# Regression Exam

## AIM: Predict price_aprox_usd for the houses in Aires Beuoe..

In [ ]:
```
#Imports
#EDA
import pandas as pd
from skimpy import skim
#Plots
import seaborn as sns
import matplotlib.pyplot as plt

#ML Algorithms
from category_encoders import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV, cross_val_score
#ML Metrices
import sklearn.metrics as metrics
from sklearn.metrics import accuracy_score, mean_absolute_error, r2_score, root_

#Save models
import joblib
```

In [ ]:
```
data_df = pd.read_csv("./data/buenos-aires-real-estate-1.csv")
print(data_df.info())
print(f"The size of dathe dataset is: {len(data_df)}")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8606 entries, 0 to 8605
Data columns (total 16 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   operation                   8606 non-null   object 
 1   property_type               8606 non-null   object 
 2   place_with_parent_names     8606 non-null   object 
 3   lat-lon                     6936 non-null   object 
 4   price                       7590 non-null   float64
 5   currency                    7590 non-null   object 
 6   price_aprox_local_currency  7590 non-null   float64
 7   price_aprox_usd             7590 non-null   float64
 8   surface_total_in_m2         5946 non-null   float64
 9   surface_covered_in_m2       7268 non-null   float64
 10  price_usd_per_m2            4895 non-null   float64
 11  price_per_m2                6520 non-null   float64
 12  floor                       1259 non-null   float64
 13  rooms                       4752 non-null   float64
 14  expenses                    875 non-null    object 
 15  properati_url               8606 non-null   object 
dtypes: float64(9), object(7)
memory usage: 1.1+ MB
None
The size of dathe dataset is: 8606
```

view missing values

In [ ]:
```
print(f"Missing values for the dataset:\n")
print(data_df.isna().sum())
```

```
Missing values for the dataset:

operation                        0
property_type                    0
place_with_parent_names          0
lat-lon                       1670
price                         1016
currency                      1016
price_aprox_local_currency    1016
price_aprox_usd               1016
surface_total_in_m2           2660
surface_covered_in_m2         1338
price_usd_per_m2              3711
price_per_m2                  2086
floor                         7347
rooms                         3854
expenses                      7731
properati_url                    0
dtype: int64
```

Use skimpy to further understand dataset

In [ ]:
```
skim(data_df)
```

skimpy summary

*Data Summary*

| dataframe | Values |
|---|---|
| Number of rows | 8606 |
| Number of columns | 16 |

*Data Types*

| Column Type | Count |
|---|---|
| float64 | 9 |
| string | 7 |

*number*

| column_name | NA | NA % | mean | sd | p0 | p25 | p5 |
|---|---|---|---|---|---|---|---|
| price | 1016 | 11.81 | 300000 | 510000 | 0 | 95000 | 1 |
| price_aprox_local_currency | 1016 | 11.81 | 3600000 | 4800000 | 0 | 1400000 | 22 |
| price_aprox_usd | 1016 | 11.81 | 240000 | 310000 | 0 | 90000 | 1 |
| surface_total_in_m2 | 2660 | 30.91 | 250 | 940 | 0 | 48 | |
| surface_covered_in_m2 | 1338 | 15.55 | 140 | 760 | 0 | 46 | |
| price_usd_per_m2 | 3711 | 43.12 | 1700 | 1600 | 0 | 920 | |
| price_per_m2 | 2086 | 24.24 | 3700 | 12000 | 2.2 | 1400 | |
| floor | 7347 | 85.37 | 8.8 | 60 | 1 | 2 | |
| rooms | 3854 | 44.78 | 3.1 | 1.4 | 1 | 2 | |

*string*

| column_name | NA | NA % | words pe |
|---|---|---|---|
| operation | 0 | 0 | |
| property_type | 0 | 0 | |
| place_with_parent_names | 0 | 0 | |
| lat-lon | 1670 | 19.41 | |
| currency | 1016 | 11.81 | |
| expenses | 7731 | 89.83 | |
| properati_url | 0 | 0 | |

End

view column expenses nee if its suppose to be numeric:

In [ ]:
```
print(len(data_df['expenses']))
print(data_df['expenses'].dropna())
print(len(data_df['expenses']))
```

```
8606
8       3400
26         2
40       364
41       450
48       300
        ...
8534     600
8546    2100
8557    1300
8565    2400
8582    3749
Name: expenses, Length: 875, dtype: object
8606
```

convert object datatype to numeric

In [ ]:
```
data_df['expenses'] = pd.to_numeric(data_df['expenses'], errors='coerce')
print(data_df['expenses'].dtype)
```

```
float64
```

In [ ]:
```
#change column lat-lon to 2 columns of numeric datataype
data_df[['lat', 'lon']] = (data_df['lat-lon'].str.split(',',
                          expand = True).astype(float))
data_df.drop(columns=['lat-lon'], inplace=True)
```

In [ ]:
```
#view the % of missing values in each column
missing_values = data_df.isna().sum()*100/len(data_df)
print(f"{missing_values.apply(lambda x: f"{x:.2f}%")}")
```

```
operation                      0.00%
property_type                  0.00%
place_with_parent_names        0.00%
price                         11.81%
currency                      11.81%
price_aprox_local_currency    11.81%
price_aprox_usd               11.81%
surface_total_in_m2           30.91%
surface_covered_in_m2         15.55%
price_usd_per_m2              43.12%
price_per_m2                  24.24%
floor                         85.37%
rooms                         44.78%
expenses                      89.83%
properati_url                  0.00%
lat                           19.41%
lon                           19.41%
dtype: object
```

Drop leaky features

- price
- price_usd_per_m2
- price_per_m2
- price_aprox_local_currency

In [ ]:
```
#drop leaky features
data_df.drop(columns=['price', 'price_usd_per_m2',
```

