DATA ANALYTICS REPORT AND EXECUTIVE SUMMARY

Research Question

A. Summarize the original real-data research question you identified in task 1. Your summary should include justification for the research question you identified in task 1, a description of the context in which the research question exists, and a discussion of your hypothesis.

Research Question: Is it possible to discover which variables influence price in Russian real estate?

Hypothesis: The region where Russian real estate is located has a statistically significant impact on the price of a given piece of real estate.

If an investor considers the opportunity of financial growth with Russian real estate, it would be crucial to understand how various factors influence the price. The multiple linear regression (MLR) model would help investors to compare the relevant aspects of different real estate opportunities to achieve an ideal minimum price they would likely have to pay.

Data Collection

B. Report on your data-collection process by describing the relevant data you collected, discussing one advantage and one disadvantage of the data-gathering methodology you used, and discussing how you overcome any challenges you encountered during the process of collecting your data.

The data has been previously collected and uploaded at Kaggle.com
https://www.kaggle.com/mrdaniilak/russia-real-estate-20182021

Advantage in using this data set: it has been previously gathered, so there was no difficulty in acquiring it.

The primary disadvantage is the inability to know how accurate the data really is.

Data Extraction and Preparation

C. Describe your data-extraction and preparation process and provide screenshots to illustrate *each* step. Explain the tools and techniques you used for data extraction and data preparation, including how these tools and techniques were used on the data. Justify why you used these particular tools and techniques, including one advantage and one disadvantage when they are used with your data-extraction and -preparation methods.

The chosen dataset contains 13 columns:

- **date** - date of publication of the announcement
- **time** - the time when the ad was published
- **geo_lat** - geographical latitude
- **geo_lon** - geographical longitude
- **region** - Region of Russia where the real estate is at. There are 85 different regions in the country.
- **building_type** - Facade type. 0 - Other. 1 - Panel. 2 - Monolithic. 3 - Brick. 4 - Blocky. 5 - Wooden
- **object_type** - Apartment type. 1 - Secondary real estate market; 2 - New building
- **level** - which building level the real estate is located
- **levels** - Number of stories of the building
- **rooms** - the number of living rooms. If the value is "-1", then it means "studio apartment"
- **area** - the total area of the apartment in m^2
- **kitchen_area** - area of the kitchen in m^2
- **price** - Price of the real estate (Rubles)

In order to analyze the data, the chosen Python IDE is PyCharm Community. The data preparation process begins with basic discovery to determine the attributes of the data set.

First step is to import libraries to work with the dataset:

```
#Importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

#Libraries for Visualization Purposes
import matplotlib.pyplot as plt


from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
import statsmodels.api as sm
from sklearn import model_selection
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn import preprocessing
```

```python
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
from scipy.signal import savgol_filter
from sklearn.model_selection import cross_val_predict
```

Second step: access the dataset using pandas and analyze its features. Here are the steps followed:

1. Determine the size of the dataset with *.shape*
2. Identify the columns within the data using *.columns*
3. Display some rows of data with *.info*
4. Determine if the columns are numerical or categorical with *.dtypes*
5. Identify the basic statistical information of the columns with *.describe()*

```python
#Loading the Real Estate Dataset
real_estate_df = pd.read_csv('/Users/bia/PycharmProjects/D214/all_v2.csv')

#Dataset Size with SHAPE
print(real_estate_df.shape)

#Dataset columns
print(real_estate_df.columns)

#Dataset Info
print(real_estate_df.info)

#Dataset Columns Types
print(real_estate_df.dtypes)

#Basic Stats of the data
print(real_estate_df.describe())
```

```
/Users/bia/PycharmProjects/pythonProject/venv/bin/python /Users/bia/PycharmProjects/D214/main.py
(5477006, 13)
Index(['price', 'date', 'time', 'geo_lat', 'geo_lon', 'region',
       'building_type', 'level', 'levels', 'rooms', 'area', 'kitchen_area',
       'object_type'],
      dtype='object')
<bound method DataFrame.info of             price        date      time  ...  area  kitchen_area  object_type
0         6050000  2018-02-19  20:00:21  ...  82.6         10.8           1
1         8650000  2018-02-27  12:04:54  ...  69.1         12.0           1
2         4000000  2018-02-28  15:44:00  ...  66.0         10.0           1
3         1850000  2018-03-01  11:24:52  ...  38.0          5.0          11
4         5450000  2018-03-01  17:42:43  ...  60.0         10.0           1
...           ...         ...       ...  ...   ...          ...         ...
5477001  19739760  2021-05-01  20:13:58  ...  93.2         13.8          11
5477002  12503160  2021-05-01  20:14:01  ...  45.9          6.6          11
5477003   8800000  2021-05-01  20:14:04  ...  86.5         11.8           1
5477004  11831910  2021-05-01  20:14:12  ...  52.1         18.9          11
5477005  13316200  2021-05-01  20:14:15  ...  55.6         20.8          11

[5477006 rows x 13 columns]>
price              int64
date              object
time              object
geo_lat          float64
geo_lon          float64
region             int64
building_type      int64
level              int64
levels             int64
rooms              int64
area             float64
kitchen_area     float64
object_type        int64
dtype: object
```

```
              price       geo_lat  ...  kitchen_area   object_type
count  5.477006e+06  5.477006e+06  ...  5.477006e+06  5.477006e+06
mean   4.422029e+06  5.403826e+01  ...  1.062840e+01  3.945399e+00
std    2.150752e+07  4.622758e+00  ...  9.792380e+00  4.558357e+00
min   -2.144967e+09  4.145906e+01  ...  1.000000e-02  1.000000e+00
25%    1.950000e+06  5.337768e+01  ...  7.000000e+00  1.000000e+00
50%    2.990000e+06  5.517139e+01  ...  9.700000e+00  1.000000e+00
75%    4.802000e+06  5.622613e+01  ...  1.270000e+01  1.100000e+01
max    2.147484e+09  7.198040e+01  ...  9.999000e+03  1.100000e+01

[8 rows x 11 columns]

Process finished with exit code 0
```

Figure 1: Getting to know the dataset

When .describe is used on the dataset, it identifies a few possible errors. There are negative prices, in addition to unrealistically big and small areas. The total area will be limited to a range between 20 and 200 m^2, and price will be reduced to a range between 1.5 MM Rubles and 50 MM Rubles, where most of the dataset resides. This change to the price column will also take care of the negative price values.

