data=filtered_dtrain, inner=None)
plt.title('Log10 of median price by material of home')
plt.show()
```



עשינו את אותו דבר גם לחרטורים, אפשר לראות שבחומר השלישי אין יותר מידי תוצאות אך למרות זאת החלטנו לא לשער אותו לשום דבר אחר

```
In [ ]: #לראות בין את ההבדל
dtrain.groupby('material')['price_doc'].median()
```

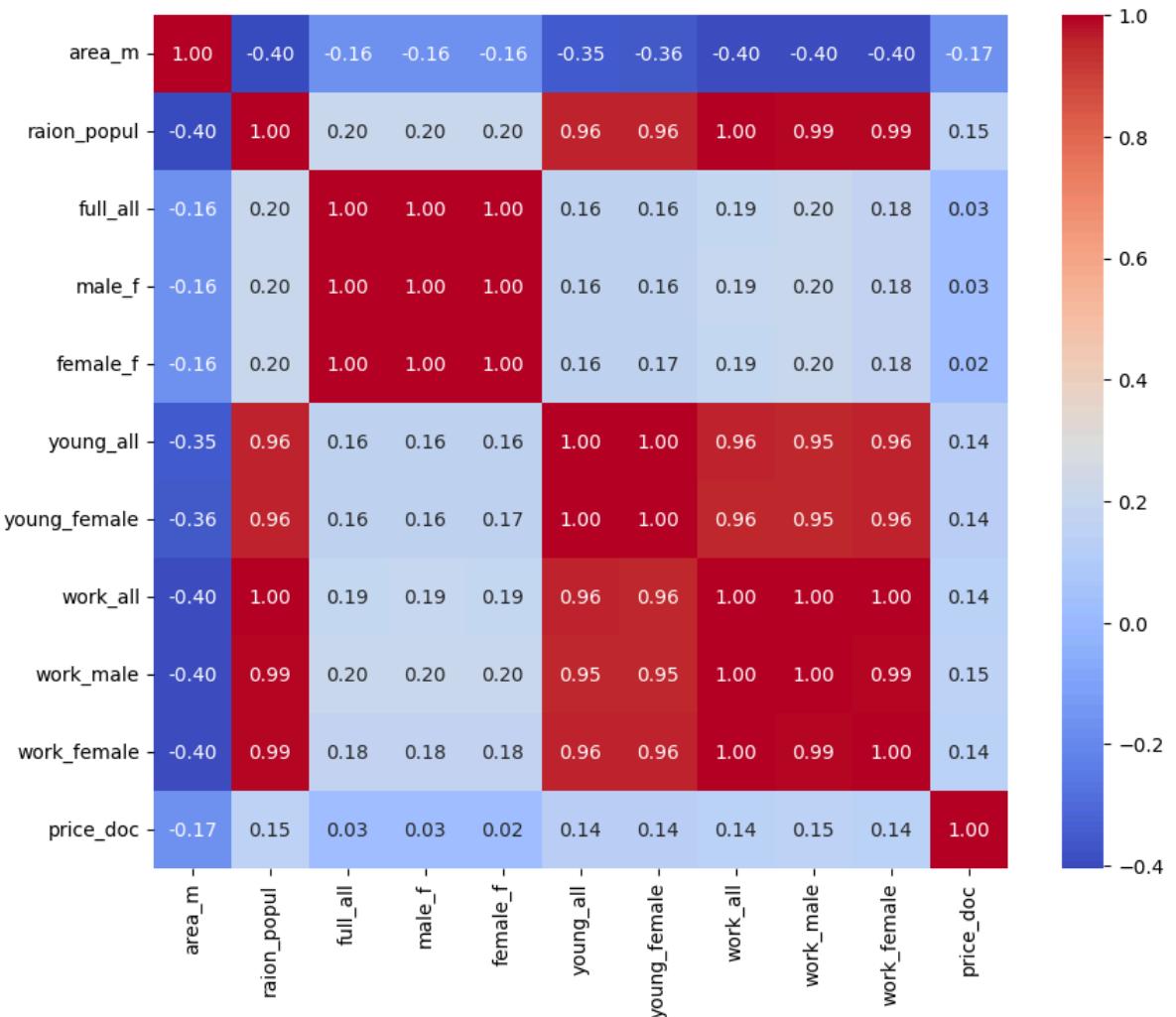
```
Out[ ]:
material
1.0    6500000.0
2.0    6900000.0
3.0    6931143.0
4.0    7247869.5
5.0    6492000.0
6.0    6362318.0
Name: price_doc, dtype: float64
```

Demographic Characteristics

```
In [ ]: #בדיקה קרווליזה לפיצרים דמוגרפיים
demo_vars = ['area_m', 'raion_popul', 'full_all', 'male_f', 'female_f', 'young_all',
             'young_female', 'work_all', 'work_male', 'work_female', 'price_doc']

# Calculate correlation matrix
corr_matrix = dtrain[demo_vars].corr()

# Plot correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True, square=True)
plt.show()
```

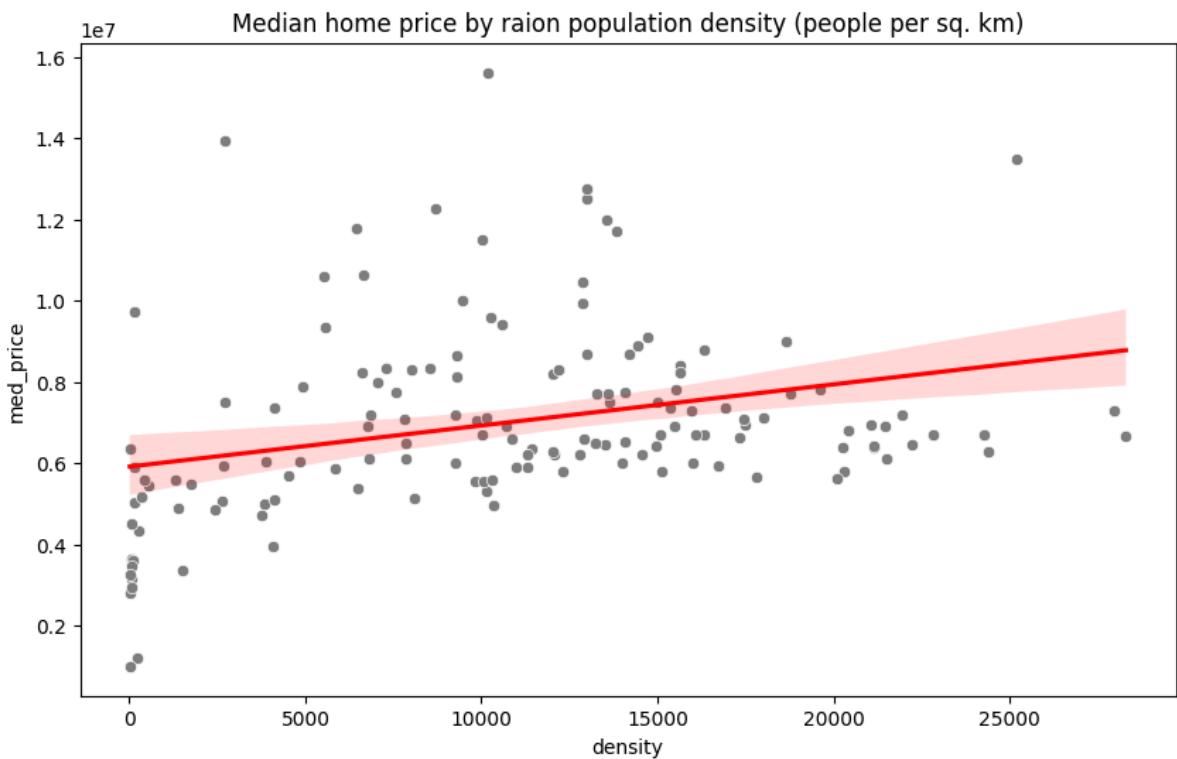


כאן רואים שלפנינו קורלציית הדמוגרפיה אין קורלציה גבוהה עם המחיר, אך יש קורלציה גבוהה בין מספר משתנים לבין עצם דבר אשר נטפל בו בהמשך

```
In [ ]: # חישוב של הצפיפות האוכלוסינית ואת המחיר החציוני לכל אזור, והזנה בגרף של המחיר החציוני לפי הצפיפות #
dtrain['area_km'] = dtrain['area_m'] / 1000000
dtrain['density'] = dtrain['raion_popul'] / dtrain['area_km']

# Group by sub_area and calculate median density and price
grouped_dtrain = dtrain.groupby('sub_area').agg({'density': 'median', 'price_doc': 'median'})
grouped_dtrain.columns = ['sub_area', 'density', 'med_price']

# Plot median price by density
plt.figure(figsize=(10, 6))
sns.scatterplot(x='density', y='med_price', data=grouped_dtrain, color='grey')
sns.regplot(x='density', y='med_price', data=grouped_dtrain, scatter=False, color='red')
plt.title('Median home price by raion population density (people per sq. km)')
plt.show()
```

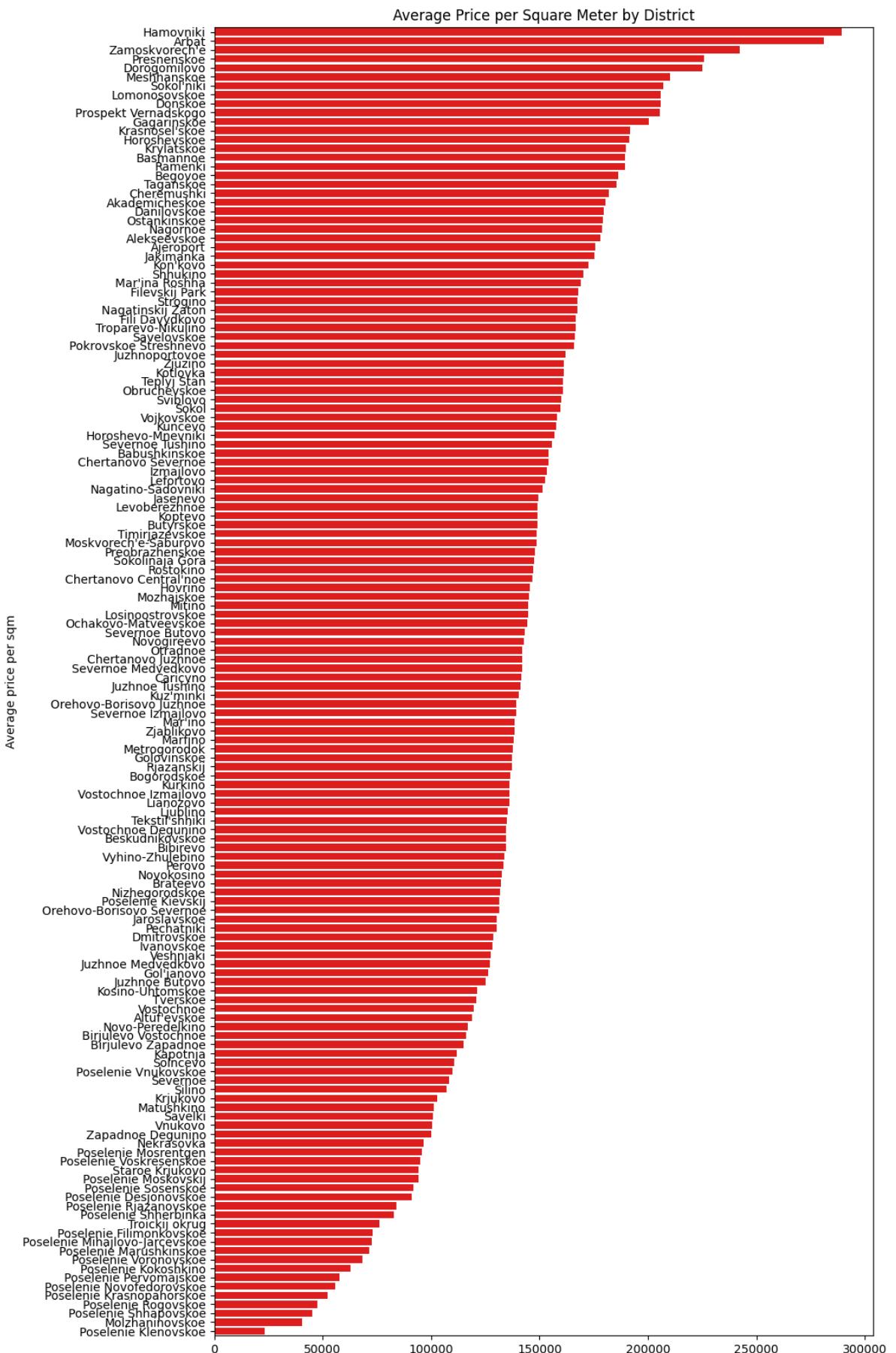


אפשר לראות שכל שהאזורים מואוכלסים יותר כך חציו המהיר עולה, אפשר לחשב על זה שבחזירים מרוחקים ומכובדים המחיר נטוך יותר מאשר מרכזי