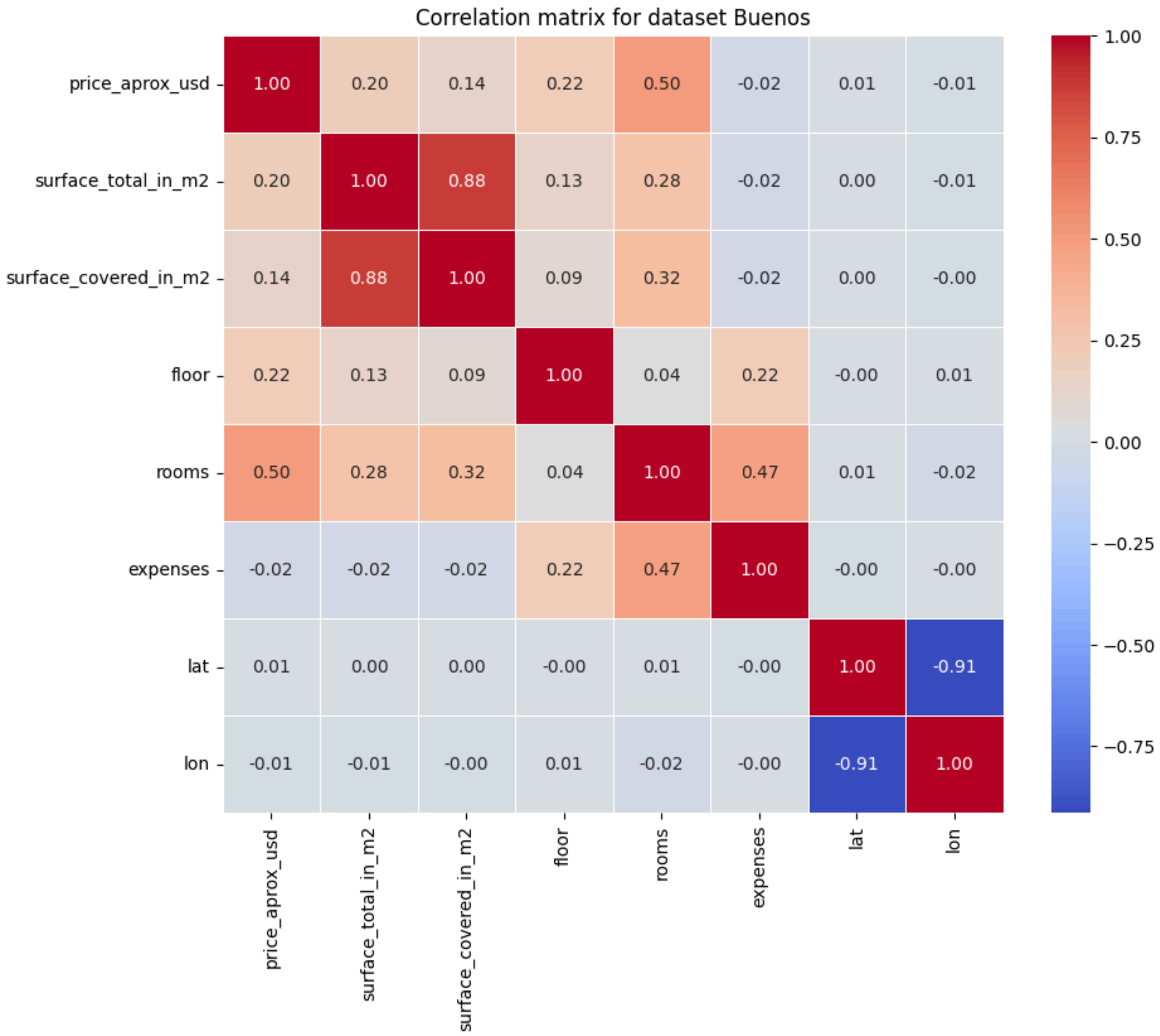
```
'price_per_m2', 'price_aprox_local_currency'], inplace=True)
print(data_df.columns)
```

```
Index(['operation', 'property_type', 'place_with_parent_names', 'currency',
       'price_aprox_usd', 'surface_total_in_m2', 'surface_covered_in_m2',
       'floor', 'rooms', 'expenses', 'properati_url', 'lat', 'lon'],
      dtype='object')
```

correlation matrix

In [ ]:
```
def plt_heat_map():
    corr_df = data_df.select_dtypes('number').corr()
    plt.figure(figsize=(10,8))
    sns.heatmap(corr_df, annot=True, cmap="coolwarm",
                fmt=".2f", linewidths=0.5)
    plt.title("Correlation matrix for dataset Buenos")
    plt.show()
    return corr_df
```

In [ ]:
```
corr_df = plt_heat_map()
```




it is seen from the correlation matrix that surface_total_in_m2 and surface_covered_in_m2 have a high correlation coefficient of 0.88. These two are highly linear related and provide very similar information.

- surface_total_in_m2: This generally refers to the total area of the property, including all spaces (both covered and uncovered).

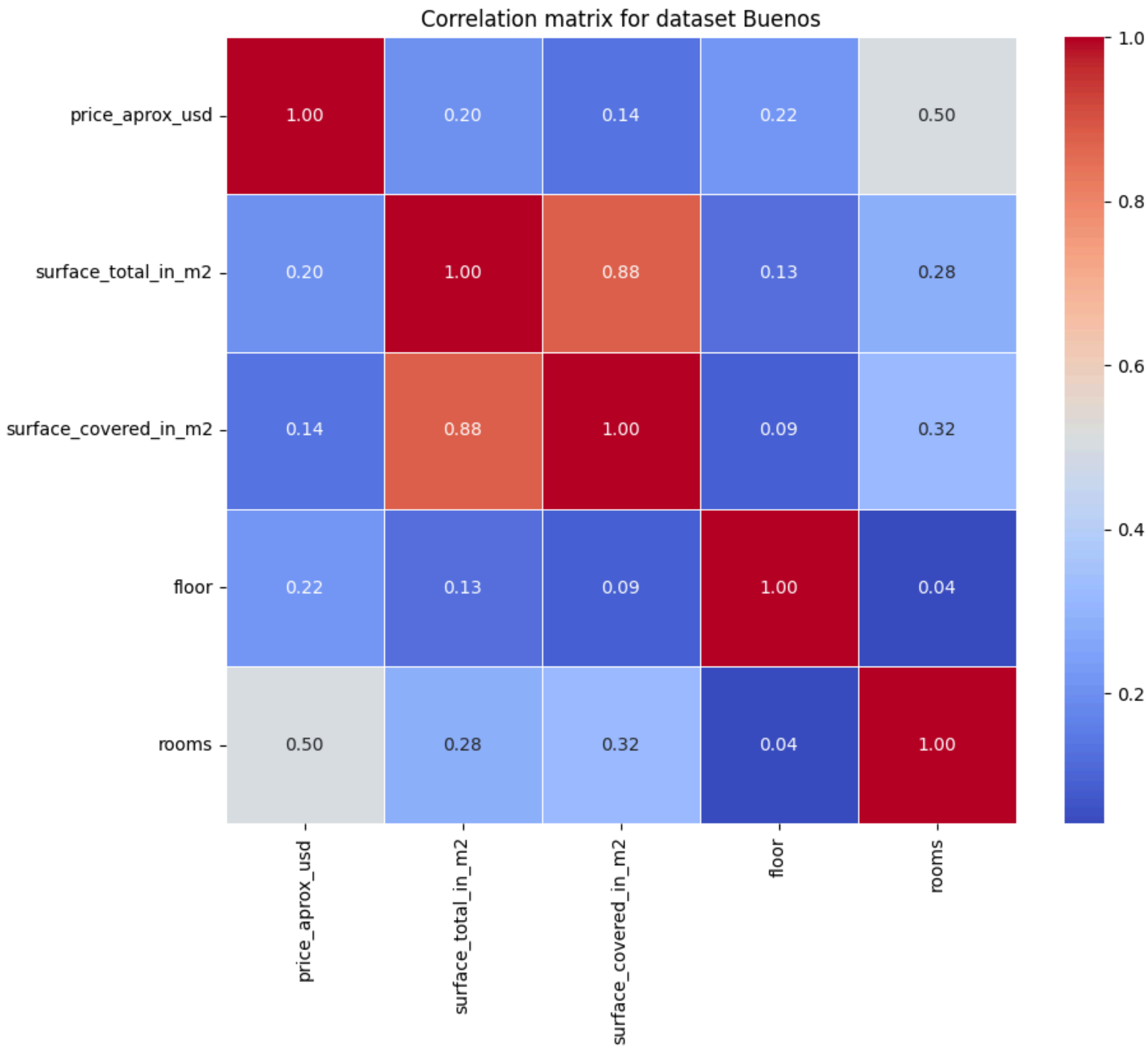
- surface_covered_in_m2: This typically refers to the area that is covered, such as the building itself.