Cleaning the dataset:

```
#Limiting price to 1.5 to 50 MM Rubles
outliers_high = real_estate_df[real_estate_df['price'] > 50000000].index
real_estate_df.drop(outliers_high, inplace = True)

outliers_low = real_estate_df[real_estate_df['price'] < 1500000].index
real_estate_df.drop(outliers_low, inplace = True)

#Removing Real Estates outside 20 - 200 sqm
outliers_low_area = real_estate_df[real_estate_df['area'] < 20].index
real_estate_df.drop(outliers_low_area, inplace = True)

outliers_high_area = real_estate_df[real_estate_df['area'] > 200].index
real_estate_df.drop(outliers_high_area, inplace = True)
```

Validating the changes:

```
print(real_estate_df.price.describe())
print(real_estate_df.area.describe())
```

```
[8 rows x 11 columns]
count    4.747485e+06
mean     4.575473e+06
std      4.115356e+06
min      1.500000e+06
25%      2.300000e+06
50%      3.300000e+06
75%      5.200000e+06
max      5.000000e+07
Name: price, dtype: float64
count    4.747485e+06
mean     5.550804e+01
std      2.150767e+01
min      2.000000e+01
25%      4.000000e+01
50%      5.150000e+01
75%      6.500000e+01
max      2.000000e+02
Name: area, dtype: float64

Process finished with exit code 0
```

Figure 2: Verifying Price and Area Columns

Searching for missing values:

```
#Finding missing values in my dataset
real_estate_df.isnull().any(axis=1)
null_values = real_estate_df.isna().any()
print(null_values)
```

```
dtype: object
price            False
date             False
time             False
geo_lat          False
geo_lon          False
region           False
building_type    False
level            False
levels           False
rooms            False
area             False
kitchen_area     False
object_type      False
dtype: bool

Process finished with exit code 0
```

Figure 3: Searching for missing values

Features and Datatypes:

The data types will be analyzed more carefully, and plot pie charts will be generated in order to better understand each feature.

- Categorical features:
  - **region** (numerically encoded geographical area)
  - **building_type** (numerically encoded type of the building)
  - **object_type** (apartment type, where 1 stands for secondary real estate market, 11 - new building)
- Numerical features:
  - **area**
  - **kitchen_area**
  - **rooms**
  - **level**
  - **levels**
- Geospatial features:
  - **latitude** - geographical coordinate of the property
  - **longitude** - geographical coordinate of the property
- Temporal features:
  - **date**
  - **time**

Pie Charts:

1-) **Building_type**:

It is good practice to start this type of analysis by searching for unique values in order to figure out how to properly display the variables.

```
#Unique values for building_types
print(real_estate_df['building_type'].unique())
```

```
dtype: object
[1 3 4 2 0 5]
```

Figure 4: Unique Values - building_types

Since the **building_type** variable has values from 0 to 5 that don't have an obvious meaning, each value was assigned a type, according to the data dictionary.

```
#Building Type:
def building_type(real_estate_df):
    if real_estate_df['building_type'] == 0:
        return "Other"
    elif real_estate_df['rooms'] == 1:
        return "Panel"
    elif real_estate_df['rooms'] == 2:
        return "Monolithic"
    elif real_estate_df['rooms'] == 3:
        return "Brick"
    elif real_estate_df['rooms'] == 4:
        return "Blocky"
    elif real_estate_df['rooms'] == 5:
        return "Wooden"

df_building_type = real_estate_df.copy()
df_building_type['building_type'] = df_building_type.apply(lambda
df_building_type:building_type(df_building_type), axis=1)

building_type0_count =
df_building_type['building_type'].value_counts()['Other']
building_type1_count =
df_building_type['building_type'].value_counts()['Panel']
building_type2_count =
df_building_type['building_type'].value_counts()['Monolithic']
building_type3_count =
df_building_type['building_type'].value_counts()['Brick']
building_type4_count =
df_building_type['building_type'].value_counts()['Blocky']
building_type5_count =
df_building_type['building_type'].value_counts()['Wooden']

building_type_total = building_type0_count + building_type1_count +
building_type2_count + building_type3_count + \
                      building_type4_count + building_type5_count
```

```
pct_building_type0 = (building_type0_count / building_type_total) * 100
pct_building_type1 = (building_type1_count / building_type_total) * 100
pct_building_type2 = (building_type2_count / building_type_total) * 100
pct_building_type3 = (building_type3_count / building_type_total) * 100
pct_building_type4 = (building_type4_count / building_type_total) * 100
pct_building_type5 = (building_type5_count / building_type_total) * 100

#Pie Chart of Building_type

plt.figure(figsize=(5,5))
labels = ["Other", "Panel", "Monolithic", "Brick", "Blocky", "Wooden"]
values = [pct_building_type0, pct_building_type1, pct_building_type2,
pct_building_type3, pct_building_type4,
          pct_building_type5]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```

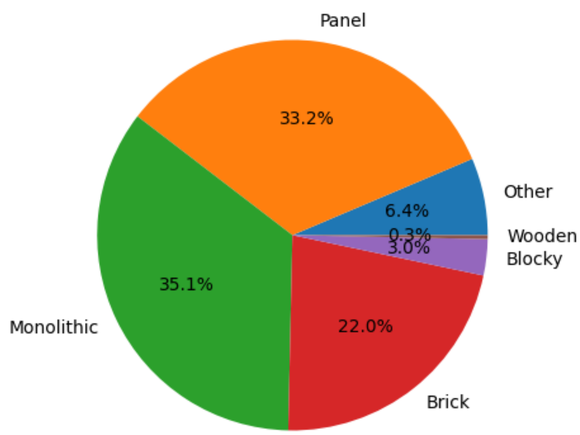

Figure 5: Pie Chart building_type

2-) Object_type:

```
#Unique values for object_type
print(real_estate_df['object_type'].unique())
```

```
dtype: object
[ 1 11]
```

Figure 6: Unique Values - object_type

```
def object_type(real_estate_df):
    if real_estate_df['object_type'] == 1:
        return "Secondary Market"
    elif real_estate_df['object_type'] == 11:
        return "New Building"

df_object_type = real_estate_df.copy()
df_object_type['object_type'] = df_object_type.apply(lambda
df_object_type:object_type(df_object_type), axis=1)

object_type1_count = df_object_type['object_type'].value_counts()['Secondary
Market']
object_type11_count = df_object_type['object_type'].value_counts()['New
Building']


object_type_total = object_type1_count + object_type11_count

pct_object_type1 = (object_type1_count / object_type_total) * 100
pct_object_type11 = (object_type11_count / object_type_total) * 100