```
In [ ]: # בדיקת מחיר ממוצע למטר רבוע לפי אזור
grouped_dtrain_avg_price_per_sqm = dtrain.groupby('sub_area').agg({'price_doc': 'sum',
# Calculate the average price per square meter for each sub_area
grouped_dtrain_avg_price_per_sqm['avg_price_per_sqm'] = grouped_dtrain_avg_price_per_sqm['price_doc'] / grouped_dtrain_avg_price_per_sqm['sqm']

# Display the result
grouped_dtrain_avg_price_per_sqm[['sub_area', 'avg_price_per_sqm']]

# plot
grouped_dtrain_avg_price_per_sqm = grouped_dtrain_avg_price_per_sqm.sort_values('avg_price_per_sqm')
plt.figure(figsize=(10, 20))
sns.barplot(x='avg_price_per_sqm', y='sub_area', data=grouped_dtrain_avg_price_per_sqm)
plt.xlabel('')
plt.ylabel('Average price per sqm')
plt.title('Average Price per Square Meter by District')
plt.show()
```



```
In [ ]: dtrain = dtrain.drop(['young_female', 'work_female', 'work_male', 'male_f', 'female_f'])
```

School Characteristics

```
In [ ]: # בדיקת קורלציה לפי מאפיין חינוך
school_chars = ['children_preschool', 'preschool_quota', 'preschool_education_center']
```

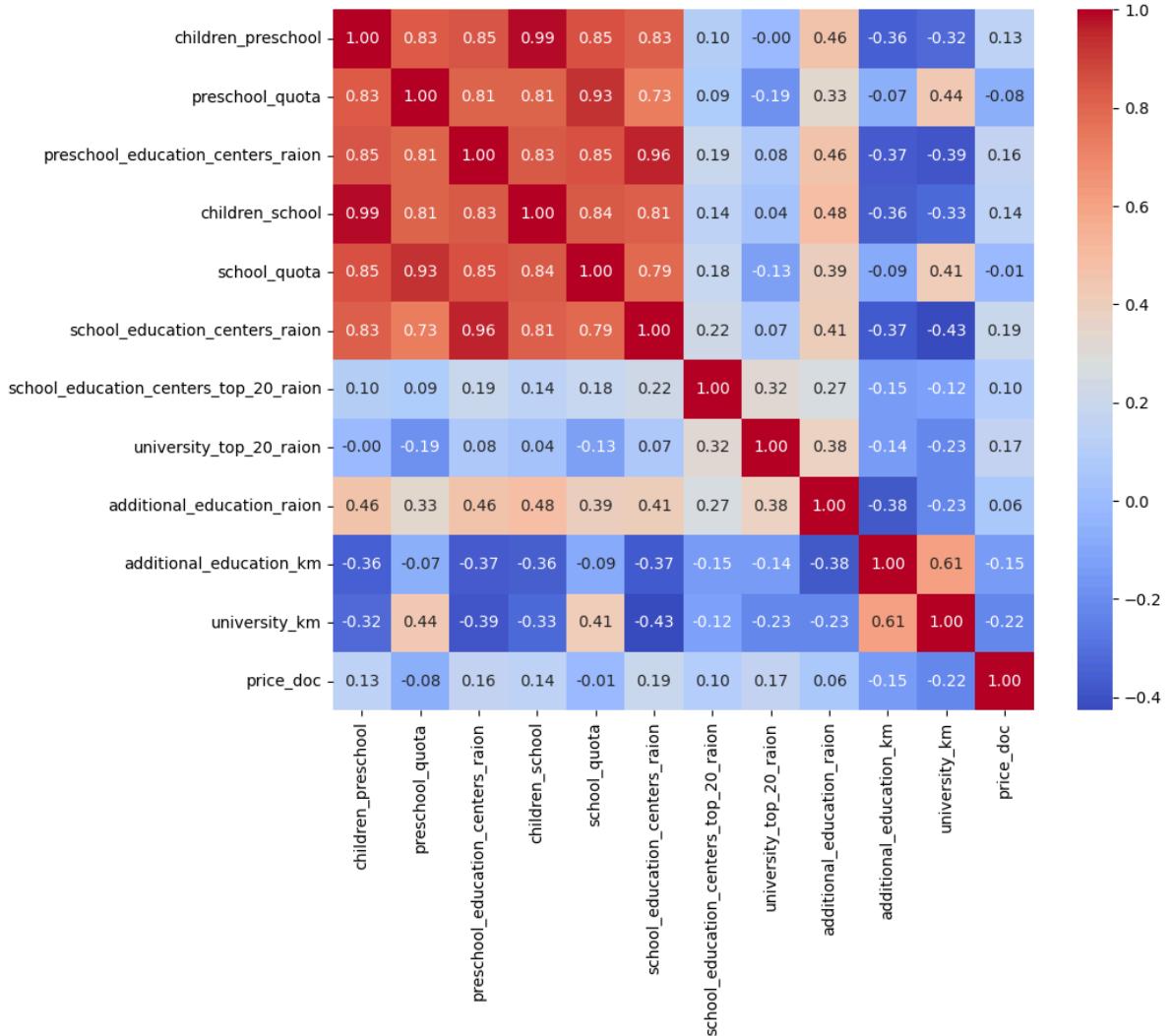
```

'children_school', 'school_quota', 'school_education_centers_raion',
'school_education_centers_top_20_raion', 'university_top_20_raion',
'additional_education_raion', 'additional_education_km', 'universit
'price_doc']

# Calculate correlation matrix
corr_matrix = dtrain[school_chars].corr()

# Plot correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True, square=True)
plt.show()

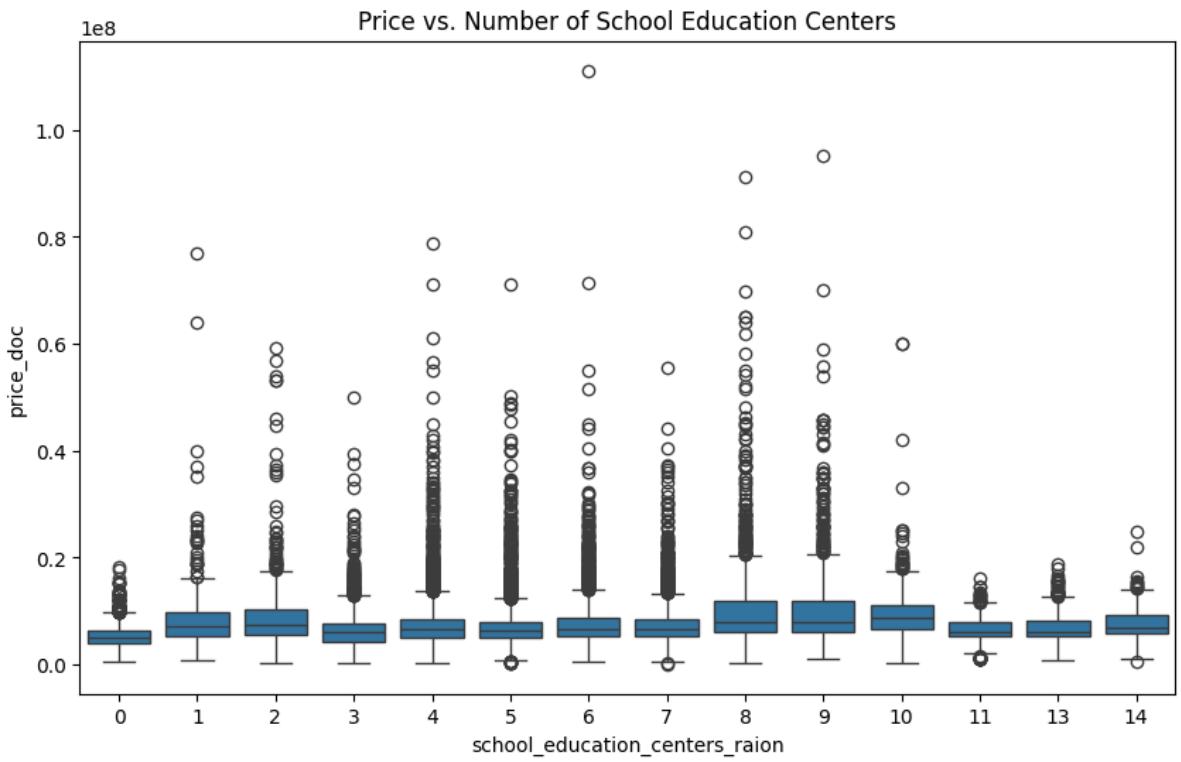
```



In []: `dtrain = dtrain.drop(['preschool_education_centers_raion', 'children_preschool'], axis=1)`

אפשר לראות שאין קורלציה גבוהה בין המחיר לבין הפיצרים של החינוך, אך עדיין יש קורלציות גבוהות בין פיצרים לעצם אשר נתיחס אליהם אחרי כן.

In []: `# school_education_centers against price_doc
plt.figure(figsize=(10, 6))
sns.boxplot(x='school_education_centers_raion', y='price_doc', data=dtrain)
plt.title('Price vs. Number of School Education Centers')
plt.show()`

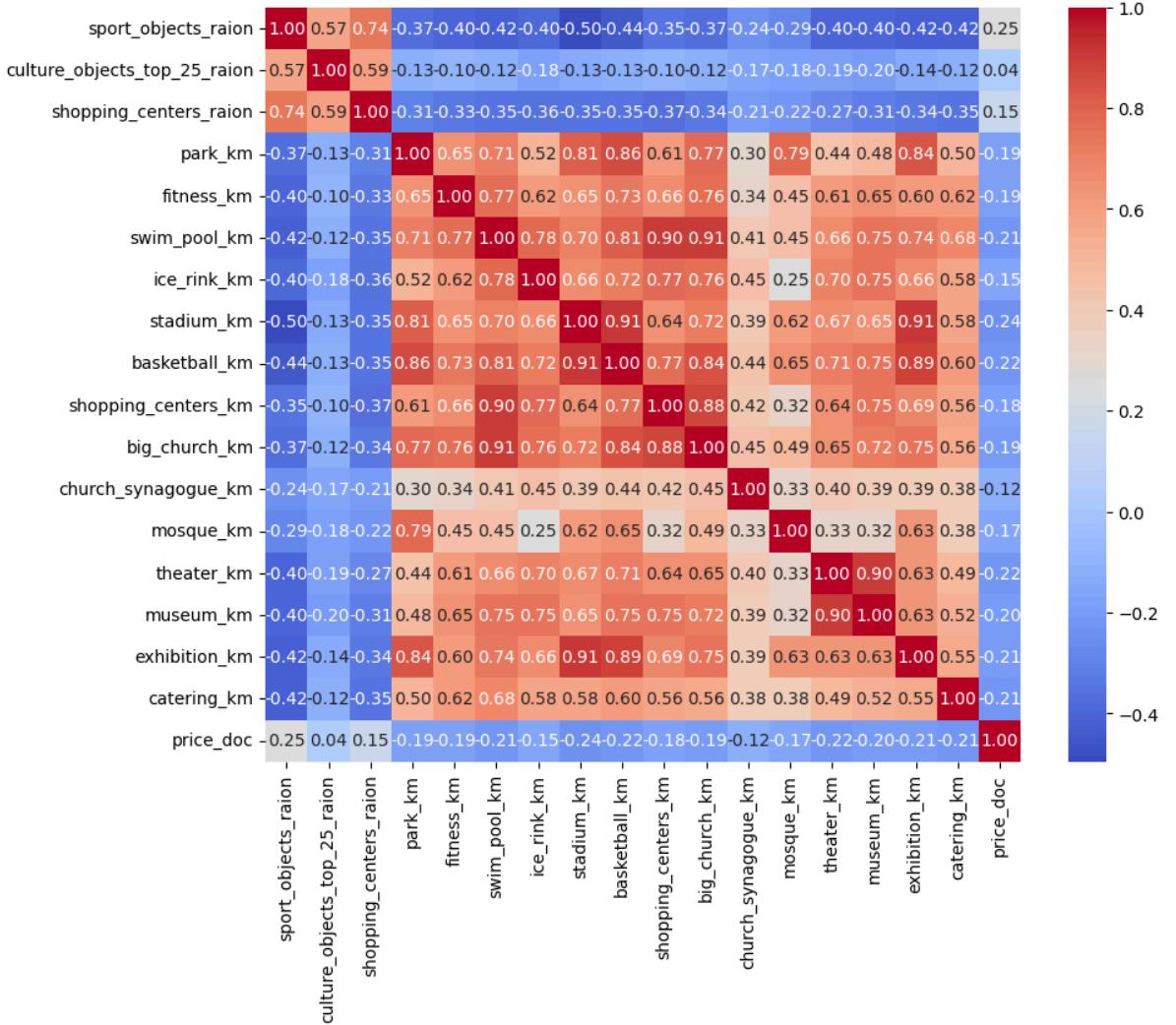


Cultural/Recreational Characteristics

