# Real Estate Regression

In [1]:

```
import pandas as pd
import numpy as np
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sb
import folium
import joblib
import warnings

from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler


%matplotlib inline
sb.set_theme(style = 'white')
warnings.filterwarnings('ignore')
```

In [2]:

```
# plot functions
# User defined histogram plot function
def hist_plot(data, x_arg, title, x_label, y_label, bin_size, kde):
    sb.histplot(data = data, x = x_arg, bins = bin_size, kde = kde)
    plt.axvline(data[x_arg].mean(), color = 'red', linestyle = '-', linewidth = 2, label = 'Mean')
    plt.axvline(data[x_arg].median(), color = 'cyan', linestyle = ':', linewidth = 2, label = 'Median')
    plt.title(title, size = 15, weight = 'bold')
    plt.xlabel(x_label, size = 15, weight = 'bold')
    plt.ylabel(y_label, size = 15, weight = 'bold')
    plt.legend()

# User defined box plot function
def box_plot(data, x_arg, y_arg, title, x_label):
    ax = sb.boxplot(data = data, x = x_arg, y = y_arg, color = 'white')#sb.color_palette()[0])
    plt.title(title, size = 15, weight = 'bold')
    plt.xlabel(x_label, size = 15, weight = 'bold')
    plt.setp(ax.lines, color = 'black')

def scatter_plot(data, x_arg, y_arg, x_label, y_label, title):
    sb.scatterplot(data = data, x = x_arg, y = y_arg)
    plt.title(title, size = 12, weight = 'bold')
    plt.xlabel(x_label, size = 10, weight = 'bold')
    plt.ylabel(y_label, size = 10, weight = 'bold')
    plt.show()
```

In [3]:

```
re_data = pd.read_csv('real_estate.csv')
re_data.head()
```

Out[3]:

| | transaction_date | house_age | transit_distance | local_convenience_stores | latitude | longitude | price_per_unit |
|---|---|---|---|---|---|---|---|
| **0** | 2012.917 | 32.0 | 84.87882 | 10 | 24.98298 | 121.54024 | 37.9 |
| **1** | 2012.917 | 19.5 | 306.59470 | 9 | 24.98034 | 121.53951 | 42.2 |
| **2** | 2013.583 | 13.3 | 561.98450 | 5 | 24.98746 | 121.54391 | 47.3 |
| **3** | 2013.500 | 13.3 | 561.98450 | 5 | 24.98746 | 121.54391 | 54.8 |
| **4** | 2012.833 | 5.0 | 390.56840 | 5 | 24.97937 | 121.54245 | 43.1 |

## Exploratory Analysis

In [4]:

```
re_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 414 entries, 0 to 413
Data columns (total 7 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   transaction_date          414 non-null    float64
 1   house_age                 414 non-null    float64
 2   transit_distance          414 non-null    float64
 3   local_convenience_stores  414 non-null    int64  
 4   latitude                  414 non-null    float64
 5   longitude                 414 non-null    float64
 6   price_per_unit            414 non-null    float64
dtypes: float64(6), int64(1)
memory usage: 22.8 KB
```

Form the results, the dataset is complete and their are no missing values.

## Estate location

In [5]:

```

fig = px.scatter_mapbox(re_data,
                        lat = 'latitude',
                        lon = 'longitude',
                        color = 'price_per_unit',
                        width = 1000,
                        height = 800,
                        hover_data = ['price_per_unit'],
                        zoom = 12,
                        title = 'Estate Locations')
# stamen-terrain, open-street-map, carto-positron
fig.update_layout(mapbox_style = 'carto-positron')
fig.show()
```

In [6]:

```python
# Create 3D scatter plot
fig = px.scatter_3d(re_data,
                    x = 'latitude',
                    y = 'longitude',
                    z = 'price_per_unit',
                    width = 800,
                    height = 600,
                    title = '3D view of Estate Locations')
fig.update_traces(marker = {'size': 4, 'line': {'width': 2, 'color': 'DarkSlateGrey'}},
                  selector = {'mode': 'markers'})
fig.show()
```

In [7]:

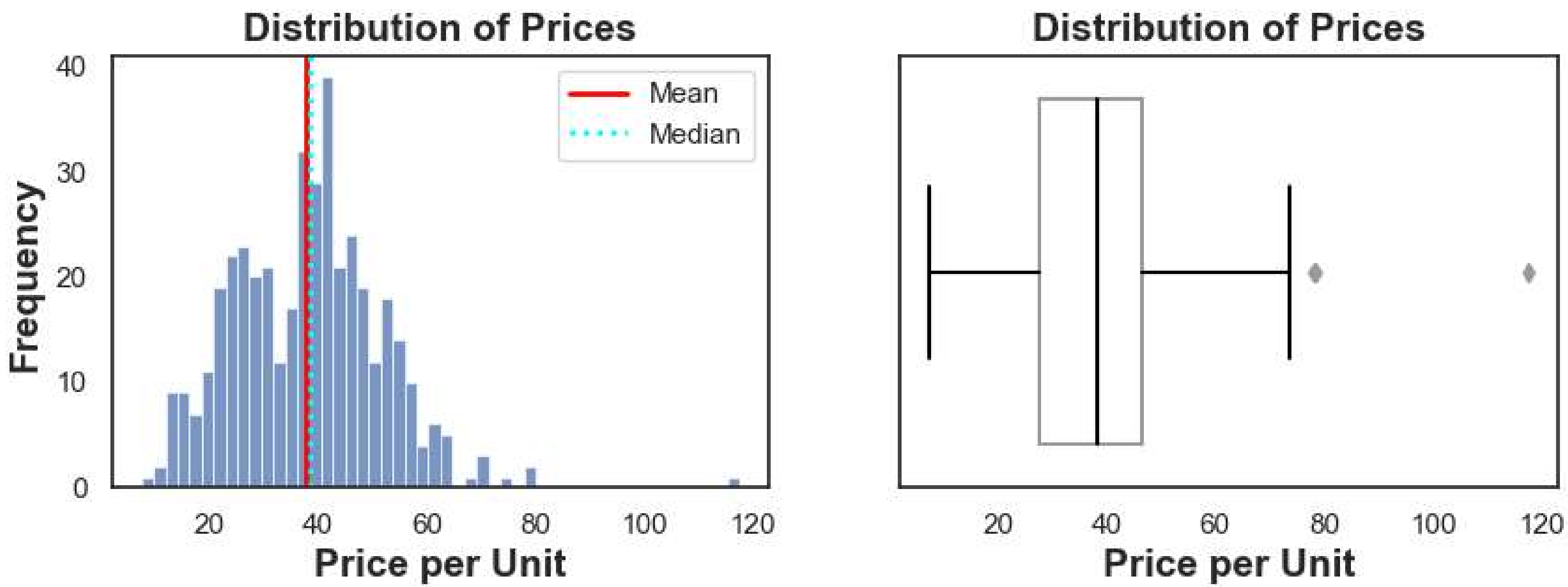
```
# center = [24.98298, 121.54024]
# estate_locations = folium.Map(location = center, zoom_start = 13)
# for i, j in re_data.iterrows():
#     location = [j['latitude'], j['longitude']]
#     folium.CircleMarker(location, radius = 2).add_to(estate_locations)
# estate_locations
```