#Pie Chart of object_type

plt.figure(figsize=(5,5))
labels = ["Secondary Market", "New Building"]
values = [pct_object_type1, pct_object_type11]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```

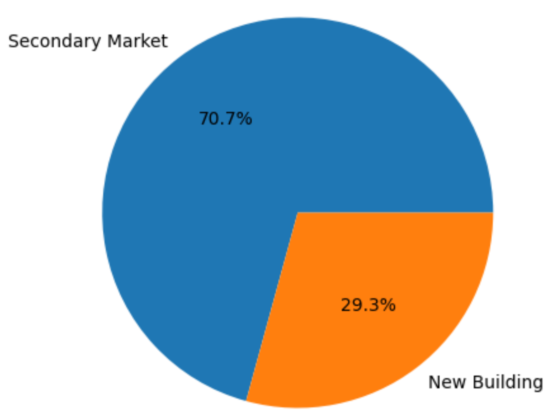

Figure 7: Pie Chart - object_type

3-) Rooms:

```
#Rooms
def room(real_estate_df):
    if real_estate_df['rooms'] == 1:
        return "1 Room"
    elif real_estate_df['rooms'] == 2:
        return "2 Rooms"
    elif real_estate_df['rooms'] == 3:
        return "3 Rooms"
    elif real_estate_df['rooms'] >= 4:
        return "4 Plus Rooms"

df_room = real_estate_df.copy()
df_room['rooms'] = df_room.apply(lambda df_room:room(df_room), axis=1)

room1_count = df_room['rooms'].value_counts()['1 Room']
room2_count = df_room['rooms'].value_counts()['2 Rooms']
room3_count = df_room['rooms'].value_counts()['3 Rooms']
room4_count = df_room['rooms'].value_counts()['4 Plus Rooms']

room_total = room1_count + room2_count + room3_count + room4_count

pct_room1 = (room1_count / room_total) * 100
pct_room2 = (room2_count / room_total) * 100
pct_room3 = (room3_count / room_total) * 100
pct_room4 = (room4_count / room_total) * 100

#Pie Chart of Rooms

plt.figure(figsize=(5,5))
labels = ["1 Room", "2 Rooms", "3 Rooms", "4 Plus Rooms"]
values = [pct_room1, pct_room2, pct_room3, pct_room4]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```

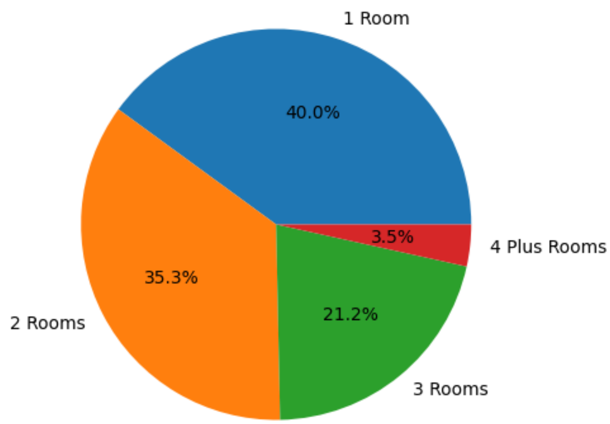

Figure 8: Pie Chart - rooms

4-) Region:

Regions were encoded with numerical IDs. The next step is to determine which regions have the most listings.

```
#Regions are encoded with numeric IDs.
regions = real_estate_df['region'].value_counts()
print(regions.head(10))

plt.hist(regions.values, bins=5)
plt.title('Listings by Region')
plt.show()
```

```
dtype: object
9654    812372
2843    575693
81      480497
2661    453621
3       411225
2922    213368
6171    205363
3230    196428
3991    132777
5282    103757
Name: region, dtype: int64

Process finished with exit code 0
```

Figure 9: First 10 regions with most listings

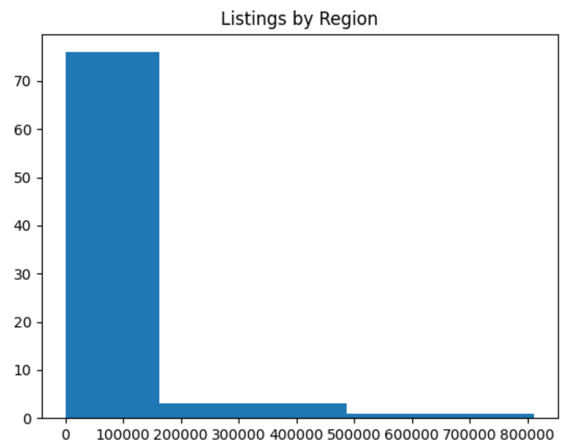

Figure 10: Listing by Region

The regions in this dataset do not have names, only an ID number, so the latitude and longitude of the data will be used to identify each region.

```
#Find out what regions are represented in the data set.
for region in real_estate_df['region'].unique():
    subset = real_estate_df[real_estate_df['region'] == region]
    lat, lon = np.round(subset[['geo_lat', 'geo_lon']].mean(), 2)
    print(f'Region {region}: latitude = {lat}, longitude = {lon}')
```