Surface_total_in_m2 is sufficient to provide the required information about the surface of the property who's price we are trying to predict, however the ratio of the covered area to the not covered would provide some valuable information. Hence, a feature will be engineered that gives the ratio of the surface covered.

In [ ]:
```
#Extract the correlation of the features with the target
corr_with_target = corr_df['price_aprox_usd']
features_to_drop = corr_with_target[abs(corr_with_target) < 0.1].index
data_df.drop(columns=features_to_drop, inplace=True)
print(data_df.columns)
```

```
Index(['operation', 'property_type', 'place_with_parent_names', 'currency',
       'price_aprox_usd', 'surface_total_in_m2', 'surface_covered_in_m2',
       'floor', 'rooms', 'properati_url'],
      dtype='object')
```

In [ ]:
```
corr_df = plt_heat_map()
```




Because we are keeping both total surface and surface covered we can drop rooms.

In [ ]:
```
data_df.drop(columns=['rooms'], inplace=True)
```

In [ ]:
```
EDA_df = data_df
```

## Exploratory Data Analysis

view % of missing values again

In [ ]:
```
def view_missing_values():
    missing_values = EDA_df.isna().sum()*100/len(EDA_df)
    print(f"{missing_values.apply(lambda x: f"{x: .2f}%")}")
    return
```

In [ ]:
```
view_missing_values()
```

```
operation                   0.00%
property_type               0.00%
place_with_parent_names     0.00%
currency                   11.81%
price_aprox_usd            11.81%
surface_total_in_m2        30.91%
surface_covered_in_m2      15.55%
floor                      85.37%
properati_url               0.00%
dtype: object
```

In [ ]:
```
#Drop floor too many missing values
EDA_df.drop(columns=['floor', 'properati_url'], inplace=True)
view_missing_values()
```

```
operation                   0.00%
property_type               0.00%
place_with_parent_names     0.00%
currency                   11.81%
price_aprox_usd            11.81%
surface_total_in_m2        30.91%
surface_covered_in_m2      15.55%
dtype: object
```

## Categorical data

In [ ]:
```
#Cardinality
cat_features = EDA_df.select_dtypes('object')
print(cat_features.columns)
```

```
Index(['operation', 'property_type', 'place_with_parent_names', 'currency'], dtyp
e='object')
```

In [ ]:
```
#'place_with_parent_names'
print(EDA_df['place_with_parent_names'].head())
```

```
0              |Argentina|Capital Federal|Villa Crespo|
1    |Argentina|Bs.As. G.B.A. Zona Oeste|La Matanza...
2    |Argentina|Bs.As. G.B.A. Zona Oeste|Morón|Cast...
3    |Argentina|Bs.As. G.B.A. Zona Oeste|Tres de Fe...
4                 |Argentina|Capital Federal|Chacarita|
Name: place_with_parent_names, dtype: object
```

In [ ]:
```
#The column in this data is organised as country, state ...
#keep just the neighborhood
EDA_df['neigborhood'] = EDA_df['place_with_parent_names'].str.split('|', expand=
```

```
print(EDA_df['neigborhood'].head())
EDA_df.drop(columns=['place_with_parent_names'], inplace=True)
```

```
0       Villa Crespo
1          La Matanza
2               Morón
3    Tres de Febrero
4          Chacarita
Name: neigborhood, dtype: object
```

In [ ]:
```
#Cardinality
cat_features = EDA_df.select_dtypes('object').nunique()
print(cat_features)
```

```
operation         1
property_type     4
currency          2
neigborhood      88
dtype: int64
```

In [ ]:
```
#Drop features with a high and low cardinality
EDA_df.drop(columns=['operation', 'neigborhood'], inplace=True)
```

In [ ]:
```
view_missing_values()
```

```
property_type            0.00%
currency                11.81%
price_aprox_usd         11.81%
surface_total_in_m2     30.91%
surface_covered_in_m2   15.55%
dtype: object
```

## Fill Missing values

In [ ]:
```
#Fill in missinf values
#categorical data
EDA_df['currency'].fillna(EDA_df['currency'].mode().iloc[0], inplace=True)

#numeric data
EDA_df['price_aprox_usd'].fillna(EDA_df['price_aprox_usd'].mean(), inplace=True)
EDA_df['surface_total_in_m2'].fillna(EDA_df['surface_total_in_m2'].mean(),  inpl
EDA_df['surface_covered_in_m2'].fillna(EDA_df['surface_covered_in_m2'].mean(),
view_missing_values()
```

```
property_type           0.00%
currency                0.00%
price_aprox_usd         0.00%
surface_total_in_m2     0.00%
surface_covered_in_m2   0.00%
dtype: object
```

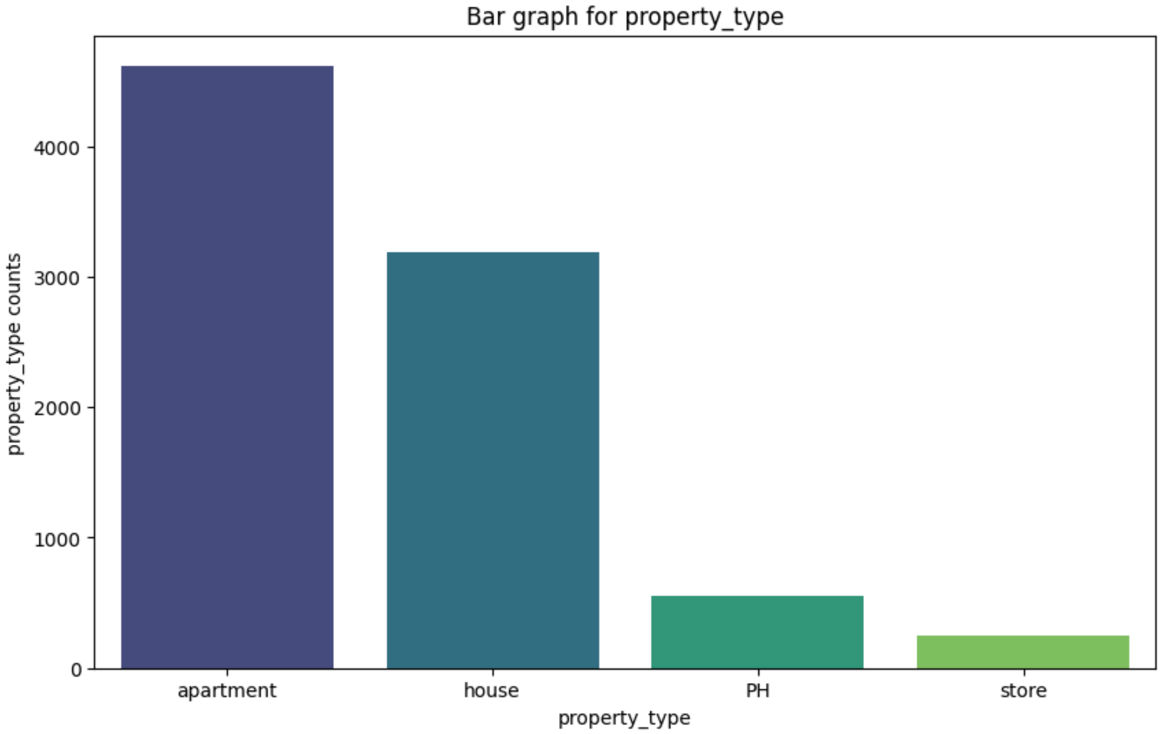
### EDA

In [ ]:
```
#select only categorical data
cat_columns = EDA_df.select_dtypes('object').columns
```