```
In [ ]: # בדיקת קורלציה בין הפייצרים של תרבות והספורט#
cult_chars = ['sport_objects_raion', 'culture_objects_top_25_raion', 'shopping_center_km', 'fitness_km', 'swim_pool_km', 'ice_rink_km', 'stadium_km', 'shopping_centers_km', 'big_church_km', 'church_synagogue_km', 'mosque_museum_km', 'exhibition_km', 'catering_km', 'price_doc']

# Calculate correlation matrix
corr_matrix = dtrain[cult_chars].corr()

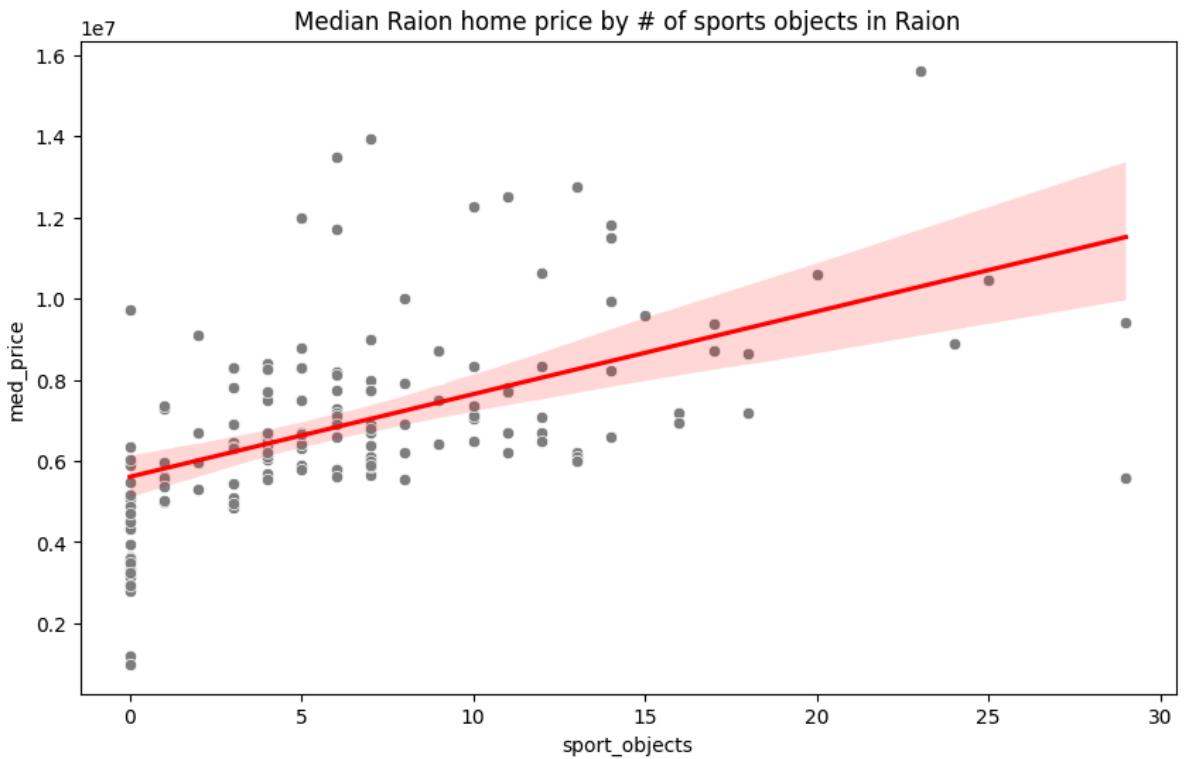
# Plot correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True, square=True)
plt.show()
```



ראינו שיש קורלציה אחת יחסית גבוהה בין מתקני הספורט לבין המחיר אז החלטנו לבדוק זאת, בנוסף אפשר שיש קשר שלילי בין מרחק של הבית לדברים תרבותיים.

```
In [ ]: # היישוב הממוצע לפי מספר המתקנים הספורטיביים ואת המחיר החזינו לכלאזור
grouped_dtrain = dtrain.groupby('sub_area').agg({'sport_objects_raion': 'mean', 'pr
grouped_dtrain.columns = ['sub_area', 'sport_objects', 'med_price']

# Plot median price by sport_objects
plt.figure(figsize=(10, 6))
sns.scatterplot(x='sport_objects', y='med_price', data=grouped_dtrain, color='grey')
sns.regplot(x='sport_objects', y='med_price', data=grouped_dtrain, scatter=False, c
plt.title('Median Raion home price by # of sports objects in Raion')
plt.show()
```



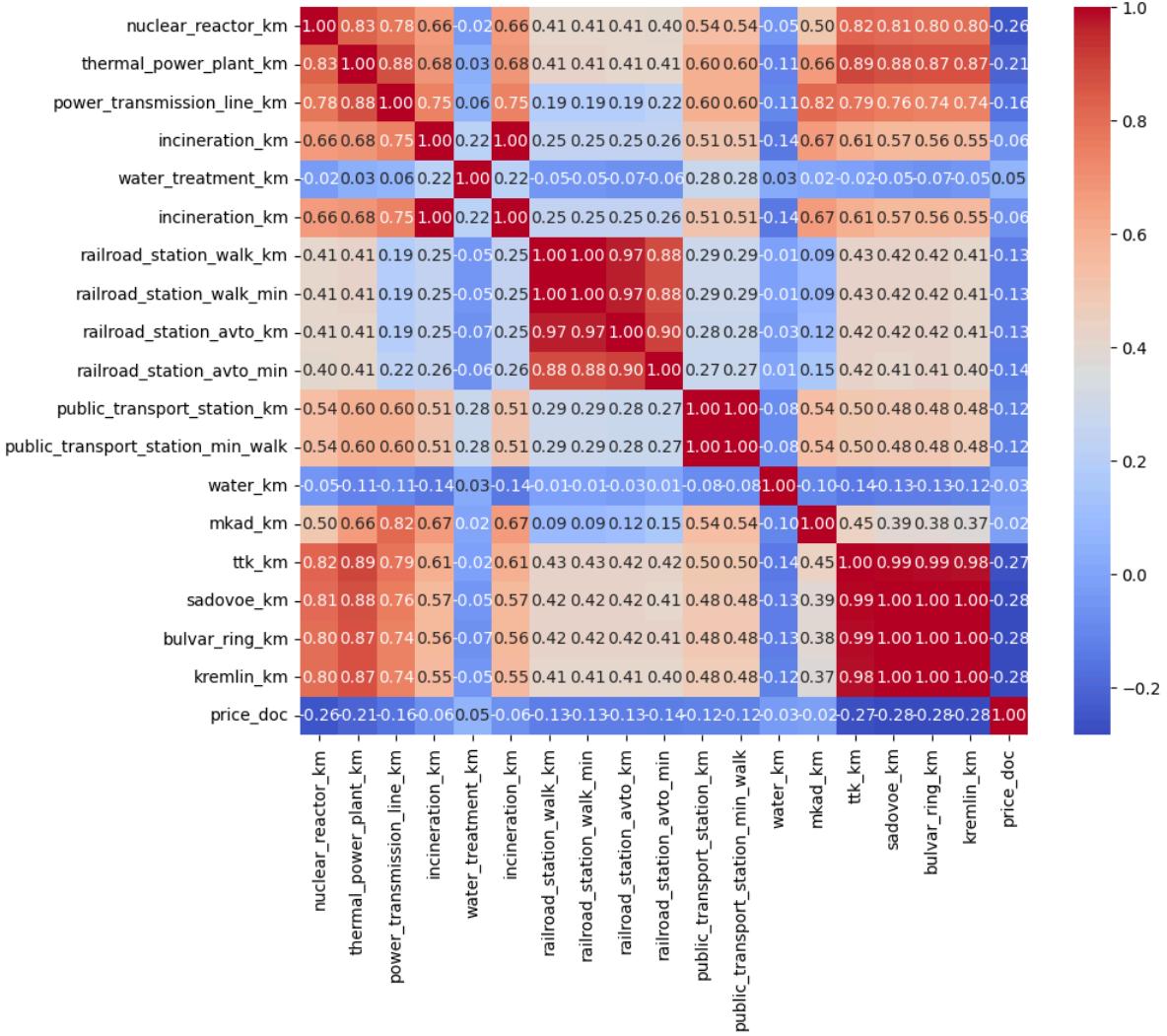
אפשר לראות קשר ביןvariety של אובייקטים ספורטיביים לבין מחירי הדיור. הדבר יכול לשמש אותנו בעתיד ולהבין שזה משתנה חשוב

Infrastructure Features

```
In [ ]: # בדיקת קרולציה לפיזרים מפעליים
inf_features = ['nuclear_reactor_km', 'thermal_power_plant_km', 'power_transmission_km',
'incineration_km', 'water_treatment_km', 'incineration_km', 'railroad_km',
'railroad_station_walk_min', 'railroad_station_avto_km', 'railroad_km',
'public_transport_station_km', 'public_transport_station_min_walk',
'mkad_km', 'ttk_km', 'sadovoe_km', 'bulvar_ring_km', 'kremlin_km', '']