```
dtype: object
Region 2661: latitude = 59.93, longitude = 30.32
Region 81: latitude = 55.73, longitude = 37.77
Region 2871: latitude = 56.24, longitude = 43.91
Region 2843: latitude = 44.86, longitude = 38.88
Region 3: latitude = 55.72, longitude = 37.58
Region 3106: latitude = 53.25, longitude = 50.08
Region 2922: latitude = 55.75, longitude = 49.91
Region 2722: latitude = 54.61, longitude = 55.76
Region 3230: latitude = 47.27, longitude = 39.73
Region 4417: latitude = 62.41, longitude = 51.94
Region 5282: latitude = 55.08, longitude = 61.2
Region 5368: latitude = 53.14, longitude = 103.66
Region 3446: latitude = 59.99, longitude = 30.41
Region 5520: latitude = 58.03, longitude = 56.19
Region 6171: latitude = 56.88, longitude = 60.6
Region 9579: latitude = 51.92, longitude = 107.65
Region 1010: latitude = 56.91, longitude = 53.26
Region 9648: latitude = 43.5, longitude = 43.61
Region 2604: latitude = 57.62, longitude = 39.77
Region 3019: latitude = 56.11, longitude = 47.27
Region 4982: latitude = 56.61, longitude = 47.89
Region 3870: latitude = 56.22, longitude = 92.79
Region 6817: latitude = 53.26, longitude = 83.78
Region 2900: latitude = 44.62, longitude = 42.37
Region 3991: latitude = 57.17, longitude = 65.67
Region 5241: latitude = 54.19, longitude = 45.15
Region 9654: latitude = 55.0, longitude = 82.96
Region 2072: latitude = 51.64, longitude = 39.27
Region 8090: latitude = 61.89, longitude = 34.16
Region 4007: latitude = 42.9, longitude = 47.54
Region 13919: latitude = 43.09, longitude = 44.65
Region 11171: latitude = 60.56, longitude = 125.07
Region 10160: latitude = 52.01, longitude = 113.59
Region 2860: latitude = 55.14, longitude = 86.23
Region 7873: latitude = 44.59, longitude = 33.5
Region 2359: latitude = 53.69, longitude = 91.41
Region 2594: latitude = 58.5, longitude = 49.54
```

Figure 11: Geo Coordinates for Each Region in the Dataset

A google maps search of the coordinates for the first region (number ID 2661) reveals that the region is the city of Saint Petersburg.

Since the goal of this project is to find the best possible real estate model to predict prices and finding a good model using all of the regions would not be practical, only the region of Saint Petersburg will be considered for this project.

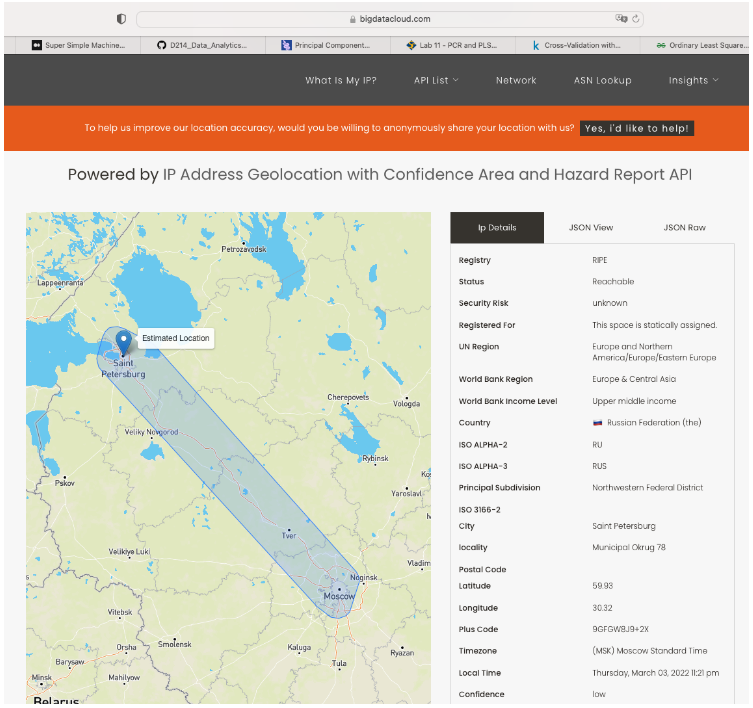

Figure 12: Geo Coordinates for the Chosen Region (2661)

```
#Removing Regions that are not ID 2661
regions = real_estate_df[real_estate_df['region'] != 2661].index
real_estate_df.drop(regions, inplace = True)
```

```
#Let's re analyze our dataset
#Dataset Size with SHAPE
print(real_estate_df.shape)

#Dataset columns
print(real_estate_df.columns)

#Dataset Info
print(real_estate_df.info)

#Dataset Columns Types
print(real_estate_df.dtypes)

#Basic Stats of the data
print(real_estate_df.describe())
```

```
(453621, 11)
Index(['price', 'geo_lat', 'geo_lon', 'region', 'building_type', 'level',
       'levels', 'rooms', 'area', 'kitchen_area', 'object_type'],
      dtype='object')
<bound method DataFrame.info of             price    geo_lat    geo_lon  ...  area  kitchen_area  object_type
0         6050000  59.805808  30.376141  ...  82.6          10.8            1
7         3600000  59.875526  30.395457  ...  31.1           6.0            1
36        3200000  59.827465  30.201827  ...  31.0           7.0            1
47        6500000  59.988334  29.786928  ...  89.0          10.0            1
54        6300000  59.911622  30.284556  ...  99.9          14.5            1
...           ...        ...        ...  ...   ...           ...          ...
5476909   8000000  59.951248  30.492657  ...  60.0           6.2            1
5476937  12500000  59.853716  30.396701  ...  67.0           6.0            1
5476949  30000000  59.961501  30.255689  ...  92.0          21.6            1
5476964   9600000  59.907618  30.322752  ...  62.0           8.6            1
5476998   4900000  59.850103  30.357299  ...  31.0           6.0            1

[453621 rows x 11 columns]>
price              int64
geo_lat          float64
geo_lon          float64
region             int64
building_type      int64
level              int64
levels             int64
rooms              int64
area             float64
kitchen_area     float64
object_type        int64
dtype: object
              price        geo_lat  ...   kitchen_area    object_type
count  4.536210e+05  453621.000000  ...  453621.000000  453621.000000
mean   7.663511e+06      59.932700  ...      12.611991       5.304541
std    5.601934e+06       0.084982  ...       7.006627       4.951403
min    1.500000e+06      59.647383  ...       0.050000       1.000000
25%    4.437600e+06      59.863116  ...       8.500000       1.000000
50%    5.940000e+06      59.939084  ...      11.100000       1.000000
75%    8.711033e+06      60.000338  ...      15.070000      11.000000
max    5.000000e+07      60.241984  ...    1272.000000      11.000000

[8 rows x 11 columns]

Process finished with exit code 0
```

Figure 13: Analysis of the Dataset containing only one region

The columns “date” and “time” are irrelevant to the analysis, so they were dropped.