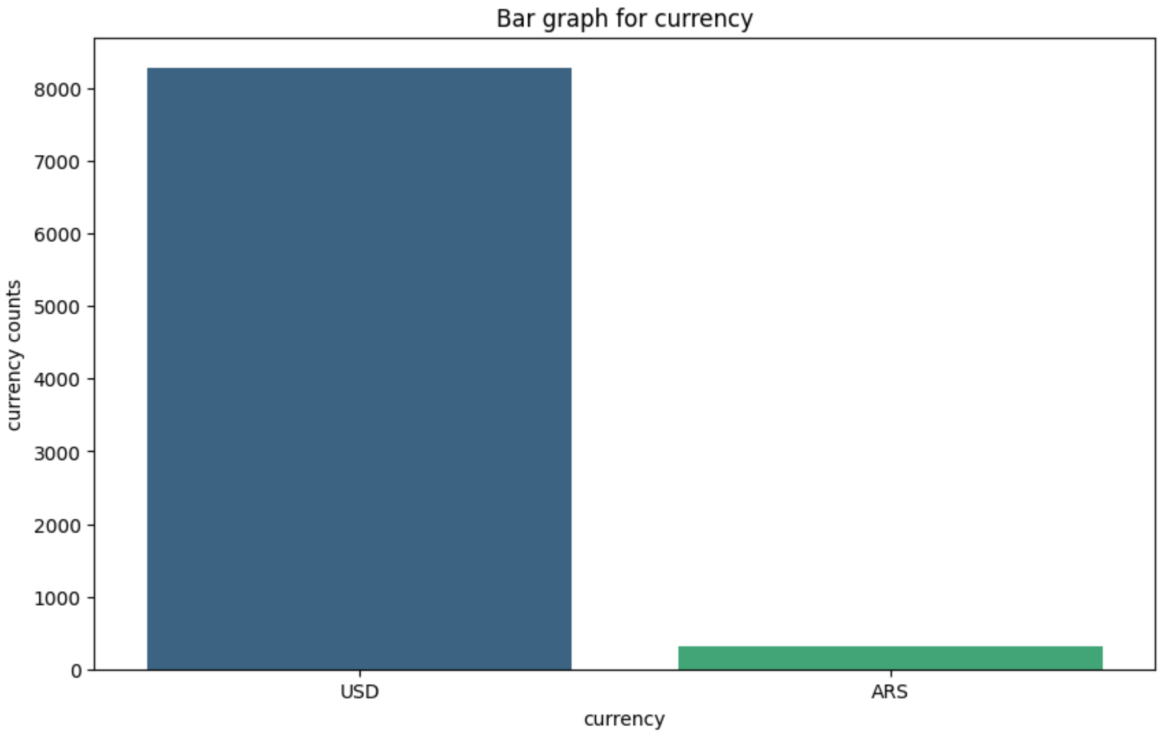
In [ ]:
```
def plot_bar(cat_columns, df):
    for column in cat_columns:
        cat_counts = df[column].value_counts().reset_index()
```

```
        cat_counts.columns = [column, f"{column} counts"]
        plt.figure(figsize=(10,6))
        sns.barplot(x = f"{column}", y = f"{column} counts",
        data = cat_counts, palette='viridis', hue=column, legend = False)
        plt.title(f"Bar graph for {column}")
        plt.show()

        #print countss for each variable
        percent_counts = df[column].value_counts()*100/len(df[column])
        print(f"Percentange of counts for each value in {column}")
        print(f"{percent_counts.apply(lambda x: f"{x:.2f}%")}")
    return
```

In [ ]:
```
plot_bar(cat_columns, EDA_df)
```



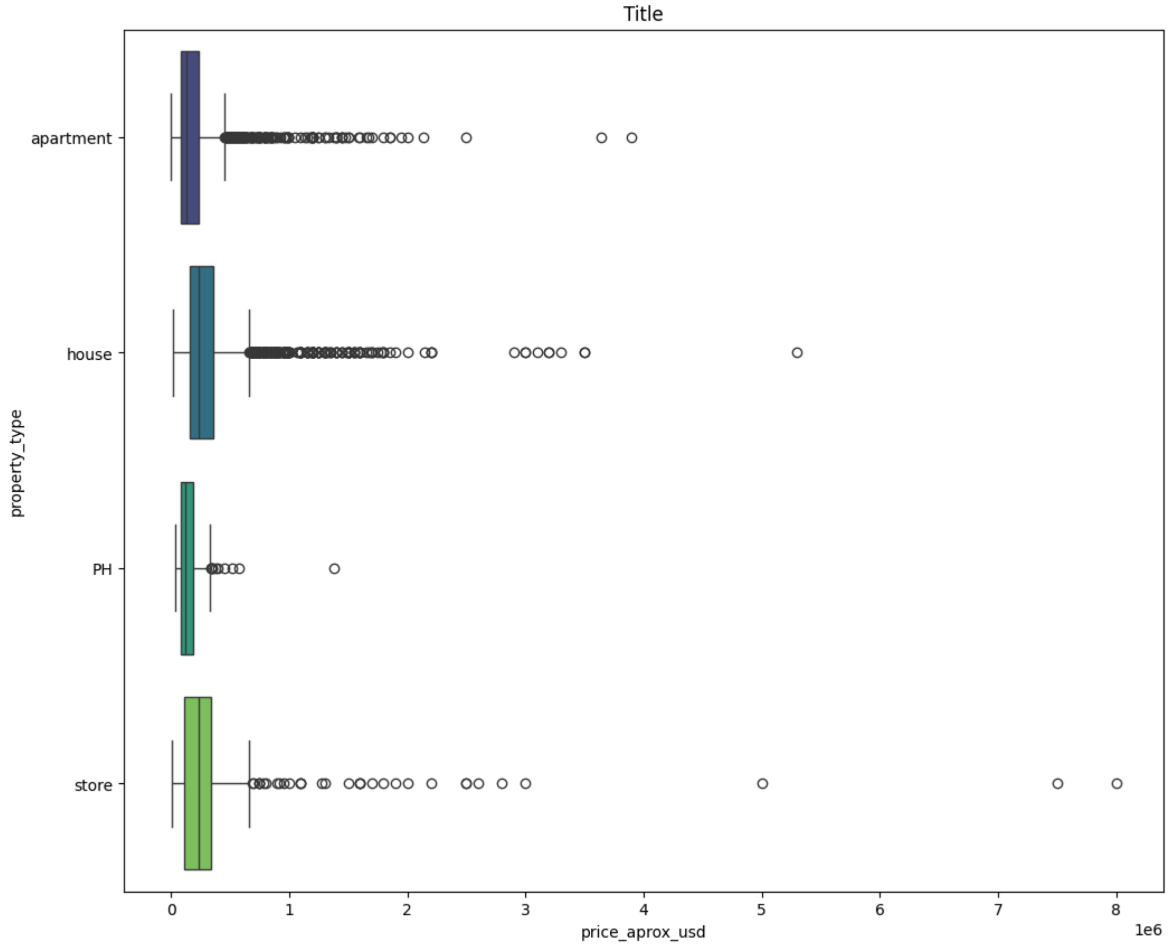

```
Percentange of counts for each value in property_type
property_type
apartment    53.65%
house        37.07%
PH            6.37%
store         2.92%
Name: count, dtype: object
```