# Calculate correlation matrix
corr_matrix = dtrain[inf_features].corr()

# Plot correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True, square=True)
plt.show()
```



```
In [ ]: dtrain = dtrain.drop(['ttk_km', 'sadovoe_km', 'bulvar_ring_km'], axis=1, errors='ignore')
```

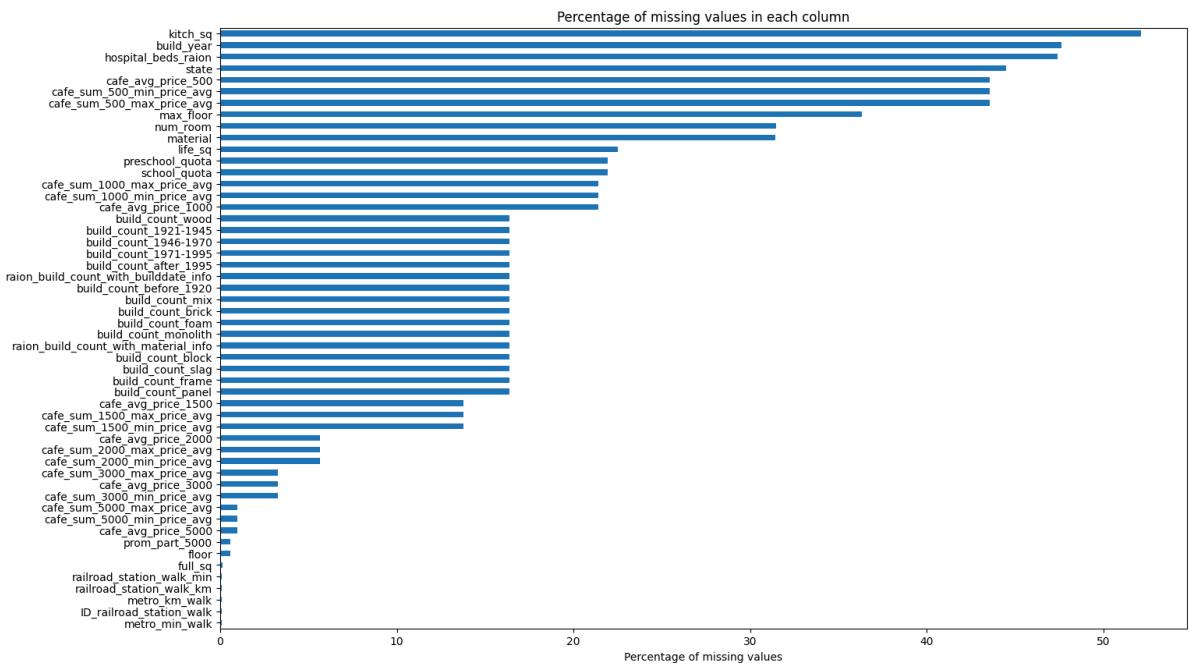
כאן אפשר לראות רק קורלציות חלשות בין המחיר לבין הפיצרים, אך אפשר לראות גם קורלציות מושלמות בין הפיצרים עצמם. כאן אפשר לראות רק קורלציות חלשות בין המחיר לבין הפיצרים, אך אפשר לראות גם קורלציות מושלמות בין הפיצרים עצמם.

עד כאן עשינו את החלק של חקירת הדטה עצם בחלק זה חישבנו קורלציות בין המחיר לבין קטגוריות מסוימות של פיצרים, עשינו זאת כדי להתמודד בהמשך הדטה וגם ללחוץ תוכנות, דרכו פועלה עדידות ואפיו קצת "סידרנו" את הדטה. ועכשו אחריו כל זה אנחנו מוכנים להתחילה ולבסוף ממש על דאטנה לנוקות אותה בסדר אותה ולהבין מה המשתנים החשובים שיתנו לנו את המודל האופטימלי. יוצאים לדרכ!

```
In [ ]: #נתקל במקרה שהפרמטרים בפיז'רים בלטת כדי לראות האם יש קורלציות גבות בין המשתנים או בין המשתנים לשני המטריה
#ולבדוק האם יש מה לעובוד איתם או אפשר למחוק אותם
missing_values_percent = dtrain.isnull().mean() * 100

missing_values_percent_filtered = missing_values_percent[missing_values_percent > 60]

plt.figure(figsize=(16, 10))
missing_values_percent_filtered.sort_values().plot(kind='barh')
plt.xlabel('Percentage of missing values')
plt.title('Percentage of missing values in each column')
plt.show()
```

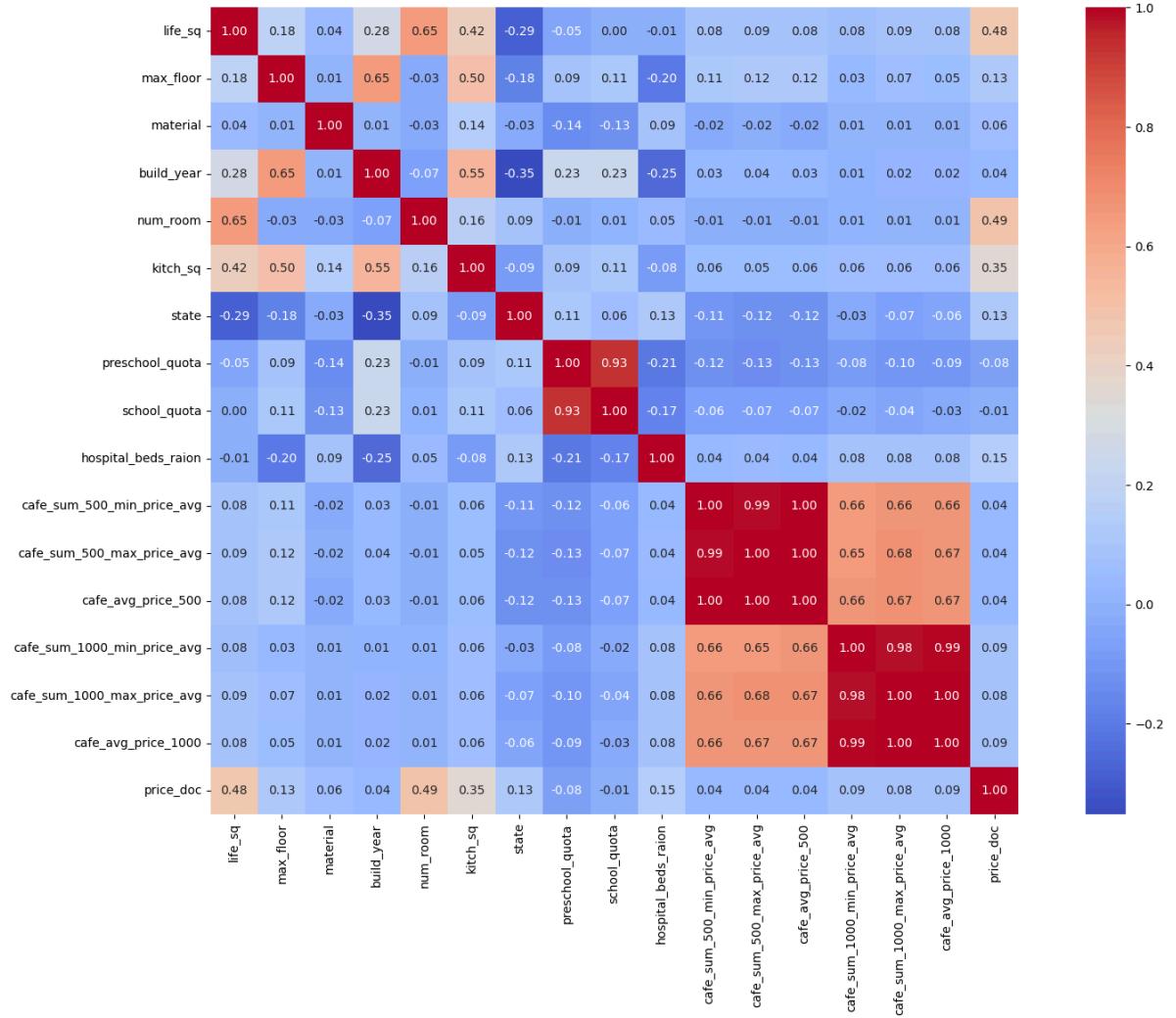


```
In [ ]: #הערכים החסרים בפיז'רים כדי לבדוק אם יש קורלציות גבוהות בין המשנים או בין המשנים למשני המטריך
#ולבדוק האם יש מה לעובד אותם או אפשר למחוק אותם
high_missing_features= missing_values_percent[(missing_values_percent >= 20)].index

features = list(high_missing_features) + ['price_doc']

corr_matrix = dtrain[features].corr()

plt.figure(figsize=(20, 12))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True, square=True)
plt.show()
```



```
In [ ]: #הורדת הפי'רים בעלי קוולוציה גבוהה אחד לשני
dtrain = dtrain.drop(['cafe_sum_500_min_price_avg', 'cafe_sum_500_max_price_avg', 'cafe_avg_price_500', 'cafe_sum_1000_min_price_avg', 'cafe_sum_1000_max_price_avg', 'cafe_avg_price_1000'], axis=1)
```



```
In [ ]: #החלפת משתנים קטגוריאליים באורדריללים (יעשה 0 או 1 במשתנים של כן או לא ואחרים לפי סדר ממשי)
label_enc = LabelEncoder()
for col in categorical_columns:
    if col != 'timestamp' and col != 'sub_area':
        dtrain[col] = label_enc.fit_transform(dtrain[col].astype(str))
```



```
In [ ]: #זה שמייצג את השכונות מכיוון שהוא אינו מראה על סדר מסוים. החלפנו את הערכים בפי'ר זה במשמעותו של אותו איזור#
# Create a target encoder with smoothing
target_enc = ce.TargetEncoder(cols=['sub_area'], smoothing=10.0)

# Fit and transform the 'sub_area' column
dtrain['sub_area'] = target_enc.fit_transform(dtrain['sub_area'], dtrain['price_doc'])

dtrain[categorical_columns]
```

Out[]:

	timestamp	product_type	sub_area	culture_objects_top_25	thermal_power_plant_raion
0	2011-08-20	0	6.670177e+06	0	0
1	2011-08-23	0	8.392539e+06	1	0
2	2011-08-27	0	6.605066e+06	0	0
3	2011-09-01	0	8.809623e+06	0	0
4	2011-09-05	0	1.158589e+07	0	0
...
30466	2015-06-30	0	7.127992e+06	0	0
30467	2015-06-30	0	7.109681e+06	1	0
30468	2015-06-30	1	6.215746e+06	0	0
30469	2015-06-30	0	1.385088e+07	0	0
30470	2015-06-30	0	6.168624e+06	0	0

30471 rows × 16 columns

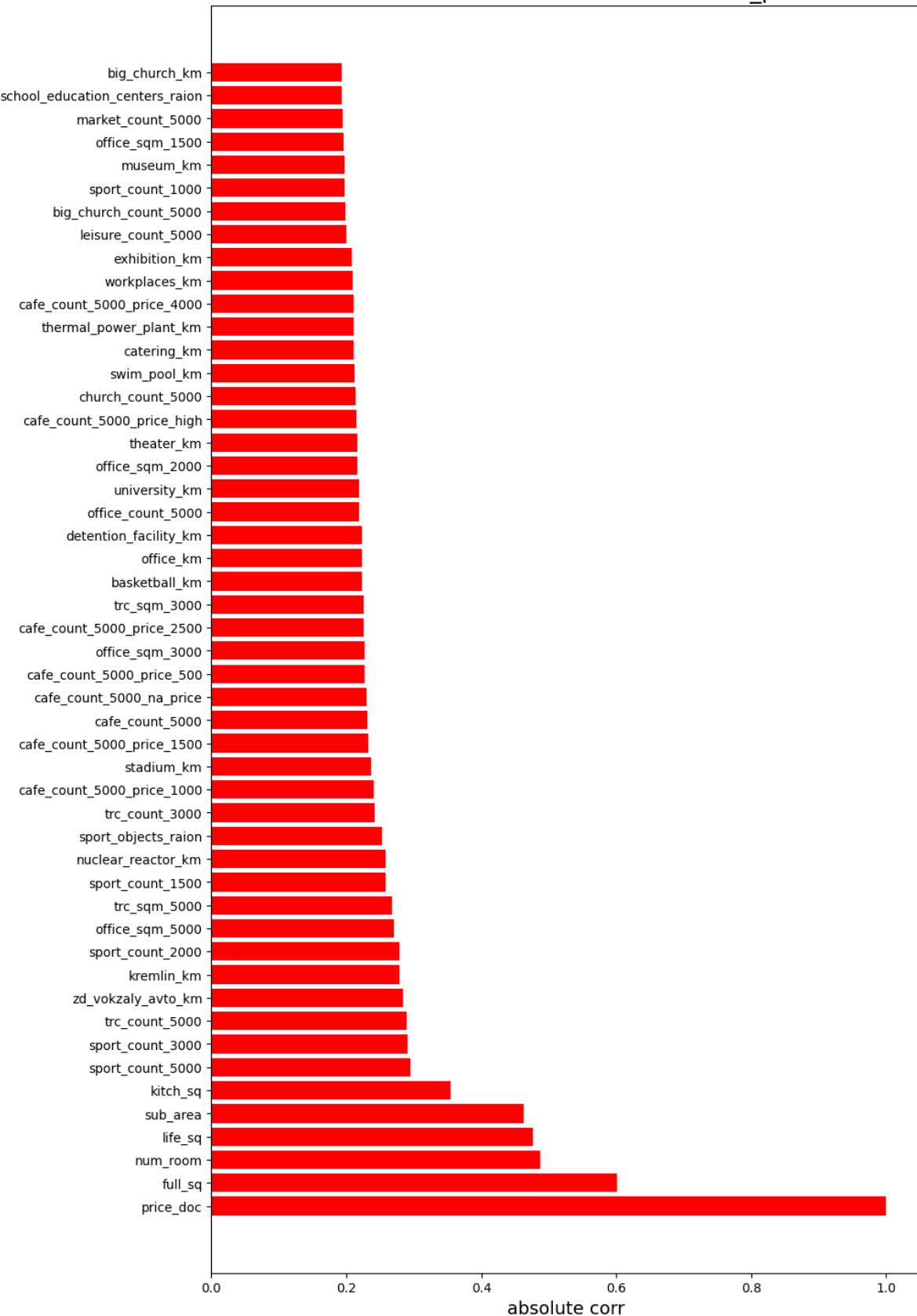
הקורסואטיה כי גובהה עם משתנה המטרה כדי לבדוק אחר כך האם משתנים אלה יכולים לעזור להויר את סיבוכיות המודל#