```
# #Removing some unnecessary data
real_estate_df = real_estate_df.drop(columns=['date', 'time'])
```

Histograms were created to visualize the remaining data:

```
#Histogram for Numerical Data
dataset = real_estate_df[['price', 'geo_lat', 'geo_lon', 'region',
'building_type', 'level',
                      'levels','rooms', 'area', 'kitchen_area', 'object_type']]
fig = plt.figure(figsize=(15, 12))
plt.suptitle('Histograms - Numerical Data \n', horizontalalignment="center",
fontstyle="normal", fontsize=24,
              fontfamily="sans-serif")
for i in range(dataset.shape[1]):
```

```
    plt.subplot(6, 3, i + 1)
    f = plt.gca()
    f.set_title(dataset.columns.values[i])
    vals = np.size(dataset.iloc[:, i].unique())
    if vals >= 100:
        vals = 100
    plt.hist(dataset.iloc[:, i], bins=vals, color='#ec838a')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

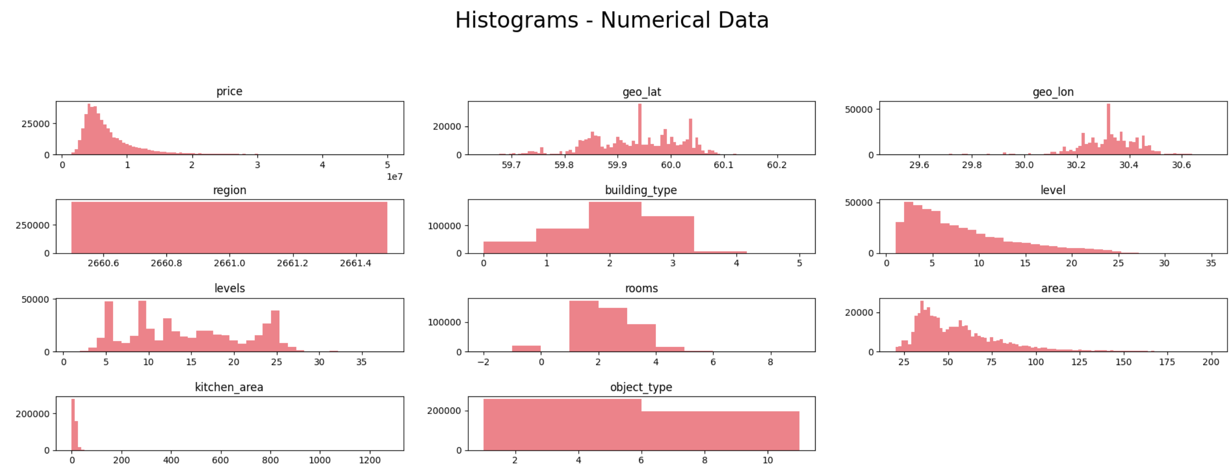

Figure 14: Histograms

```
#Correlation Matrix
#Since region is only one value, lets drop it for the correlation matrix
data = real_estate_df.drop(columns=['region'])
sns.heatmap(data.corr(), annot = True)
plt.show()
```

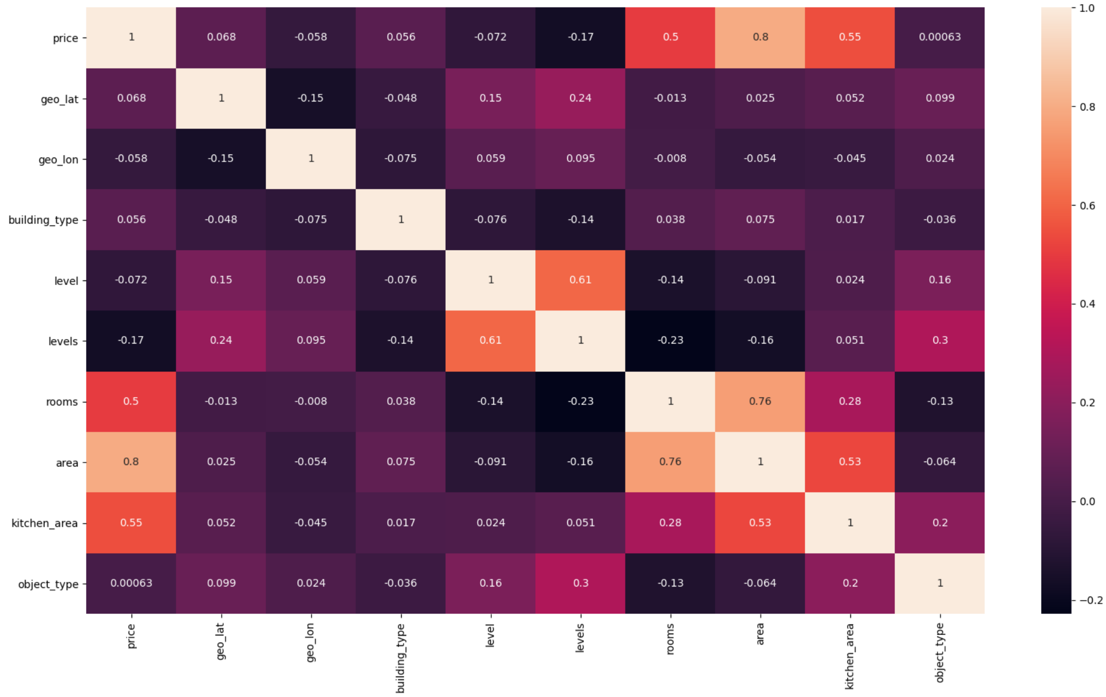

Figure 15: Correlation Matrix

```
#Plotting features and Price correlations in descending order
data.corr()['price'].sort_values(ascending = False).plot(kind ='bar', figsize
= (10, 5), color = 'Red')
plt.title('Correlation Of Variables With Price', fontsize = 20, fontweight =
'bold')
plt.xticks(fontsize = 10, fontweight = 'bold')
plt.yticks(fontweight = 'bold', fontsize = 10)
plt.show()
```