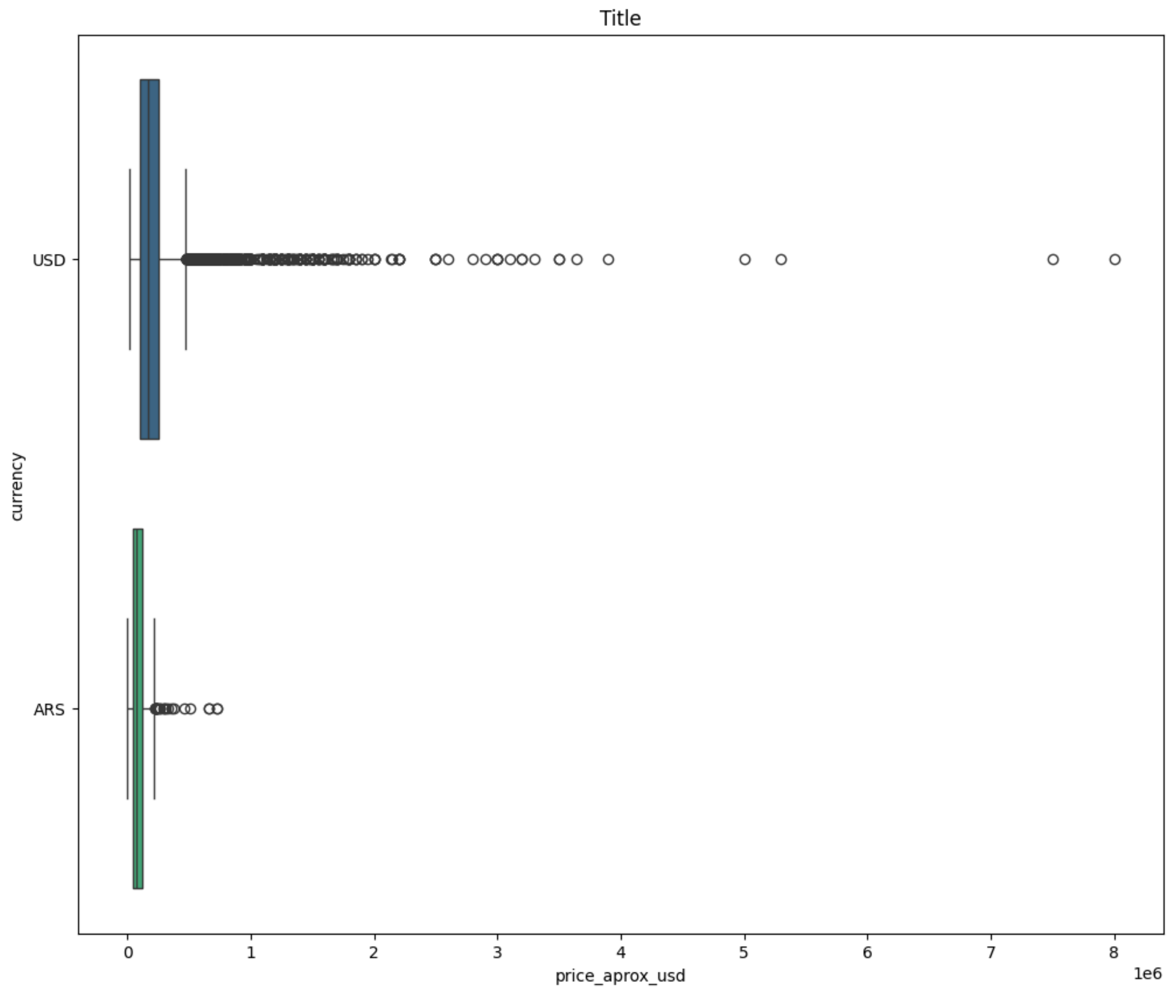



```
Percentange of counts for each value in currency
currency
USD    96.29%
ARS     3.71%
Name: count, dtype: object
```

In [ ]:
```
def plot_box(category_cols, df):
    for col in category_cols:
        plt.figure(figsize=(12,10))
        sns.boxplot(y = f'{col}', x = 'price_aprox_usd',
                    data = df, legend=False, hue = f'{col}',
                    palette='viridis')
        plt.title("Title")
        plt.xlabel('price_aprox_usd')
        plt.show()
    return
```

In [ ]:
```
plot_box(cat_columns, EDA_df)
```




In [ ]:
```
print(EDA_df.columns)
```

```
Index(['property_type', 'currency', 'price_aprox_usd', 'surface_total_in_m2',
       'surface_covered_in_m2'],
      dtype='object')
```

remove outliers

In [ ]:
```python
def calculate_outliers_percentage(group):
    Q1 = group.quantile(0.25)
    Q3 = group.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = group[(group < lower_bound) | (group > upper_bound)]
    return len(outliers) / len(group) * 100
```

In [ ]:
```python
#select only categorical data FIX!
cat_columns = EDA_df.select_dtypes('object').columns

#Dictionary variable to hold grouped data
grouped_data = {}
outlier_perc ={}
for col in cat_columns:
    grouped_data[col] = EDA_df.groupby(col)['price_aprox_usd'].describe()
    outlier_perc[col] = {}
    for group_name, group_data in EDA_df.groupby(col)['price_aprox_usd']:
        outlier_perc[col][group_name] = calculate_outliers_percentage(group_data

for col, data in grouped_data.items():
    print(f"\nStatistical analysis:\n{data}")
    print(f"\nPercentage of Outliers for {col}:\n")
    for group, perc in outlier_perc[col].items():
        print(f"{group}: {round(perc, 2)}%")
```

```
Statistical analysis:
                count           mean            std       min       25%  \
property_type
PH              548.0  139625.261616   89974.803347  35000.00   85000.0
apartment      4617.0  176802.644510  189373.123153      0.00   85000.0
house          3190.0  325249.885265  322228.103388  13811.60  160000.0
store           251.0  431601.263169  871678.027985   6576.95  110000.0

                         50%       75%        max
property_type
PH             119000.000000  184250.0  1380000.0
apartment      128000.000000  231944.0  3900000.0
house          236891.878238  360000.0  5300000.0
store          236891.878238  339500.0  8000000.0

Percentage of Outliers for property_type:

PH: 1.82%
apartment: 4.12%
house: 8.97%
store: 13.15%

Statistical analysis:
           count           mean            std      min        25%        50%  \
currency
ARS        319.0  102730.814514   96490.290450      0.0  50313.715   77608.09
USD       8287.0  242056.277820  298518.534584  15000.0  99000.000  171675.00

               75%         max
currency
ARS       118697.1   730772.98
USD       250000.0  8000000.00

Percentage of Outliers for currency:

ARS: 7.21%
USD: 8.78%
```