```

In [ ]: corrmat = dtrain.drop(["id", "timestamp"], axis=1).corr(method='pearson', min_periods=1)
corrmat = np.abs(corrmat)
remain_num = 50
corr_target = corrmat['price_doc'].reset_index()[:-2]
corr_target.columns = ['feature', 'abs_corr']
corr_target = corr_target.sort_values(by = 'abs_corr', ascending = False)[:remain_num]
ind = np.arange(corr_target.shape[0])
width = 0.9
fig, ax = plt.subplots(figsize=(10,18))
rects = ax.barh(ind, corr_target.abs_corr.values, color='r')
ax.set_yticks(ind)
ax.set_yticklabels(corr_target.feature.values, rotation='horizontal')
ax.set_xlabel("absolute corr", fontsize = 14)
ax.set_title("Correlations between features and doc_price ", fontsize = 18)
plt.show()

```

Correlations between features and doc_price



```
In [ ]: #בדיקה האם מתח פיצרים אלה ישנים נאלה בעלי קוורולוציה גבוהה אחד עם החשינה
corr_target_f = list(corr_target.feature.values)
corr_target_f2 = corr_target_f
high_corr = dtrain[corr_target_f2].corr(method='pearson', min_periods=1000)
high_corr = np.abs(high_corr)*100
f, ax = plt.subplots(figsize=(11, 11))
sns.heatmap(high_corr, cbar=False, annot=True, square=True, fmt='%.0f',
            annot_kws={'size': 8})
plt.title('High-correlation Features')
plt.show()
```

High-correlation Features			
price_doc	-0.60	0.49	0.48
full_sq	0.60	0.49	0.48
num_room	0.49	0.73	0.65
life_sq	0.48	0.65	0.60
sub_area	0.46	0.14	0.17
kitch_sq	0.35	0.54	0.42
sport_count_5000	0.29	0	0.9
sport_count_3000	0.29	0	0.10
trc_count_5000	0.29	0	0.10
zd_vokzaly_avto_km	0.28	0	0.14
kremlin_km	0.28	0	0.11
sport_count_2000	0.21	0	0.9
office_sqm_5000	0.27	0	0.5
trc_sqm_5000	0.27	0	0.8
sport_count_1500	0.26	0	0.9
nuclear_reactor_km	0.26	0	0.6
sport_objects_raion	0.25	0	0.9
trc_count_3000	0.24	0	0.9
cafe_count_5000_price_1000	0.24	0	0.7
cafe_count_5000_price_1500	0.24	0	0.7
cafe_count_5000_price_2000	0.23	0	0.7
cafe_count_5000_na_price	0.23	0	0.7
cafe_count_5000_price_500	0.23	0	0.7
cafe_count_5000_price_2500	0.23	0	0.7
trc_sqm_3000	0.23	0	0.5
basketball_km	0.22	0	0.17
office_km	0.22	0	0.17
detection_facility_km	0.22	0	0.17
office_count_5000	0.22	0	0.17
university_km	0.22	0	0.17
office_sqm_2000	0.22	0	0.17
theater_km	0.22	0	0.17
cafe_count_5000_price_high	0.21	0	0.17
church_count_5000	0.21	0	0.17
swim_pool_km	0.21	0	0.17
catering_km	0.21	0	0.17
thermal_power_plant_km	0.21	0	0.17
cafe_count_5000_price_4000	0.21	0	0.17
workplaces_km	0.21	0	0.17
exhibition_km	0.21	0	0.17
leisure_count_5000	0.20	0	0.17
big_church_count_5000	0.20	0	0.17
sport_count_1000	0.20	0	0.17
museum_km	0.20	0	0.17
office_sqm_1500	0.20	0	0.17
market_count_5000	0.19	0	0.17
school_education_centers_raion	0.19	0	0.17
big_church_km	0.19	0	0.17
price_doc	0.60	0.49	0.48
full_sq	0.60	0.49	0.48
num_room	0.49	0.73	0.65
life_sq	0.48	0.65	0.60
sub_area	0.46	0.14	0.17
kitch_sq	0.35	0.54	0.42
sport_count_5000	0.29	0	0.9
trc_count_5000	0.29	0	0.10
zd_vokzaly_avto_km	0.28	0	0.14
kremlin_km	0.28	0	0.11
sport_count_2000	0.21	0	0.9
office_sqm_5000	0.27	0	0.5
trc_sqm_5000	0.27	0	0.8
sport_count_1500	0.26	0	0.9
nuclear_reactor_km	0.26	0	0.6
sport_objects_raion	0.25	0	0.9
trc_count_3000	0.24	0	0.9
cafe_count_5000_price_1000	0.24	0	0.7
cafe_count_5000_price_1500	0.24	0	0.7
cafe_count_5000_price_2000	0.23	0	0.7
cafe_count_5000_na_price	0.23	0	0.7
cafe_count_5000_price_500	0.23	0	0.7
cafe_count_5000_price_2500	0.23	0	0.7
trc_sqm_3000	0.23	0	0.5
basketball_km	0.22	0	0.17
office_km	0.22	0	0.17
detection_facility_km	0.22	0	0.17
office_count_5000	0.22	0	0.17
university_km	0.22	0	0.17
office_sqm_2000	0.22	0	0.17
theater_km	0.22	0	0.17
cafe_count_5000_price_high	0.21	0	0.17
church_count_5000	0.21	0	0.17
swim_pool_km	0.21	0	0.17
catering_km	0.21	0	0.17
thermal_power_plant_km	0.21	0	0.17
cafe_count_5000_price_4000	0.21	0	0.17
workplaces_km	0.21	0	0.17
exhibition_km	0.21	0	0.17
leisure_count_5000	0.20	0	0.17
big_church_count_5000	0.20	0	0.17
sport_count_1000	0.20	0	0.17
museum_km	0.20	0	0.17
office_sqm_1500	0.20	0	0.17
market_count_5000	0.19	0	0.17
school_education_centers_raion	0.19	0	0.17
big_church_km	0.19	0	0.17

In []: `dtrain = dtrain.drop('id', axis=1)`

כטוע זה אנו מבצעים ריגרסיות מסווג RF כדי לראות את השפעות במצב הראשון לאחר הEDA
 לאחר מכן בוצע בדיקה עם השלמת ערכיהם, בנוסף בוצע גם מידול עם פיצרים חדשים אשר יצרנו, ולבסוף יצרנו שילוב של הפיצרים החשובים למודל בשילוב עם הפיצרים עם הקורלציה הגבוהה ביותר עם המשתנה מטרה בודדנו כל גירסיה עם שינוי קטן בשילוב האם בכלל ככל שינוי יש השפעה והאם כדי להמשיך אותו ולשפר עוד יותר את המודל

In []: `#random forest #במצב הראשון לאחר ה-EDA`

```
# Define the features and target
X = dtrain.drop(['price_doc', 'timestamp'], axis=1)
y = dtrain['price_doc']

# Split the data into train and temporary sets (70% train, 30% temp)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)

# Split the temporary set into validation and test sets (50% validation, 50% test)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Initialize the Random Forest regressor
rf_regressor_withNA = RandomForestRegressor(n_estimators=100, random_state=42)
```

```

# Train the model on the training data
rf_regressor_withNA.fit(X_train, y_train)

# Calculate MSE on validation set using 5-Fold Cross-Validation
cv_scores = cross_val_score(rf_regressor_withNA, X_val, y_val, cv=5, scoring='neg_mean_squared_error')
cv_rmse_scores = np.sqrt(-cv_scores)
print(f"Cross-Validation RMSE scores: {cv_rmse_scores}")
print(f"Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores)}")

```

הן ערכים חסרים ע"י ראנדום פורסט והציגו אימפיטר. עשינו זאת על דאטה פרימ חדש כדי לבדוק האם זה באמת מועליל

```

dtrain_noNA = dtrain.copy()
missing_values_percent = dtrain.isnull().mean() * 100
high_missing_features = missing_values_percent[(missing_values_percent >= 20)].index

imputer_rf = IterativeImputer(estimator=RandomForestRegressor(n_estimators=10), random_state=42)

df_imputed_high = imputer_rf.fit_transform(dtrain_noNA[list(high_missing_features)])
df_imputed_high = pd.DataFrame(df_imputed_high, columns=list(high_missing_features))

low_missing_features = missing_values_percent[(missing_values_percent > 0) & (missing_values_percent < 20)]
imputer_median = SimpleImputer(strategy='median')

df_imputed_low = imputer_median.fit_transform(dtrain_noNA[low_missing_features])
df_imputed_low = pd.DataFrame(df_imputed_low, columns=low_missing_features)

df_imputed = pd.concat([df_imputed_high, df_imputed_low], axis=1)
df_imputed = df_imputed.reindex(columns=dtrain_noNA.columns)

```

In []: #random forest - עם המינימיזציה של NA

```

X1 = df_imputed.drop(['price_doc', 'timestamp'], axis=1)
y1 = dtrain['price_doc']

X1_train, X1_temp, y1_train, y1_temp = train_test_split(X1, y1, test_size=0.3, random_state=42)
X1_val, X1_test, y1_val, y1_test = train_test_split(X1_temp, y1_temp, test_size=0.5, random_state=42)

rf_regressor_noNA = RandomForestRegressor(n_estimators=100, random_state=42)

rf_regressor_noNA.fit(X1_train, y1_train)

cv_scores = cross_val_score(rf_regressor_noNA, X1_val, y1_val, cv=5, scoring='neg_mean_squared_error')
cv_rmse_scores = np.sqrt(-cv_scores)
print(f"Cross-Validation RMSE scores: {cv_rmse_scores}")
print(f"Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores)}")

```

Cross-Validation RMSE scores: [3316554.58769802 2762481.56193022 2638919.57258903
2485264.92193019
3038035.35284563]
Mean Cross-Validation RMSE: 2848251.1993986154

In []: #future engenier-
צירה משתנים חדשים ובודקם עם הרגסיה

```

# convert the timestamp to datetime
dtrain['dt'] = pd.to_datetime(dtrain['timestamp'].copy())
# נקודת ההתחלה של הזמן
reference_date = dtrain['dt'].min()

# הפרש מנקודת ההתחלה
dtrain['day_count'] = (dtrain['dt'] - reference_date).dt.days

# Residential & kitchen area to total area ratio:
dtrain['resident_to_total_ratio'] = dtrain['life_sq']/dtrain['full_sq']
dtrain['kitchen_to_total_ratio'] = dtrain['kitch_sq']/dtrain['full_sq']

```

```

dtrain['avg_room_area'] = dtrain['life_sq']/dtrain['num_room']

dtrain['extra_area'] = dtrain['full_sq'] - dtrain['life_sq']