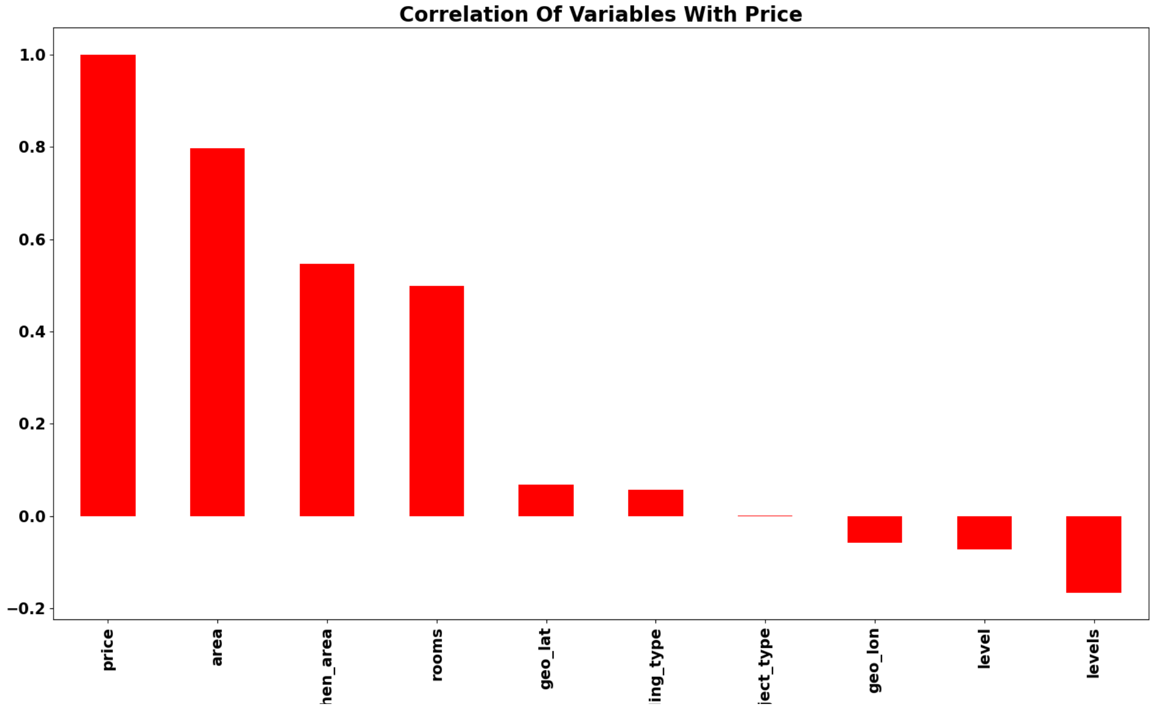

Figure 16: Correlations with Price in Descending Order

Analysis

D. Report on your data-analysis process by describing the analysis technique(s) you used to appropriately analyze the data. Include the calculations you performed and their outputs. Justify how you selected the analysis technique(s) you used, including one advantage and one disadvantage of these technique(s).

Linear Regression is a standard tool to analyze the relationship in between two or more variables. The most common technique to determine the parameters of the linear equation is the Ordinary Least Square (OLS). This model finds the parameters for a linear equation, while at the same time minimizing the sum of the squared residuals[1].

The model is constructed in *statsmodel* using the OLS function:

```
#Approach 1: OLS Model (Ordinary Least Squares) using statsmodel
#OLS Model Summary
real_estate_df['intercept'] = 1
lm_real_estate = sm.OLS(real_estate_df['price'],
real_estate_df[['geo_lon','geo_lat', 'building_type',
                                                                   'level',
'levels', 'rooms', 'area', 'kitchen_area',
```

```
'object_type','intercept']]).fit()

print(lm_real_estate.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.685
Model:                            OLS   Adj. R-squared:                  0.685
Method:                 Least Squares   F-statistic:                 1.096e+05
Date:                Thu, 03 Mar 2022   Prob (F-statistic):               0.00
Time:                        18:27:04   Log-Likelihood:            -7.4304e+06
No. Observations:              453621   AIC:                         1.486e+07
Df Residuals:                  453611   BIC:                         1.486e+07
Df Model:                           9
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
geo_lon         5.45e+05   3.97e+04     13.718      0.000    4.67e+05    6.23e+05
geo_lat        3.861e+06   5.76e+04     66.986      0.000    3.75e+06    3.97e+06
building_type -7.681e+04   4974.082    -15.443      0.000   -8.66e+04   -6.71e+04
level          3.626e+04   1033.392     35.089      0.000    3.42e+04    3.83e+04
levels        -1.014e+05    900.831   -112.533      0.000   -1.03e+05   -9.96e+04
rooms         -1.157e+06   6882.473   -168.108      0.000   -1.17e+06   -1.14e+06
area           1.835e+05    314.319    583.797      0.000    1.83e+05    1.84e+05
kitchen_area   1.185e+05    833.935    142.152      0.000    1.17e+05     1.2e+05
object_type    2.728e+04   1022.111     26.691      0.000    2.53e+04    2.93e+04
intercept      -2.49e+08   3.86e+06    -64.586      0.000   -2.57e+08   -2.41e+08
==============================================================================
Omnibus:                   258597.266   Durbin-Watson:                   1.551
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         28442956.536
Skew:                           1.817   Prob(JB):                         0.00
Kurtosis:                      41.622   Cond. No.                     7.63e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.63e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Process finished with exit code 0
```

Figure 17: OLS Results

The price within the region of Saint Petersburg was determined using this equation:

Price = -2.49e+08 + 5.45e+05*geo_lon + 3.861e+06*geo_lat - 7.681e+04*building_type + 3.626e+04*level - 1.014e+05*levels -1.157e+06*rooms + 1.835e+05*area + 1.185e+05*kitchen_area + 2.728e+04*object_type

This model has a variance of 68.5%. p-values at zero are statically significant for all variables.

Advantage: different variables affect the output in different proportions and OLS makes this very clear by calculating the coefficients.

Disadvantage: a large dataset is needed to achieve a decent accuracy when using OLS

The second approach for analysis: Use PCA on the Linear Regression with Cross Validation:

Principal Component Analysis (PCA) is a widely used technique to reduce the dimension of a feature space. For example, in a system with 10 independent variables, 10 “new” variables are created that are a combination of each of the 10 original ones. The new variables are created in a specific way, and ordered by how well they predict the dependent variable (in this present case, price of a real estate property). These new 10 variables carry the most valuable parts of the original variables. Therefore, it’s not a problem to drop some of the newly created variables. A huge benefit of using PCA is that the new variables are independent of one another[2].

When PCA is applied to a linear regression model it is called principal component regression (PCR).

```
#Target is the price (Y)
y_data = data.price.values

#Remove price from remaining features
X_data = data.drop('price', axis = 1)

#Proposal 2 with PCA and CV
pca = PCA()
X_reduced = pca.fit_transform(scale(X_data))

#Lets see principal components of first few
print(pd.DataFrame(pca.components_.T).loc[:4,:5])

#Cross Validation
#10-fold CV, with shuffle
n = len(X_reduced)
kf_10 = model_selection.KFold(n_splits=10, shuffle=True, random_state=100)

regr = LinearRegression()
mse = []

#Calculate MSE with only the intercept (no principal components in
regression)
score = -1 * model_selection.cross_val_score(regr, np.ones((n, 1)),
y_data.ravel(), cv=kf_10,

scoring='neg_mean_squared_error').mean()
mse.append(score)

#Calculate MSE using Cross Validation for the 10 principle components, adding
one component at the time.
for i in np.arange(1, 11):
    score = -1 * model_selection.cross_val_score(regr, X_reduced[:, :i],
y_data.ravel(), cv=kf_10,
```

```
scoring='neg_mean_squared_error').mean()
    mse.append(score)
```

In the analysis above, a linear regression between the variables **X_reduced** and **y** is built. It is a linear relation between principal components and the corresponding price. It is expected that the *prediction* of the real estate price in other samples (not included in the calibration) will be accurate.