In [ ]:
```python
#print(EDA_df['currency'].dtype)
group_currency = EDA_df.groupby('currency')['price_aprox_usd'].mean()
Q1 = EDA_df.groupby('currency')['price_aprox_usd'].quantile(0.25)
Q3 = EDA_df.groupby('currency')['price_aprox_usd'].quantile(0.75)
IQR = Q3 - Q1

#------Outliers calculation-----------------------------------------
#extract unique values in the column
unique_values = EDA_df['currency'].unique()
outliers = []

for currency in unique_values:
    #data mask
    mask = EDA_df['currency'] == currency

    lower_bound = Q1[currency] -1.5*IQR[currency]
    upper_bound = Q1[currency] +1.5*IQR[currency]

    outliers_col = (EDA_df[mask &
                       ((EDA_df['price_aprox_usd'] < lower_bound)
                        |(EDA_df['price_aprox_usd'] > upper_bound))
                       ])
```

```
    outliers.append(outliers_col)

#concat all outliers into 1 df
outlier_df = pd.concat(outliers)
print(len(EDA_df))

#drop the outliers
EDA_df = EDA_df.drop(outlier_df.index)

print(len(EDA_df))
```

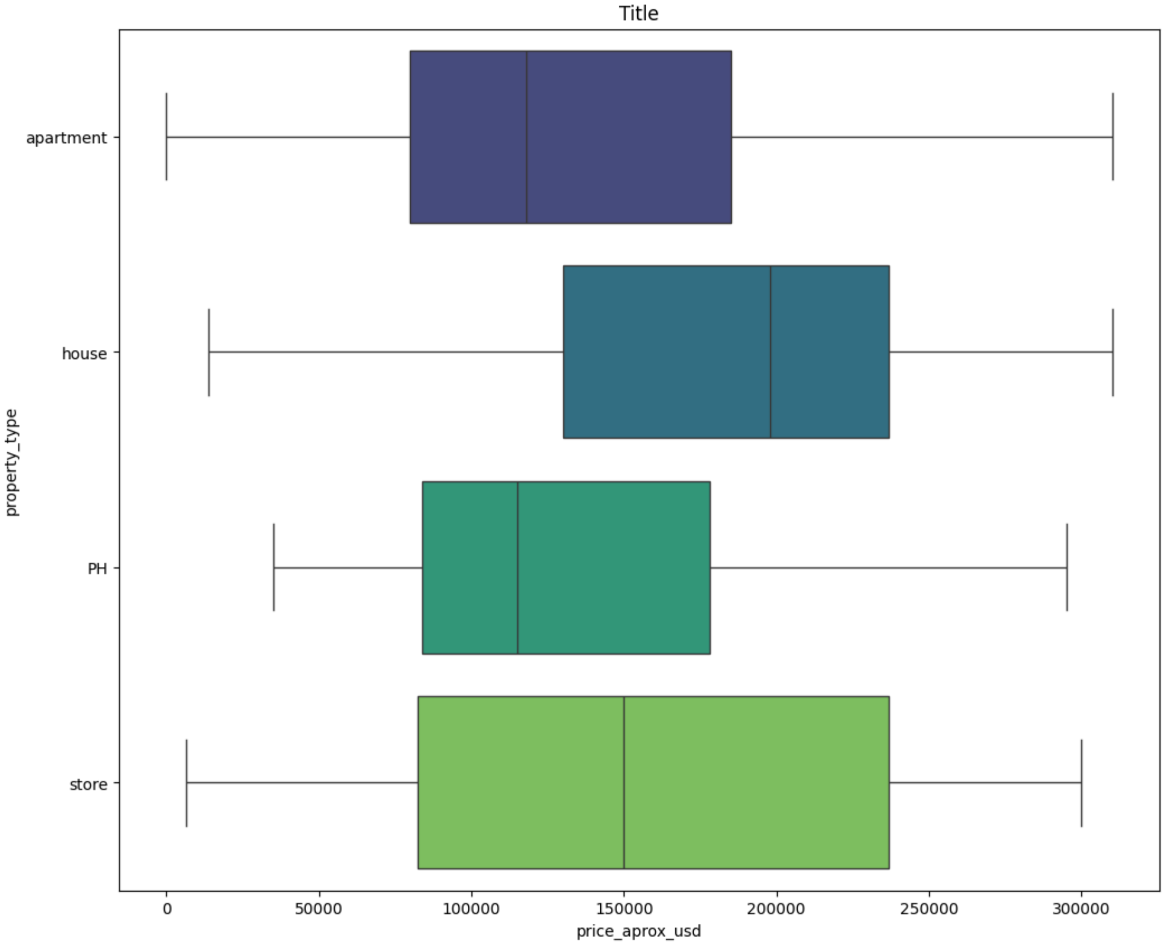
```
8606
7132

7132
```

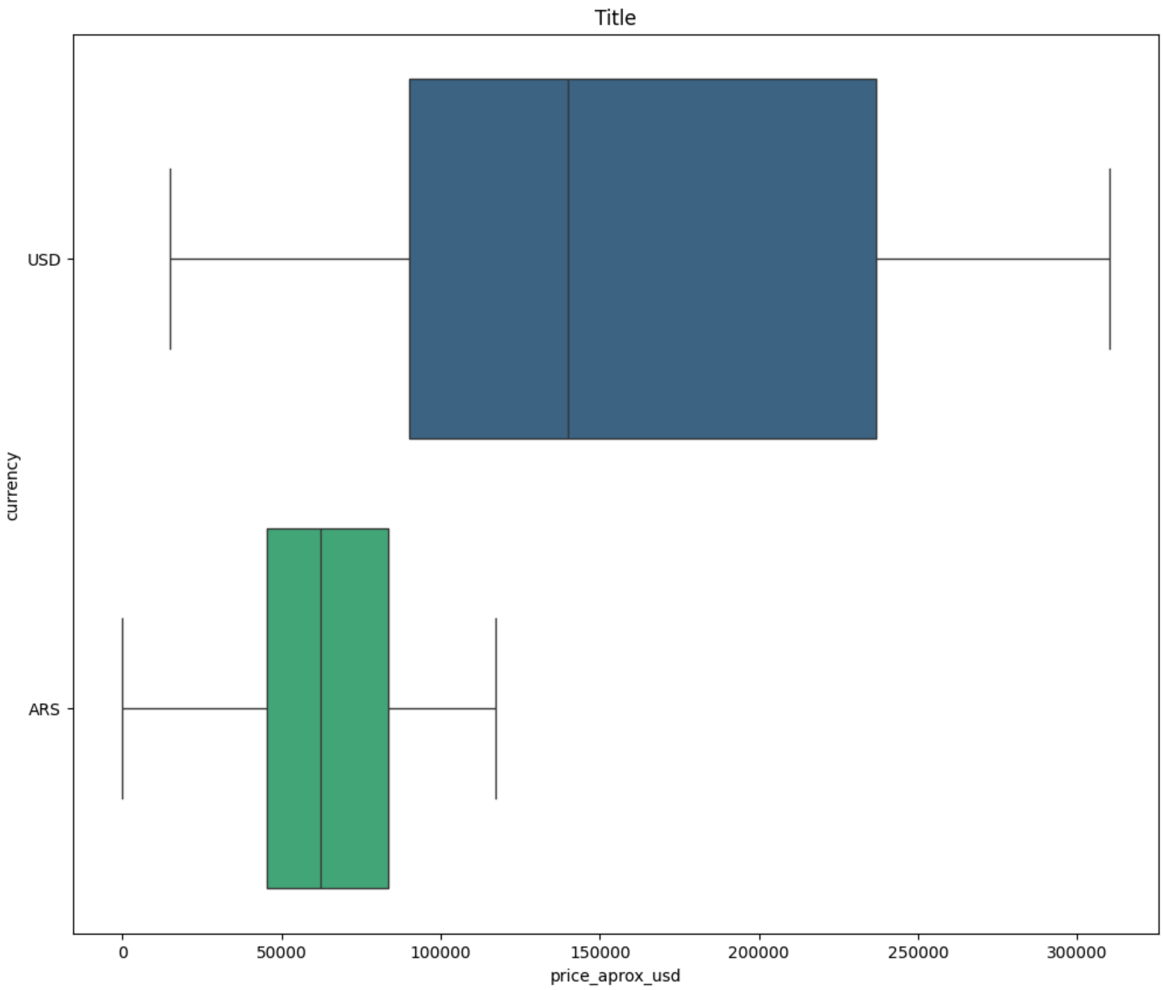
In [ ]:
```
#select only categorical data
cat_columns = EDA_df.select_dtypes('object').columns
```