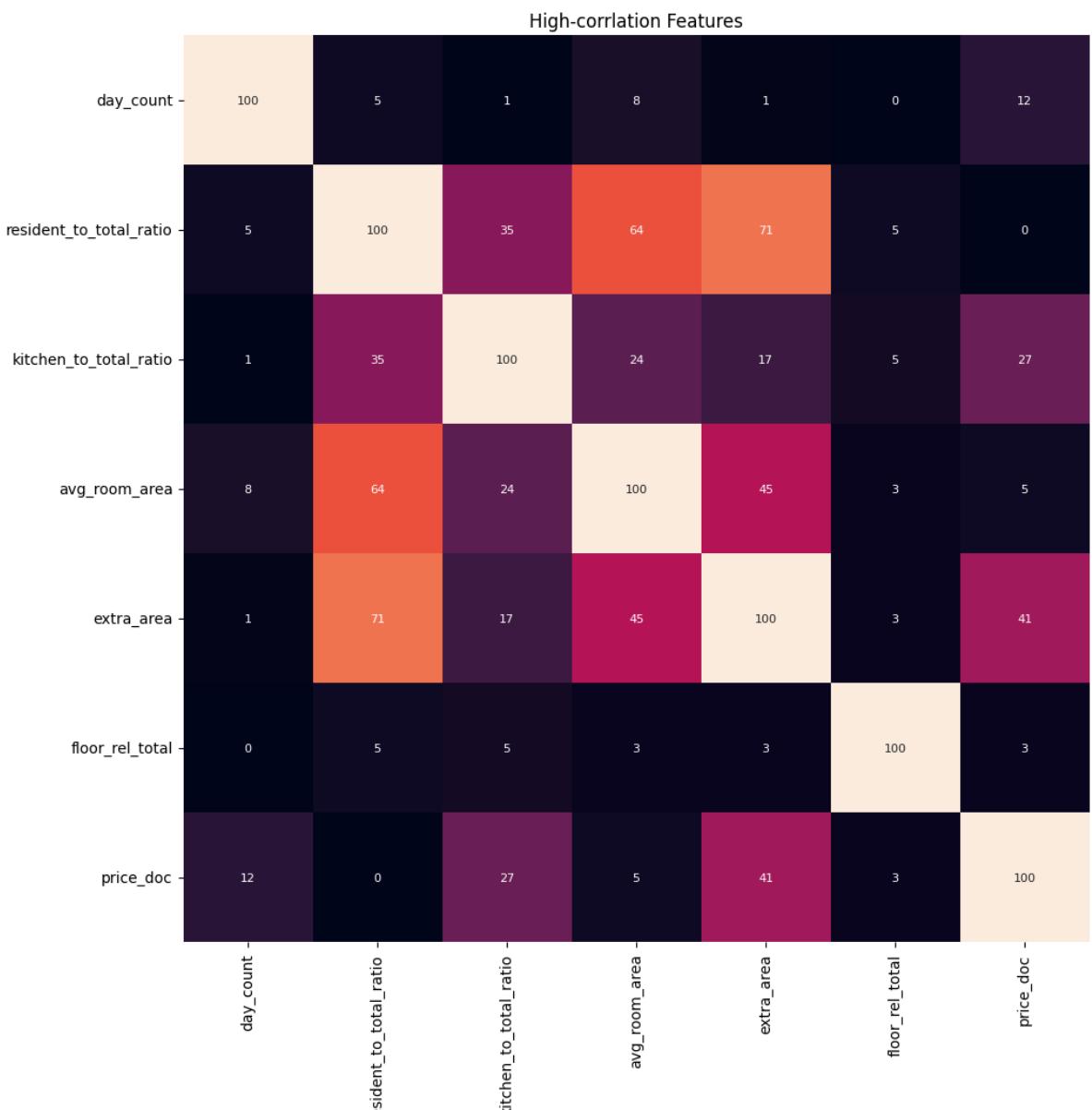
# floor & max_floor to 'floor_rel_total'
dtrain['floor_rel_total'] = dtrain['floor'] / dtrain['max_floor']

# Visualize median house prices over time:

new_features = ['day_count','resident_to_total_ratio','kitchen_to_total_ratio','avg

```

```
In [ ]: new_features.append('price_doc')
new_features_corr = dtrain[new_features].corr(method='pearson', min_periods=1000)
new_features_corr = np.abs(new_features_corr)*100
f, ax = plt.subplots(figsize=(11, 11))
sns.heatmap(new_features_corr, cbar=False, annot=True, square=True, fmt=' .0f',
            annot_kws={'size': 8})
plt.title('High-correlation Features')
plt.show()
```



```
In [ ]: #random forest - בדיקה עם המושנים החדשניים
X2 = dtrain.drop(['price_doc','timestamp','dt'], axis=1)
y2 = dtrain['price_doc']
```

```

X2_train, X2_temp, y2_train, y2_temp = train_test_split(X2, y2, test_size=0.3, random_state=42)

X2_val, X2_test, y2_val, y2_test = train_test_split(X2_temp, y2_temp, test_size=0.5)

rf_regressor_newfeatures = RandomForestRegressor(n_estimators=100, random_state=42)

rf_regressor_newfeatures.fit(X2_train, y2_train)

cv_scores = cross_val_score(rf_regressor_newfeatures, X2_val, y2_val, cv=5, scoring='neg_mean_squared_error')
cv_rmse_scores = np.sqrt(-cv_scores)
print(f"Cross-Validation RMSE scores: {cv_rmse_scores}")
print(f"Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores)}")

```

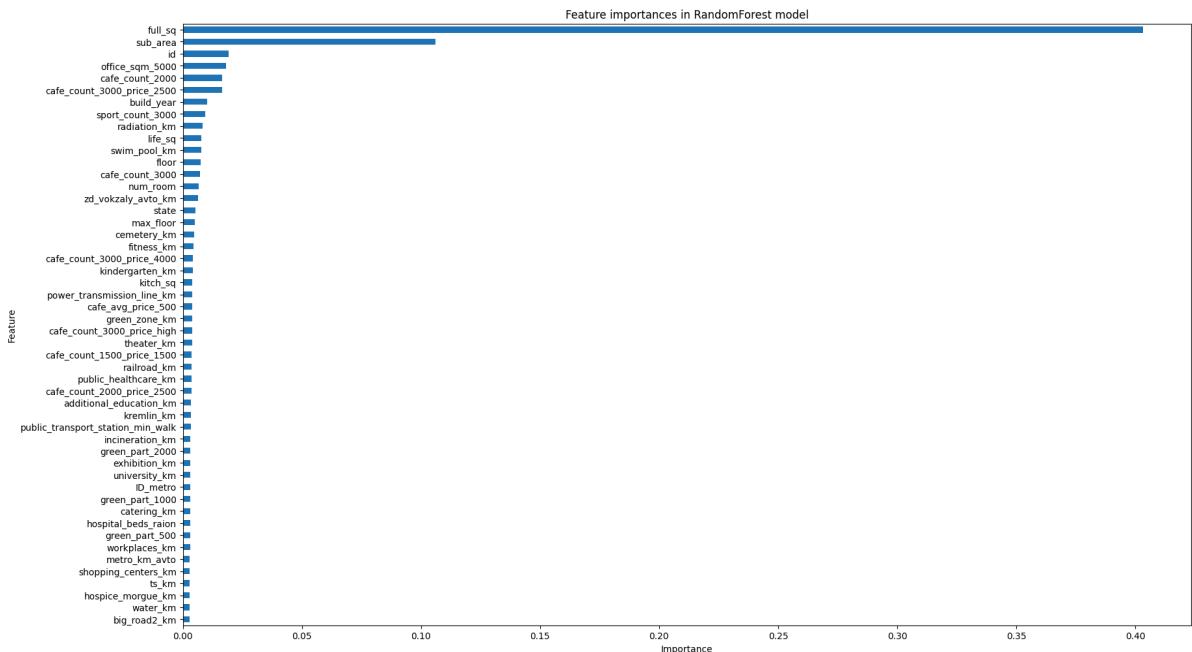
Cross-Validation RMSE scores: [3325385.65392329 2770432.38321692 2620179.35362765
2296385.37064867
3016506.81089989]
Mean Cross-Validation RMSE: 2805777.9144632835

In []: # Feature Importance

```

importances = rf_regressor_withNA.feature_importances_
feature_importances = pd.Series(importances, index=X_train.columns).sort_values(ascending=False)
# צירת הפלט עם המשתנים הכי חשובים
plt.figure(figsize=(20, 12))
feature_importances[:50].sort_values().plot(kind='barh')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature importances in RandomForest model')
plt.show()

```



In []: # שילוב של המשתנים החשובים עם הקרולזיה לרשימה אחת

```

combined_features = corr_target_f + list(feature_importances[:50].index)

combined_features = list(dict.fromkeys(combined_features))
combined_features.remove('price_doc')

```

In []: X4 = dtrain[combined_features]
y4 = dtrain['price_doc']

X4_train, X4_temp, y4_train, y4_temp = train_test_split(X4, y4, test_size=0.3, random_state=42)

X4_val, X4_test, y4_val, y4_test = train_test_split(X4_temp, y4_temp, test_size=0.5)

rf_regressor_important = RandomForestRegressor(n_estimators=100, random_state=42)

```

rf_regressor_important.fit(X4_train, y4_train)

cv_scores4 = cross_val_score(rf_regressor_important, X4_val, y4_val, cv=5, scoring='neg_mean_squared_error')
cv_rmse_scores4 = np.sqrt(-cv_scores4)
print(f"Cross-Validation RMSE scores: {cv_rmse_scores4}")
print(f"Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores4)}")

```

```

Cross-Validation RMSE scores: [3302989.35726447 2764716.50467697 2622148.04230658
2294059.60211855
3046454.98850998]

```

```
Mean Cross-Validation RMSE: 2806073.6989753097
```

בשלב זה ניסנו לשנות גישה ולנסות לעשות רגרסיה מסוגה XGBoost כמו מומקdem בראנדום פורסט גם פה ניסנו לעשות כל פעם שינוי קטן למודל בשכיל לראות האם הצלחנו לשפר את התוצאה בקאgel המודל הראשון שהרכינו הוא מודל רק עם השינויים אשר עשינו בהתחלה בEDA של ניקוי נתונים והסידור שלהם, לאחר מכן ניסנו לקחת את המשתנים החשובים ל-XGBoost ולהריץ את המודל עליהם, ובנוסף עשינו גם GRIDSEARCH על מנת למצוא את הפרמטרים הטובים למודל שלנו ולסייע לקחנו את הפיצרים החדשניים אשר יצרנו והוסףנו אותו למודל

```

In [ ]: # Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [2, 3, 5],
    'learning_rate': [0.01, 0.1, 0.2],
}

X_5 = dtrain.drop(['timestamp', 'dt', 'price_doc'] + new_features, axis=1)
y_5 = dtrain['price_doc']

X5_train, X5_temp, y5_train, y5_temp = train_test_split(X_5, y_5, test_size=0.3, random_state=42)

# Initialize the XGBRegressor
xgb_regressor = XGBRegressor(random_state=42)

# Initialize the GridSearchCV
grid_search = GridSearchCV(estimator=xgb_regressor, param_grid=param_grid, cv=3, scoring='neg_mean_squared_error')

# Fit the GridSearchCV to the data
grid_search.fit(X5_train, y5_train)

# Get the best parameters
best_params = grid_search.best_params_
print(f"Best parameters: {best_params}")

# Get the best score
best_score = grid_search.best_score_
print(f"Best score: {best_score}")

```

```

In [ ]: X_5 = dtrain.drop(['timestamp', 'dt', 'price_doc'] + new_features, axis=1)
y_5 = dtrain['price_doc']
X5_train, X5_temp, y5_train, y5_temp = train_test_split(X_5, y_5, test_size=0.3, random_state=42)

X5_val, X5_test, y5_val, y5_test = train_test_split(X5_temp, y5_temp, test_size=0.5)
xgb_regressor = XGBRegressor(n_estimators=200, learning_rate=0.1, max_depth=3, random_state=42)
xgb_regressor.fit(X5_train, y5_train)

cv_scores5 = cross_val_score(xgb_regressor, X5_val, y5_val, cv=5, scoring='neg_mean_squared_error')
cv_rmse_scores5 = np.sqrt(-cv_scores5)
print(f"Cross-Validation RMSE scores: {cv_rmse_scores5}")
print(f"Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores5)}")

```