In order to make predictions, future (unknown) data need to be handled with an acceptable level of accuracy. For that, an independent set of real estate data, often referred to as *validation* data is needed. If there isn't an independent validation set, the next best thing is to split the input data into calibration and cross-validation sets. Only the calibration data is used to build the regression model. The cross-validation data is then used as an independent data set to verify the predictive value of the model. Scikit-learn comes with a handy function to do just that. To simplify: y = cross_val_predict(regr, X_reduced, y, cv=10). The parameter cv=10 means that the data is divided into 10 parts, with one part being withheld for cross validation.

```
plt.plot(np.array(mse), '-v')
plt.xlabel('Number of principal components in regression: Training')
plt.ylabel('MSE')
plt.title('Price')
plt.xlim(xmin=-1)
plt.show()
```

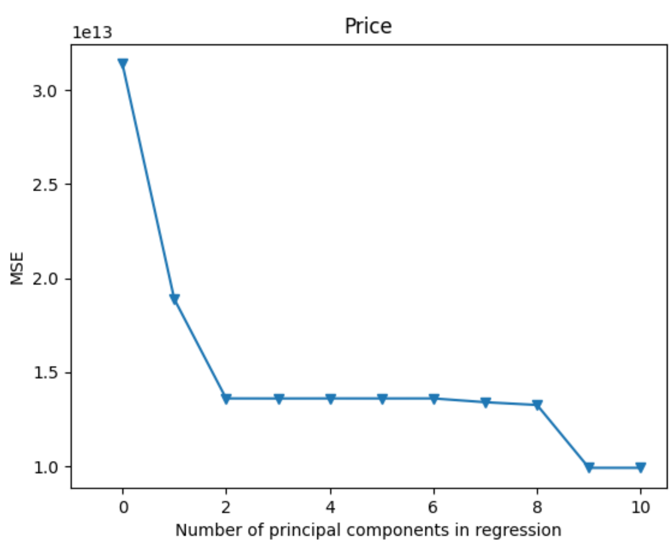

Figure 18: Number of PC in Regression

Based on the sudden jump at the second principal component, it can be assumed that at least two principal components are required to describe the data.

```
#Explained Variance for each component
print('Explained Variance is ',
np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100))
```

```
[8 rows x 11 columns]
          0         1         2         3         4         5
0 -0.107720  0.302715  0.525480 -0.354613 -0.016777 -0.684388
1 -0.082042 -0.003945 -0.779001  0.060902 -0.141454 -0.581653
2  0.130631 -0.099312  0.291330  0.697354 -0.599904 -0.194922
3 -0.336837  0.422277 -0.091445 -0.089567 -0.451676  0.327489
4 -0.404377  0.458779 -0.070379 -0.032509 -0.201326  0.096601
Explained Variance is  [ 25.03  44.97  57.63  68.68  78.97  87.45  93.96  97.91 100.  ]
```

Figure 19: Explained Variance for Each of the 9 Components

```
#PCA on Training Data
pca2 = PCA()

#Split into training and test sets
X_train, X_test , y_train, y_test =
model_selection.train_test_split(X_reduced, y_data, test_size=0.3,
random_state=1)

#Scale the data
X_reduced_train = pca2.fit_transform(scale(X_train))
n = len(X_reduced_train)

#10-fold CV, with shuffle
kf_10 = model_selection.KFold(n_splits=10, shuffle=True, random_state=100)

regr = LinearRegression()
mse = []

#Calculate MSE with only the intercept (no principal components in
regression)
score = -1*model_selection.cross_val_score(regr, np.ones((n,1)),
y_train.ravel(), cv=kf_10, scoring='neg_mean_squared_error').mean()
mse.append(score)

#Calculate MSE using CV for the 9 principle components, adding one component
at the time.
for i in np.arange(1, 10):
    score = -1*model_selection.cross_val_score(regr, X_reduced_train[:,:i],
y_train.ravel(), cv=kf_10, scoring='neg_mean_squared_error').mean()
    mse.append(score)

plt.plot(np.array(mse), '-v')
plt.xlabel('Number of principal components in regression: Training')
plt.ylabel('MSE')
```

```
plt.title('Price')
plt.xlim(xmin=-1)
plt.show()

X_reduced_test = pca2.transform(scale(X_test))[:,:10]

#Train regression model on training data
regr.fit(X_reduced_train[:,:10], y_train)

#Prediction with test data
pred = regr.predict(X_reduced_test)

print('MSE: %.3f' % mean_squared_error(y_test, pred))
print('R^2: %.3f' % r2_score(y_test, pred))
```

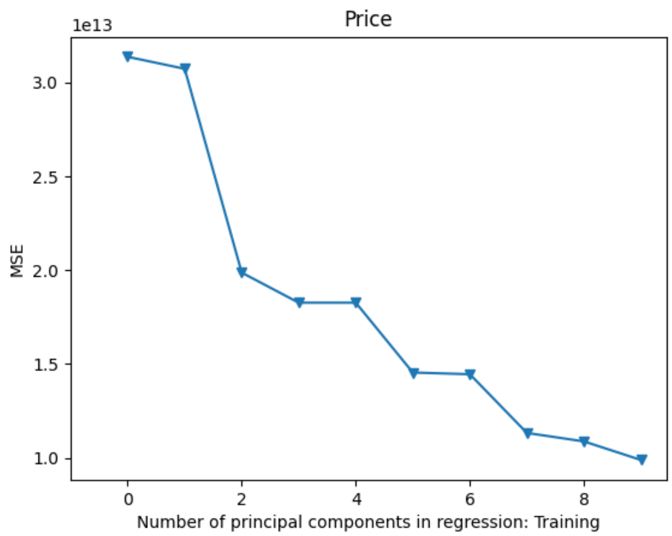

Figure 20: Number of PC in Regression - Training

According to the graph seen previously in **Figure 20**, the MSE value keeps dropping until the last component, so it is safe to say that all nine components would be needed to achieve the best model.

```
3 -0.336837  0.422277 -0.091445 -0.089567 -0.451676  0.327489
4 -0.404377  0.458779 -0.070379 -0.032509 -0.201326  0.096601
Explained Variance is  [ 25.03  44.97  57.63  68.68  78.97  87.45  93.96  97.91 100.  ]
MSE: 10000912440304.393
R^2: 0.681
```

Figure 22: MSE and R^2

Using all nine components, this model presents a similar accuracy to the OLS model.