In [ ]:
```
def remove_outliers(cols, df):
    for col in cols:
        Q1 = df.groupby(col)['price_aprox_usd'].quantile(0.25)
        Q3 = df.groupby(col)['price_aprox_usd'].quantile(0.75)
        IQR = Q3 - Q1

        #------Outliers calculation-----------------------------------------
        #extract unique values in the column
        unique_values = df[col].unique()
        outliers = []

        for currency in unique_values:
            #data mask
            mask = df[col] == currency

            lower_bound = Q1[currency] -1.5*IQR[currency]
            upper_bound = Q1[currency] +1.5*IQR[currency]

            outliers_col = (df[mask &
                              ((df['price_aprox_usd'] < lower_bound)
                                  |(df['price_aprox_usd'] > upper_bound))
                              ])
            outliers.append(outliers_col)

    #concat all outliers into 1 df
    outlier_df = pd.concat(outliers)
    print(len(df))

    #drop the outliers
    df = df.drop(outlier_df.index)

    print(len(df))

    return df
```

In [ ]:
```
data_df = remove_outliers(cat_columns, EDA_df)
plot_box(cat_columns, data_df)
```

```
7132
7009
```


Title
apartment
house
PH
store
property_type
0
50000
100000
150000
200000
250000
300000
price_aprox_usd


Title
USD
ARS
currency
0
50000
100000
150000
200000
250000
300000
price_aprox_usd

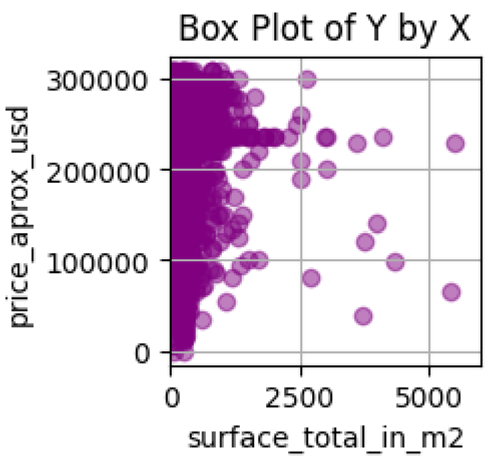
## Numeric data columms

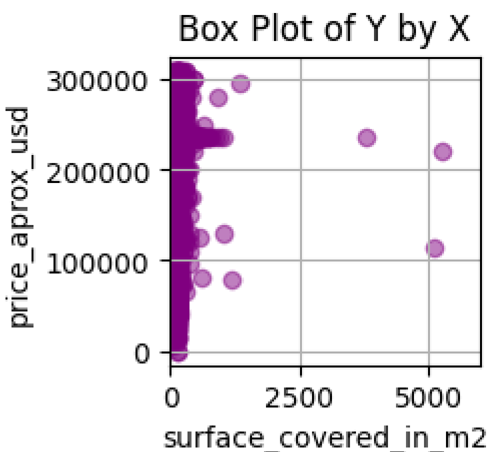
In [ ]:
```
num_cols = data_df.select_dtypes('number').columns.tolist()
num_cols.remove('price_aprox_usd')
print(num_cols)
```

```
['surface_total_in_m2', 'surface_covered_in_m2']
```

In [ ]:
```
def plot_scatter(num_cols, df):
    #define the input and target cols
    target = df['price_aprox_usd']

    for col in num_cols:
        x = df[col]
        plt.figure(figsize=(2, 2))
        plt.scatter(x, target, color='purple', alpha=0.5)
        plt.title('Box Plot of Y by X')
        plt.xlabel(f'{x.name}')
        plt.ylabel(f'{target.name}')
        plt.xlim(-1, 6000)
        plt.grid(True)
        plt.show()
    return
```

In [ ]:
```
plot_scatter(num_cols, data_df)
```




There isn't an intrinsic relationship between the two variables form the scatter plot. however majority of tthe surface area covered falls below 2500.

## Data Ttrainsformation

All categorical data is OHE except. Target is not encoded as it is numeric and continuous.

In [ ]:
```
cat_col_encode = data_df.select_dtypes('object').columns
ohe = OneHotEncoder(
    use_cat_names=True,
    cols=cat_col_encode
)

ohe_data = ohe.fit_transform(data_df)
print(ohe_data.columns)
```

```
Index(['property_type_apartment', 'property_type_house', 'property_type_PH',
       'property_type_store', 'currency_USD', 'currency_ARS',
       'price_aprox_usd', 'surface_total_in_m2', 'surface_covered_in_m2'],
      dtype='object')
```

## Train Regression Model:

In [ ]:
```
#Split data
y = ohe_data['price_aprox_usd']
X = ohe_data.drop(columns=['price_aprox_usd'])
#print(X.columns)

x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_st

print(f"train:\t{len(x_train)}\t{len(y_train)}")
print(f"train:\t{len(x_test)}\t{len(y_test)}")
```

```
train:  5607    5607
train:  1402    1402
```

### Baseline

In [ ]:
```
y_mean = (y_train.mean())
y_predic_baseline = [y_mean]*len(y_train)
baseline_MAE = mean_absolute_error(y_train, y_predic_baseline)
print(f"Mean aparatment price: ${round(y_mean,2)}")
print(f"Baseline MAE aparatment price: ${round(baseline_MAE,2)}")
```

```
Mean aparatment price: $151382.5
Baseline MAE aparatment price: $63205.07
```

### Pipeline for Linear Regression Model

In [ ]:
```
ln_model = make_pipeline(
    OneHotEncoder(use_cat_names=True),
    LinearRegression()
)

ln_model.fit(x_train, y_train)
```

```
Warning: No categorical columns found. Calling 'transform' will only return input
data.
```

Out[ ]:

```
Pipeline
  OneHotEncoder
  LinearRegression
```

## Evaluate Linear Regression Model

In [ ]:

```
y_pred_training = ln_model.predict(x_train)
y_predict_test = ln_model.predict(x_test)

#Mean Absolute error
MAE_train = mean_absolute_error(y_train, y_pred_training)
MAE_test = mean_absolute_error(y_test, y_predict_test)
print(f"Training MAE: {round(MAE_train,2)}")
print(f"Testing MAE: {round(MAE_test,2)}")
```

```
Training MAE: 56753.79
Testing MAE: 55860.72
```

In [ ]:

```
#View % of different from baseline:
train_perc_diff_baseline = (MAE_train/baseline_MAE)*100
test_perc_diff_baseline = (MAE_test/baseline_MAE)*100
print(f"The percentage difference train MAE from the baseline is: {round(train_p
print(f"The percentage difference test MAE from the baseline is: {round(test_per
```

```
The percentage difference train MAE from the baseline is: 89.79%
The percentage difference test MAE from the baseline is: 88.38%
```

Metric Intrepretation: (use next 3!)