```
Cross-Validation RMSE scores: [3376184.89716455 2689837.78467833 2638629.58093229
2238670.93423733
3045727.28197768]
Mean Cross-Validation RMSE: 2797810.095798035
```

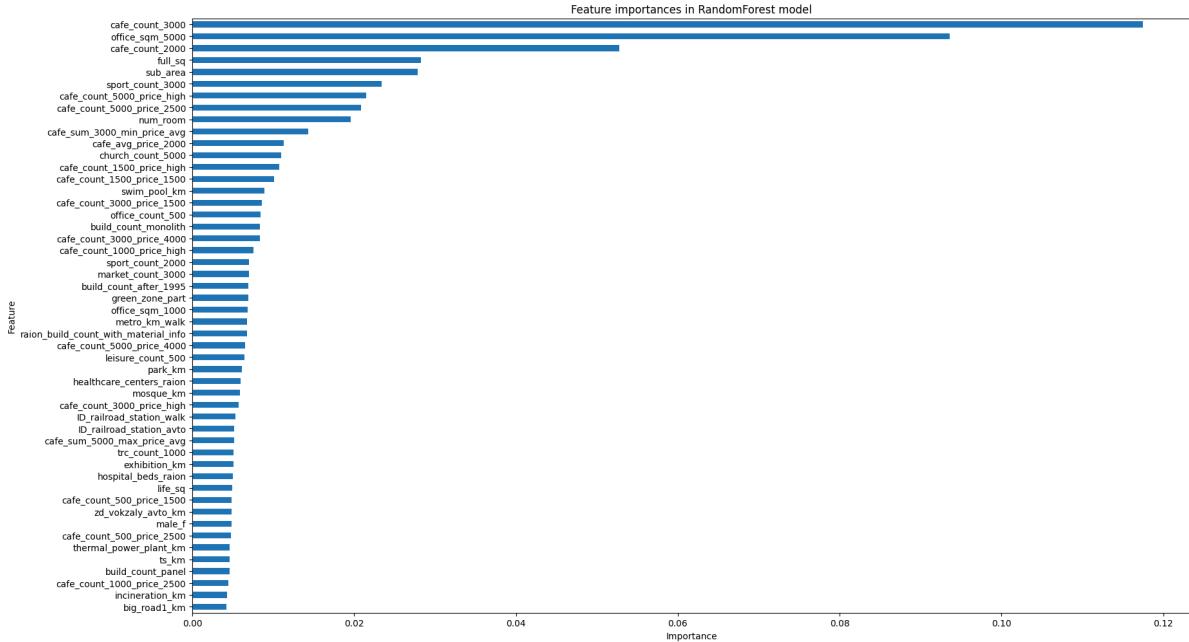
```
In [ ]: X7 = dtrain.drop(['timestamp','dt','price_doc'], axis=1)
y7 = dtrain['price_doc']

X7_train, X7_temp, y7_train, y7_temp = train_test_split(X7, y7, test_size=0.3, random_state=42)

X7_val, X7_test, y7_val, y7_test = train_test_split(X7_temp, y7_temp, test_size=0.5)
xgb_regressor_newfeatures = XGBRegressor(n_estimators=100, colsample_bytree=0.8, gamma=0)
xgb_regressor_newfeatures.fit(X7_train, y7_train)

cv_scores7 = cross_val_score(xgb_regressor_newfeatures, X7_val, y7_val, cv=5, scoring='neg_mean_squared_error')
cv_rmse_scores7 = np.sqrt(-cv_scores7)
print(f"Cross-Validation RMSE scores: {cv_rmse_scores7}")
print(f"Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores7)})")
```

```
In [ ]: # בדיקת המশנים החשובים למודל XGB
importances_xgb = xgb_regressor.feature_importances_
feature_importances_xgb = pd.Series(importances_xgb, index=X5_train.columns).sort_values(ascending=False) # צירט פלוט למשנים האלה
plt.figure(figsize=(20, 12))
feature_importances_xgb[:50].sort_values().plot(kind='barh')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature importances in RandomForest model')
plt.show()
```



```
In [ ]: # שילוב של המশנים החשובים עם הקרוליזה לרשימה אחת
combined_features_xgb = corr_target_f + list(feature_importances_xgb[:50].index)

combined_features_xgb = list(dict.fromkeys(combined_features_xgb))
combined_features_xgb.remove('price_doc')
```

```
In [ ]: X6=dtrain[combined_features_xgb]
y6 = dtrain['price_doc']

X6_train, X6_temp, y6_train, y6_temp = train_test_split(X6, y6, test_size=0.3, random_state=42)

X6_val, X6_test, y6_val, y6_test = train_test_split(X6_temp, y6_temp, test_size=0.5)
```

```

xgb_regressor_import = XGBRegressor(n_estimators=100, colsample_bytree=0.8, gamma=0)
xgb_regressor_import.fit(X6_train, y6_train)

cv_scores6 = cross_val_score(xgb_regressor, X6_val, y6_val, cv=5, scoring='neg_mean_squared_error')
cv_rmse_scores6 = np.sqrt(-cv_scores6)
print(f"Cross-Validation RMSE scores: {cv_rmse_scores6}")
print(f"Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores6)}")

```

Cross-Validation RMSE scores: [3354388.12670876 2729364.82124132 2652036.42535661
2398616.95110883
3195069.31386286]
Mean Cross-Validation RMSE: 2865895.1276556742

```

In [ ]: normalized_train = dtrain.copy()

numerical_cols = normalized_train.select_dtypes(include=[np.number]).columns

scaler = MinMaxScaler()

normalized_train[numerical_cols] = scaler.fit_transform(normalized_train[numerical_cols])

```

```

In [ ]: #בדיקה על דאטה מוגמלת
X8 = normalized_train.drop(['price_doc','dt','timestamp'], axis=1)
target_scaler = MinMaxScaler()
y8 = normalized_train['price_doc'].values.reshape(-1, 1)
y8 = target_scaler.fit_transform(y8)

X8_train, X8_temp, y8_train, y8_temp = train_test_split(X8, y8, test_size=0.3, random_state=42)
X8_val, X8_test, y8_val, y8_test = train_test_split(X8_temp, y8_temp, test_size=0.5)

xgb_regressor_normalised = XGBRegressor(n_estimators=100, colsample_bytree=0.8, gamma=0)
xgb_regressor_normalised.fit(X8_train, y8_train.ravel())

cv_scores8 = cross_val_score(xgb_regressor_normalised, X8_val, y8_val.ravel(), cv=5)
cv_rmse_scores8 = np.sqrt(-cv_scores8)

print(f"Cross-Validation RMSE scores: {cv_rmse_scores8}")
print(f"Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores8)}")

```

Cross-Validation RMSE scores: [0.03002 0.02488327 0.02369948 0.02059078 0.02814356]
Mean Cross-Validation RMSE: 0.025467416211535178

```

In [ ]: #המרת תאריכים לפורט תאריך והוספת עמודה חדשה של שנת הרביעון
dtrain['timestamp'] = pd.to_datetime(dtrain['timestamp'])
dtrain['year_quarter'] = dtrain['timestamp'].dt.to_period('Q')

#חישוב המחיר הממוצע לרבעון השני של 2015 לנורמל
normalization_value = dtrain[dtrain['year_quarter'] == '2015Q2']['price_doc'].mean()

#נורמל מחירים בסט האימון #
dtrain['normalized_price_doc'] = dtrain['price_doc'] / normalization_value

```

```

In [ ]: #בדיקה על דאטה שמנורמלת לפי הרביעון השני של 2015 עם הממוצע של הרביעון השני של 2015
X9 = dtrain.drop(['timestamp','dt','price_doc','year_quarter','normalized_price_doc'], axis=1)
y9 = dtrain['normalized_price_doc']

X9_train, X9_temp, y9_train, y9_temp = train_test_split(X9, y9, test_size=0.3, random_state=42)
X9_val, X9_test, y9_val, y9_test = train_test_split(X9_temp, y9_temp, test_size=0.5)

xgb_regressor_2Qnoraml = XGBRegressor(n_estimators=100, colsample_bytree=0.8, gamma=0)
xgb_regressor_2Qnoraml.fit(X9_train, y9_train)

```

```

cv_scores9 = cross_val_score(xgb_regressor_2Qnoraml, X9_val, y9_val, cv=5, scoring='neg_mean_squared_error')
cv_rmse_scores9 = np.sqrt(-cv_scores9)
print(f"Cross-Validation RMSE scores: {cv_rmse_scores9}")
print(f"Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores9)}")

```

```

Cross-Validation RMSE scores: [0.41592941 0.34328817 0.32695641 0.27952494 0.38649213]
Mean Cross-Validation RMSE: 0.35043821329072505

```

In []:

```

# חישוב מחיר למ"ר בדתאנה פרימס חדש והוספה عمودה חדשה
psqm = dtrain.copy()
imputer = SimpleImputer(strategy='median')

# השלמת ערכים חסרים בעמודות השטח הכלול של הבית בחזין
psqm['full_sq'] = imputer.fit_transform(psqm['full_sq'].values.reshape(-1, 1))
psqm['price_per_sqm'] = psqm['price_doc'] / psqm['full_sq']

# אימון מודל על מחיר למ"ר
psqm = psqm.drop(['timestamp', 'price_doc', 'dt'] + new_features, axis=1)

X10 = psqm.drop(['price_per_sqm', 'full_sq'], axis=1)
y10 = psqm['price_per_sqm']
X10_train, X10_temp, y10_train, y10_temp = train_test_split(X10, y10, test_size=0.3)

X10_val, X10_test, y10_val, y10_test = train_test_split(X10_temp, y10_temp, test_size=0.5)
xgb_regressor_psqm = XGBRegressor(n_estimators=200, learning_rate=0.1, max_depth=3, random_state=42)
xgb_regressor_psqm.fit(X10_train, y10_train)

cv_scores10 = cross_val_score(xgb_regressor_psqm, X10_val, y10_val, cv=5, scoring='neg_mean_squared_error')
cv_rmse_scores10 = np.sqrt(-cv_scores10)
print(f"Cross-Validation RMSE scores: {cv_rmse_scores10}")
print(f"Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores10)}")

```

```

Cross-Validation RMSE scores: [44704.2662155 43570.09176145 45099.46289481 40447.5662986
43134.786524 ]
Mean Cross-Validation RMSE: 43391.234738871266

```

In []:

```

# פתיחה דואתית חדשה לפי סוג הנכס
df_investment = dtrain[dtrain['product_type'] == 0]
df_owner = dtrain[dtrain['product_type'] == 1]

imputer = SimpleImputer(strategy='median')

# השלמת ערכים חסרים בעמודות השטח הכלול של הבית בחזין
df_investment['full_sq'] = imputer.fit_transform(df_investment['full_sq'].values.reshape(-1, 1))
df_owner['full_sq'] = imputer.fit_transform(df_owner['full_sq'].values.reshape(-1, 1))

```

In []:

```

# אימון מודל על מחיר למ"ר לכל סוג נכס #
X_investment = df_investment.drop(['timestamp', 'dt', 'price_doc', 'product_type', 'full_sq'], axis=1)
y_investment = df_investment['price_doc'] / df_investment['full_sq']

X_owner = df_owner.drop(['timestamp', 'dt', 'price_doc', 'product_type', 'full_sq'], axis=1)
y_owner = df_owner['price_doc'] / df_owner['full_sq']

X_investment_train, X_investment_temp, y_investment_train, y_investment_temp = train_test_split(X_investment, y_investment, test_size=0.3)
X_investment_val, X_investment_test, y_investment_val, y_investment_test = train_test_split(X_investment_temp, y_investment_temp, test_size=0.5)

X_owner_train, X_owner_temp, y_owner_train, y_owner_temp = train_test_split(X_owner, y_owner, test_size=0.3)
X_owner_val, X_owner_test, y_owner_val, y_owner_test = train_test_split(X_owner_temp, y_owner_temp, test_size=0.5)

xgb_regressor_investment = XGBRegressor(n_estimators=200, learning_rate=0.1, max_depth=3, random_state=42)
xgb_regressor_owner = XGBRegressor(n_estimators=200, learning_rate=0.1, max_depth=3, random_state=42)

```