Visualizing the importance of the model:

```
#Get importance
importance = regr.coef_
#Summarize feature importance
for i,v in enumerate(importance):
   print('Feature: %0d, Score: %.5f' % (i,v))
#Plot feature importance
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```

```
Feature: 0, Score: 790142.40318
Feature: 1, Score: -3280993.80685
Feature: 2, Score: 1263010.46867
Feature: 3, Score: 13052.17027
Feature: 4, Score: 1927297.82891
Feature: 5, Score: 299757.86069
Feature: 6, Score: -1770200.42468
Feature: 7, Score: 672644.85152
Feature: 8, Score: 1033859.62464
```

Figure 23: Features and Scores

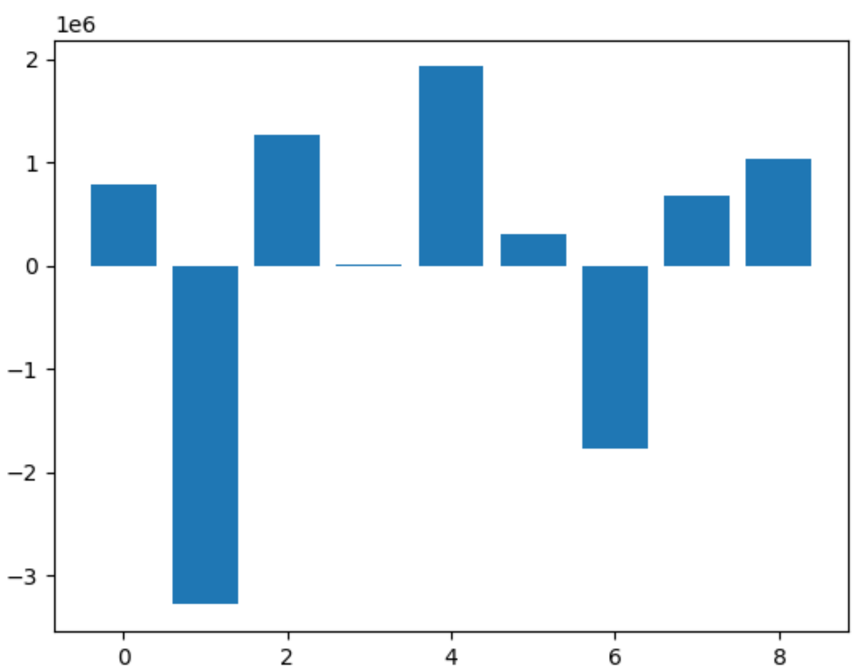

Figure 24: Importance of Each Principal Component

Data Summary and Implications

E. Summarize the implications of your data analysis by discussing the results of your data analysis in the context of the research question, including one limitation of your analysis. Within the context of your research question, recommend a course of action based on your results. Then propose **two** directions or approaches for future study of the data set.

```
#Plotting preds and actual values
plt.figure(figsize=(10,10))
plt.scatter(y_test, pred, c='crimson')
plt.yscale('log')
plt.xscale('log')

p1 = max(max(pred), max(y_test))
p2 = min(min(pred), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.show()
```

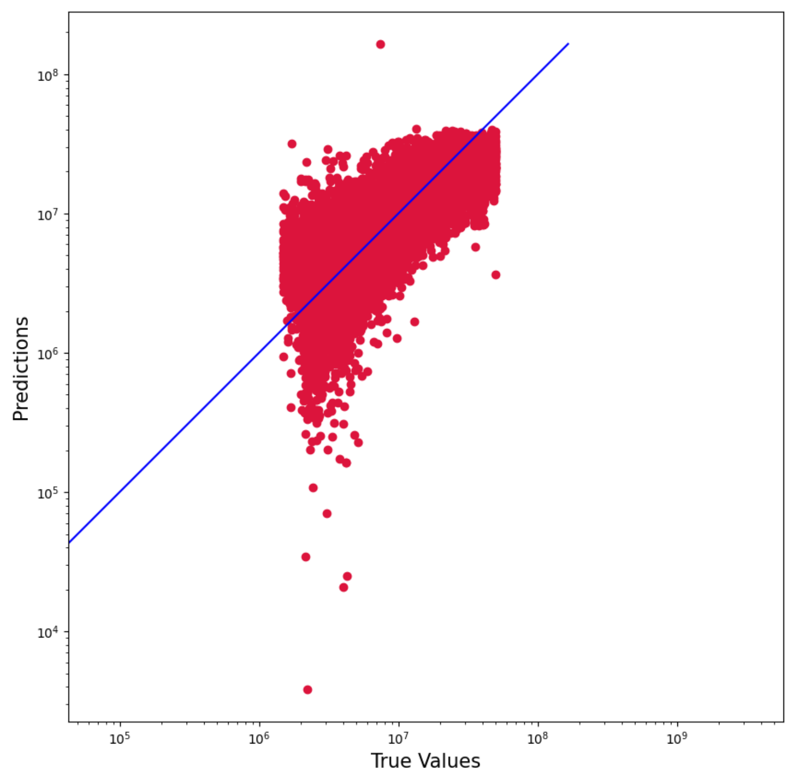

Figure 25: Predictions vs True Values

Price = -2.49e+08 + 5.45e+05*geo_lon + 3.861e+06*geo_lat - 7.681e+04*building_type + 3.626e+04*level - 1.014e+05*levels -1.157e+06*rooms + 1.835e+05*area + 1.185e+05*kitchen_area + 2.728e+04*object_type

This is the equation to predict price values based on the acquired data from the city of Saint Peterburg. The two variables with the strongest influence on price are: the location (geo_lat/geo_lon) and the number of rooms.

Limitation: unfortunately, there are mistakes in the original dataset and there is no way to know how the data was collected, and if it is accurate. That introduces errors into the analysis and results.

Course of Action: Once investors know the features that most impact the price of real estate, they could make conscious decisions if the particular real estate property is worth investing in or not.

Two Directions or Approaches: For future study of the dataset I would recommend performing the feature engineering phase differently. One approach could be grouping similar properties into clusters using KMeans.

The second recommendation would be analyzing the same dataset using another regressor like Decision Tree Regressor, Random Forest, Ridge or Lasso Regression and compare with the results obtained here.

F. Acknowledge sources, using in-text citations and references, for content that is quoted.

[1] (Unknown) SARGENT, Thomas J. and STACHURSKI, John. Linear Regression in Python https://python.quantecon.org/ols.html

[2] (Dec, 27th 2021) BISCHOFF, Bianca. Assessment Document for D212