- There is a higher MAE on Training data than on Testing data.
- This would generally be a sign of overfitting.
- however Viewing the slight different in both values relative to the baseline, tells us the model is predicting well for both. with only a difference of 1%.
- Model is predicting well on both training data and testing data.

Overfitting Assessment:

Generally, if the MAE on the training data is significantly lower than the MAE on the test data, it could be an indication of overfitting (the model performs well on training data but poorly on unseen data). In your case, the training MAE is slightly higher than the testing MAE. This might not be a typical sign of overfitting.

Percentage Difference:

The percentage difference between the training MAE and the baseline MAE is 89.79%, and for the testing MAE, it is 88.38%. Both are close to each other, suggesting that the model has similar performance on both the training and test sets relative to the baseline. The slight difference (1.41% = 89.79% - 88.38%) indicates that the model is generalizing well and not overfitting.

Model Performance:

Given that both the training and testing MAE are relatively close and much lower compared to the baseline, it suggests that the model is performing well in predicting the target variable for both the training and testing datasets.

### Save linear Regression Model

In [ ]:
```
joblib.dump(ln_model, './artifacts/linear_model.pk1')
```

Out[ ]:
```
['./artifacts/linear_model.pk1']
```

## Logistic regression

### Logistic regression Pipeline

In [ ]:
```
rf_model = make_pipeline(
    OneHotEncoder(use_cat_names=True),
    RandomForestRegressor(random_state=42)
)

rf_model.fit(x_train, y_train)
```

```
Warning: No categorical columns found. Calling 'transform' will only return input data.
```

Out[ ]:
```
Pipeline
  OneHotEncoder
  RandomForestRegressor
```

In [ ]:
```
y_pred_training = rf_model.predict(x_train)
y_predict_test = rf_model.predict(x_test)

#Mean Absolute error
MAE_train = mean_absolute_error(y_train, y_pred_training)
MAE_test = mean_absolute_error(y_test, y_predict_test)
print(f"Training MAE: {round(MAE_train,2)}")
print(f"Testing MAE: {round(MAE_test,2)}")
```

```
Training MAE: 31409.5
Testing MAE: 44603.98
```

In [ ]:
```
print(f"The percentage difference before CV between the MAEs:")
print(f"{round(100-(MAE_train*100/MAE_test),2)}%")
```

```
The percentage difference before CV between the MAEs:
29.58%
```

### Cross Validation

In [ ]:
```
params= {
    "randomforestregressor__n_estimators": range(25,100,25),
```

```python
    "randomforestregressor__max_depth": range(10,50,10)
}
```

In [ ]:

```python
rf_model2 = GridSearchCV(
    rf_model,
    param_grid=params,
    cv=5,#folds
    n_jobs=1,
    verbose=1
)

#fit model
rf_model2.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

Out[ ]:

```
▸ GridSearchCV
▸ estimator: Pipeline
    ▸ OneHotEncoder
    ▸ RandomForestRegressor
```

In [ ]:

```python
y_pred_training = rf_model2.predict(x_train)
y_predict_test = rf_model2.predict(x_test)

#Mean Absolute error
MAE_train = mean_absolute_error(y_train, y_pred_training)
MAE_test = mean_absolute_error(y_test, y_predict_test)

#Calculate r2_score()
r2_score_train = r2_score(y_train, y_pred_training)
r2_score_trest = r2_score(y_test, y_predict_test)

print(f"Training MAE: {round(MAE_train,2)}")
print(f"Testing MAE: {round(MAE_test,2)}")
print(f"\n\nr2_score Training: {round(r2_score_train,2)}")
print(f"r2_score Testing: {round(r2_score_trest,2)}")
```

```
Training MAE: 38723.81
Testing MAE: 42439.92


r2_score Training: 0.52
r2_score Testing: 0.41
```

R2_Score interpretation:

- R2 score training: this indicates how well the model fits the training data.
- R2 score test: this indicates how well the model generalises to unseen data.
- ADMIN:
  - The training $R^2$ score indicates how well the model fits the training data.
  - A high training $R^2$ score (close to 1) suggests that the model is able to explain a large portion of the variability in the target variable (y) using the features it was trained on.

- However, an excessively high training $R^2$ score (close to 1) can also indicate potential overfitting, where the model has learned noise and specifics of the training data rather than the underlying pattern.
- In your example, a training $R^2$ score of 0.85 means that 85% of the variability in the training data's target variable (y) is explained by the model.

-MORE

- $R^2$ (coefficient of determination) measures the proportion of the variance in the dependent variable (target) that is predictable from the independent variables (features).
- Training $R^2$ Score of 0.52 means that 52% of the variability in the training data's target variable (y) is explained by the model.
- Testing $R^2$ Score of 0.41 means that 41% of the variability in the testing data's target variable (y) is explained by the model.

- Your model shows moderate performance based on the metrics provided. It captures a reasonable amount of variability in the target variable (y), as indicated by the $R^2$ scores.
- There is some indication of overfitting, as seen in the difference between the training and testing $R^2$ scores and potentially in the higher testing MAE.
- Further model tuning, feature selection, or regularization techniques could potentially improve generalization to unseen data and reduce overfitting.

### R-squared mean error

In [ ]:
```
#r2_score, root_mean_squared_error
#Calculate r^2()
r2_square_train = root_mean_squared_error(y_train, y_pred_training)
r2_square_trest = root_mean_squared_error(y_test, y_predict_test)

print(f"\n\nr2_mean error Training: {round(r2_square_train,2)}")
print(f"r2_mean error Testing: {round(r2_square_trest,2)}")
print(f"%diff:\t{100 - round((r2_square_train/r2_square_trest)*100,2)}%")
```

```
r2_mean error Training: 50133.91
r2_mean error Testing: 54209.26
%diff:  7.519999999999996%
```

the RMSE provides the magnitude of the prediction of error.

- the magnitude of the prediction of error is more for testing data than training data. this is because the model is unfamilair with the training data.
- The percentage difference between these two values is 7.52%, which is not much of a difference in the magnitude of the model

Conclusion: Your model shows reasonable performance based on the MAE metrics provided. The higher MAE on testing data compared to training data suggests some level of overfitting or lack of generalization to unseen data. A 7.52% difference in MAE between training and testing datasets indicates that while there is a difference in prediction error magnitudes, it is not significantly large.

In [ ]:
```
#save model
joblib.dump(rf_model2, './artifacts/rf_model.pk2')
```

Out[ ]:
```
['./artifacts/rf_model.pk2']
```

In [ ]:
```
# Get the best estimator from GridSearchCV
best_rf_model = rf_model2.best_estimator_

# Get coefficients of features
coefficients = best_rf_model.named_steps['randomforestregressor'].feature_import

# Get feature names from onehotencoder
features = best_rf_model.named_steps['onehotencoder'].get_feature_names_out()

# Create a Series of features
feat_imp = pd.Series(data=coefficients, index=features)

# Plot feature importance
plot_feat_imp = feat_imp.sort_values(ascending=True).tail(6)
plot_feat_imp.plot(kind="barh",color=sns.color_palette('viridis', len(plot_feat_
plt.title("Feature Importance")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```