```

xgb_regressor_investment.fit(X_investment_train, y_investment_train)
xgb_regressor_owner.fit(X_owner_train, y_owner_train)

cv_scores_investment = cross_val_score(xgb_regressor_investment, X_investment_val,
cv_rmse_scores_investment = np.sqrt(-cv_scores_investment)
print(f"Investment Cross-Validation RMSE scores: {cv_rmse_scores_investment}")
print(f"Investment Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores_investment)}")

cv_scores_owner = cross_val_score(xgb_regressor_owner, X_owner_val, y_owner_val, cv_
cv_rmse_scores_owner = np.sqrt(-cv_scores_owner)
print(f"owner Cross-Validation RMSE scores: {cv_rmse_scores_owner}")
print(f"owner Mean Cross-Validation RMSE: {np.mean(cv_rmse_scores_owner)}")

Investment Cross-Validation RMSE scores: [50127.50701091 50281.82875797 53009.6367
6788 53953.91618008
55148.63946397]
Investment Mean Cross-Validation RMSE: 52504.30563616174
owner Cross-Validation RMSE scores: [30706.53109577 19523.16945435 48586.27752682
21126.26129626
16657.75643453]
owner Mean Cross-Validation RMSE: 27319.99916154505

```

בהתחלת כהאר עבדנו עם הראנדום פורסט השתperf לנו ממהודל ביסטיין שעשינו בו. אך רצינו לראות אם כמו במודל הביסטיין מודל ה-XGBoost יגרום לשיפור עם כל הדאטה. לאחר שעשינו את ה-XGBoost ראיינו באמת שיפור בתוצאות והחליטנו להמשיך עם זה. בנוספ' ידענו שערכים החסרים יכולים לפגוע בראנדום פורסט וגם אחרי טיפול בהם יצא לנו תוצאות יותר טוב ב-XGBoost מהﻄיעות שלו ומשפר את עצמו בהמשך. בנוספ' יש ל-XGBoost מנגנון מיוחד שמנועים ממנו למדוד "יתר על המידה" או להגיב בצורה קיצונית לנתחנים שאינם טיפוסיים, דבר שמסיעו לו לשימור על יציבות ועקביות בביטויים. בנוספ' יכול להתמודד טוב יותר עם ערכים חסרים ולעבד מידע בצורה יעילה יותר, מה שהופך אותו למהיר ויעיל יותר מ- Random Forest בנסיבות רבים. בכלל אלה, לעיתים XGBoost מספק תוצאות טובות יותר לעומת ראנדום פורסט, במיוחד במקרים שבהם הקשר בין הנתונים למטרה הוא מרכיב מיוחד.

התחלת שינוי הדאטה בשכיל הטסט

```

In [ ]: df_test1 = pd.read_csv('C:\\\\סטטיסטיקה\\\\למידת מכונה\\\\פרויקט סיום\\\\odedw\\\\OneDrive\\\\סומן\\\\df_test1.csv')
Id = df_test1['id']

df_test1 = df_test1.drop(['preschool_education_centers_raion','children_preschool'])
df_test1 = df_test1.drop(['ttk_km', 'sadovoe_km','bulvar_ring_km'], axis=1, errors='ignore')
df_test1 = df_test1.drop(['cafe_sum_500_min_price_avg', 'cafe_sum_500_max_price_avg'])
df_test1 = df_test1.drop(['young_female','work_female','work_male','male_f','female'])

In [ ]: #זורה על אותן פעולות נאיון המודל#
df_test1.loc[df_test1['floor'] > df_test1['max_floor'], 'max_floor'] = np.nan

df_test1.loc[df_test1['life_sq'] > df_test1['full_sq'], 'life_sq'] = np.nan

df_test1.loc[df_test1['state'] > 6, 'state'] = 4
df_test1.loc[df_test1['build_year'] > 100000, 'build_year'] = 2007
df_test1.loc[(df_test1['num_room'] == 0) | (df_test1['num_room'] >= 10), 'num_room'] = 10
df_test1.loc[(df_test1['build_year'] < 300) | (df_test1['build_year'] >= 2022), 'build_year'] = 2022

df_test1.loc[[601, 1896, 2791], 'life_sq'] = df_test1.loc[[601, 1896, 2791], 'full_sq']

```

```

df_test1.loc[df_test1['life_sq'] > df_test1['full_sq'], 'life_sq'] = np.nan

for col in ['life_sq', 'full_sq']:
    df_test1.loc[df_test1[col] < 5, col] = np.nan

df_test1.loc[(df_test1['kitch_sq'] >= df_test1['life_sq']) | df_test1['kitch_sq'].isna(), 'kitch_sq'] = np.nan

df_test1.loc[(df_test1['full_sq'] > 150) & (df_test1['life_sq'] / df_test1['full_sq'] < 0.05), 'life_sq'] = np.nan

df_test1.loc[df_test1['life_sq'] > 200, ['life_sq', 'full_sq']] = np.nan

df_test1.loc[df_test1['max_floor'] == 0, 'max_floor'] = np.nan

```

```

In [ ]: label_enc = LabelEncoder()
categorical_columns_test = df_test1.select_dtypes(include=['object']).columns

for col in categorical_columns_test:
    if col != 'timestamp' and col != 'sub_area' and 'dt':
        df_test1[col] = label_enc.fit_transform(df_test1[col].astype(str))

df_test1['sub_area'] = target_enc.transform(df_test1['sub_area'])

```

```

In [ ]: df_test1['area_km'] = df_test1['area_m'] / 1000000
df_test1['density'] = df_test1['raion_popul'] / df_test1['area_km']

```

```

In [ ]: # convert the timestamp to datetime
df_test1['dt'] = pd.to_datetime(df_test1['timestamp'].copy())
# נזקודה התחלה של היום
reference_date = df_test1['dt'].min()

# הפרש מנקודת התחלה
df_test1['day_count'] = (df_test1['dt'] - reference_date).dt.days

# Residential & kitchen area to total area ratio:
df_test1['resident_to_total_ratio'] = df_test1['life_sq']/df_test1['full_sq']
df_test1['kitchen_to_total_ratio'] = df_test1['kitch_sq']/df_test1['full_sq']

df_test1['avg_room_area'] = df_test1['life_sq']/df_test1['num_room']

df_test1['extra_area'] = df_test1['full_sq'] - df_test1['life_sq']

# floor & max_floor to 'floor_rel_total'
df_test1['floor_rel_total'] = df_test1['floor'] / df_test1['max_floor']

```

```

In [ ]: dtest = df_test1.copy()
dtest = dtest.drop(['timestamp', 'dt'] + new_features, axis=1)
imputer = SimpleImputer(strategy='median')

# השלמת ערכים חסרים בעמודות השטח הכלול של הבית בחזין
dtest['full_sq'] = imputer.fit_transform(dtest['full_sq'].values.reshape(-1, 1))

# הילפת ערכים חריגים בערך הנכס ברכנומליות חזי חזי
indices = dtest[dtest['product_type'] == 2].index

midpoint = len(indices) // 2

dtest.loc[indices[:midpoint], 'product_type'] = 0
dtest.loc[indices[midpoint:], 'product_type'] = 1

```

```

In [ ]: # בדיקה האם יש אותם פיצרים בפיינל טט והסידור שלהם
columns_diff_df1 = dtrain.columns.difference(df_test1.columns)

```

```

columns_diff_df2 = dtest.columns.difference(dtrain.columns)

print("Columns in df1 but not in df2:", columns_diff_df1)
print("Columns in df2 but not in df1:", columns_diff_df2)
final_no_price = dtrain.drop(['price_doc', 'timestamp', 'dt', 'full_sq'] + new_features)

# Define the desired order of columns
desired_order = final_no_price.columns

finaltest= dtest[desired_order]

```

Columns in df1 but not in df2: Index(['price_doc'], dtype='object')
Columns in df2 but not in df1: Index(['id'], dtype='object')

In []: *הסתכלות אחורונה לבדוק האם יש ערכים חיריגים לפני העלאה*
finaltest.describe()

Out[]:

	full_sq	life_sq	floor	max_floor	material	build_year	num_room
count	7658.000000	6146.000000	7662.000000	7019.000000	7662.000000	6055.000000	7661.000000
mean	53.698436	33.936992	7.652571	13.666192	1.854216	1988.625764	1.874168
std	19.978035	15.892010	5.099418	5.866484	1.517171	21.899159	0.812492
min	10.000000	8.100000	1.000000	1.000000	1.000000	1875.000000	1.000000
25%	39.200000	20.700000	4.000000	9.000000	1.000000	1969.000000	1.000000
50%	50.420000	30.600000	7.000000	14.000000	1.000000	1986.000000	2.000000
75%	63.300000	42.600000	11.000000	17.000000	2.000000	2013.000000	2.000000
max	403.000000	197.000000	41.000000	48.000000	6.000000	2019.000000	7.000000

8 rows × 276 columns

העלאת קבצי הטסט

In []: *פעולה סופית לפני העלאה*
predictions_rf = rf_regressor_withNA.predict(finaltest)

submission_rf_fulltest5 = pd.DataFrame({
 'Id': Id,
 'price_doc': predictions_rf
})

submission_rf_fulltest5.to_csv('submission_rf_fulltest5.csv', index=False)

In []: *פדריקציה לפי כל הערכים כולל נתונים חסרים*
predictions_xgb_grid = xgb_regressor.predict(finaltest)

predictions_xgb_grid2 = predictions_xgb_grid * 0.973

submission_xgb_grid2= pd.DataFrame({
 'Id': Id,
 'price_doc': predictions_xgb_grid2
})

submission_xgb_grid2.to_csv('submission_xgb_grid2.csv', index=False)

```
In [ ]: #פדריקציה לפי מחיר למ"ר
predictions_xgb_psqm = xgb_regressor_psqm.predict(finaltest)

predictions_xgb_psqm = predictions_xgb_psqm * dtest['full_sq']*0.965

submission_xgb_psqm= pd.DataFrame({
    'Id': Id,
    'price_doc': predictions_xgb_psqm
})

submission_xgb_psqm.to_csv('submission_xgb_psqm.csv', index=False)
```

```
In [ ]: #פדריקציה לפי סוג נכס ומחיר למ"ר
test_investment = finaltest[finaltest['product_type'] == 0]
test_owner = finaltest[finaltest['product_type'] == 1]

test_investment = test_investment.drop(['product_type'], axis=1)
test_owner = test_owner.drop(['product_type'], axis=1)

investment_ids = dtest[dtest['product_type'] == 0]['id']
owner_ids = dtest[dtest['product_type'] == 1]['id']

predictions_investment = xgb_regressor_investment.predict(test_investment) * dtest[dtest['product_type'] == 0]
predictions_owner = xgb_regressor_owner.predict(test_owner) * dtest[dtest['product_type'] == 1]

investment_predictions_df = pd.DataFrame({'id': investment_ids, 'price_doc': predictions_investment})
owner_predictions_df = pd.DataFrame({'id': owner_ids, 'price_doc': predictions_owner})
submission_xgb_psqm_pt = pd.concat([investment_predictions_df, owner_predictions_df])
submission_xgb_psqm_pt = submission_xgb_psqm_pt.sort_values('id')

submission_xgb_psqm_pt.to_csv('submission_xgb_psqm_pt.csv', index=False)
```

```
In [ ]: #פלדיקציה על ידי שילוב מודלים
best_predictions =(( predictions_xgb_psqm + submission_xgb_psqm_pt['price_doc'])/2

submission_xgb_best_predictions = pd.DataFrame({
    'Id': Id,
    'price_doc': best_predictions
})

submission_xgb_best_predictions.to_csv('submission_xgb_best_predictions.csv', index=False)
```

```
In [ ]: #פדריקציה לפי טסט מנורמל
max_price = dtrain['price_doc'].max()

test_data_normalized = finaltest.copy()
numerical_cols = test_data_normalized.select_dtypes(include=[np.number]).columns

scaler_test = MinMaxScaler()

test_data_normalized[numerical_cols] = scaler_test.fit_transform(test_data_normalized)

test_predictions = xgb_regressor_normalised.predict(test_data_normalized)

test_predictions = test_predictions * max_price
```

```
submission_xgb_normalized = pd.DataFrame({  
    'id': Id,  
    'price_doc': test_predictions.ravel() # ravel() is used to convert the 2D array  
})  
  
# Save the submission DataFrame to a CSV file  
submission_xgb_normalized.to_csv('submission_xgb_normalized.csv', index=False)
```