## Price distribution

In [8]:

```python
display(re_data[['price_per_unit']].describe().transpose())
plt.figure(figsize = [10, 3])
plt.subplot(1, 2, 1)
hist_plot(re_data, 'price_per_unit', 'Distribution of Prices', 'Price per Unit', 'Frequency', 50, False)
plt.subplot(1, 2, 2)
box_plot(re_data, 'price_per_unit', None, 'Distribution of Prices', 'Price per Unit')
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **price_per_unit** | 414.0 | 37.980193 | 13.606488 | 7.6 | 27.7 | 38.45 | 46.6 | 117.5 |




The summary statistics table, histogram and the boxplot, it is evident there are outlier values in the price and these needs to be sieved out.

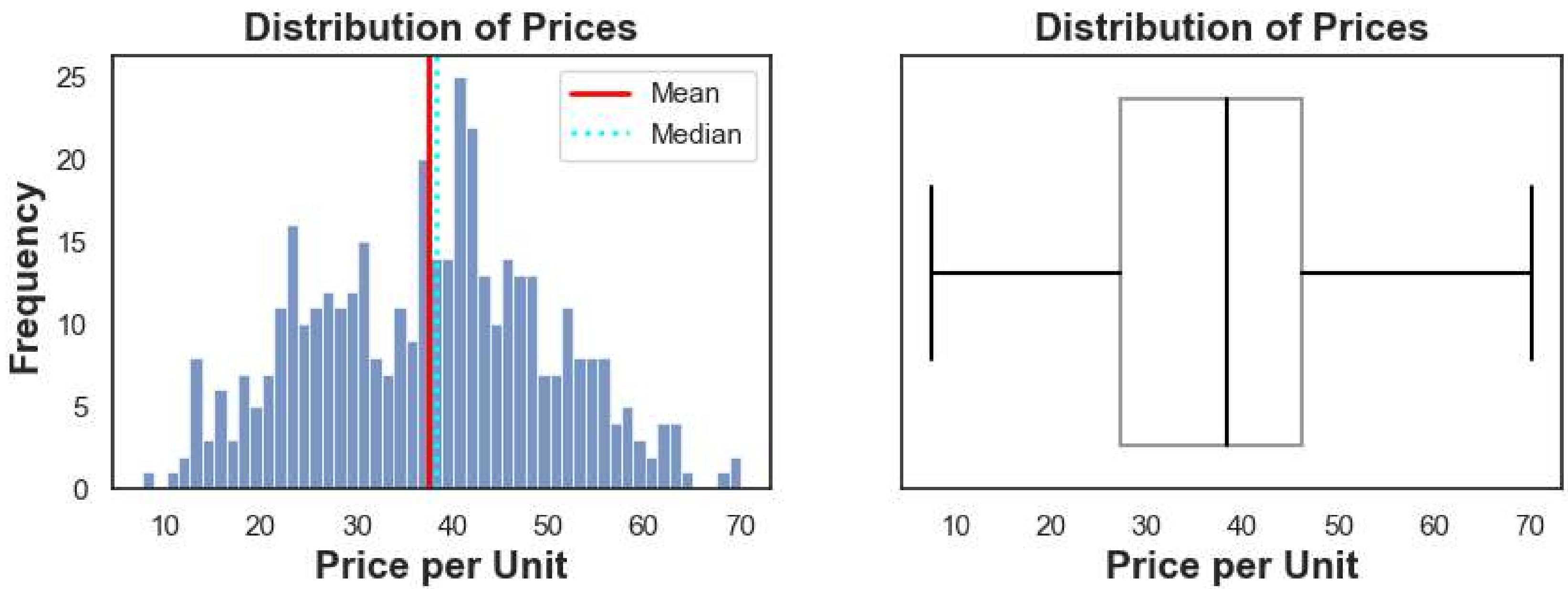
## Removing outlier values

The lower bound will be defined at 0%, and the upper bound at 99%.

In [9]:

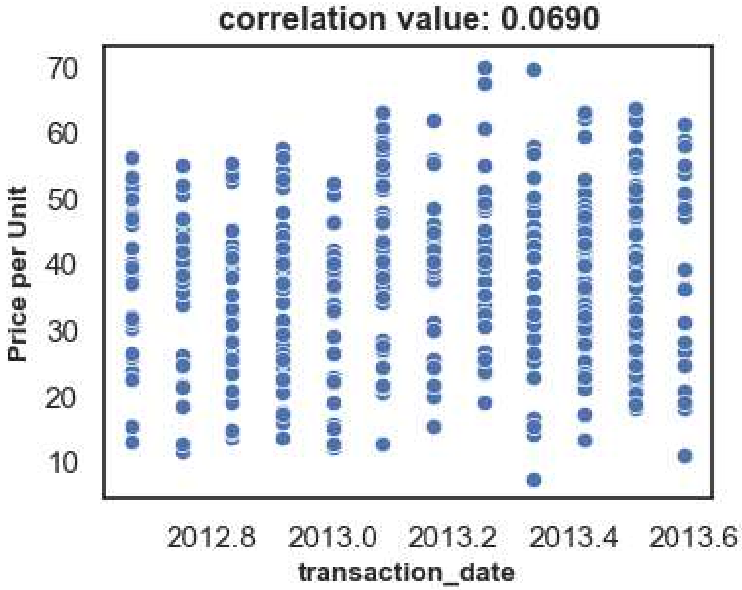
```
lower_bound = re_data['price_per_unit'].quantile(0)
upper_bound = re_data['price_per_unit'].quantile(0.99)
mark_bound_price = re_data['price_per_unit'].between(lower_bound, upper_bound)
re_data = re_data[mark_bound_price].reset_index().drop(columns = {'index'})

display(re_data[['price_per_unit']].describe().transpose())
plt.figure(figsize = [10, 3])
plt.subplot(1, 2, 1)
hist_plot(re_data, 'price_per_unit', 'Distribution of Prices', 'Price per Unit', 'Frequency', 50, False)
plt.subplot(1, 2, 2)
box_plot(re_data, 'price_per_unit', None, 'Distribution of Prices', 'Price per Unit')
```

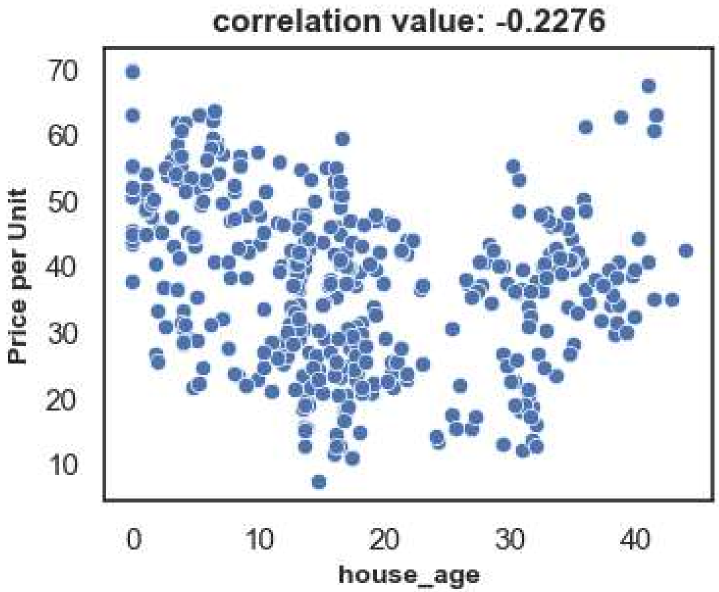
| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **price_per_unit** | 409.0 | 37.421516 | 12.565902 | 7.6 | 27.3 | 38.3 | 46.1 | 70.1 |

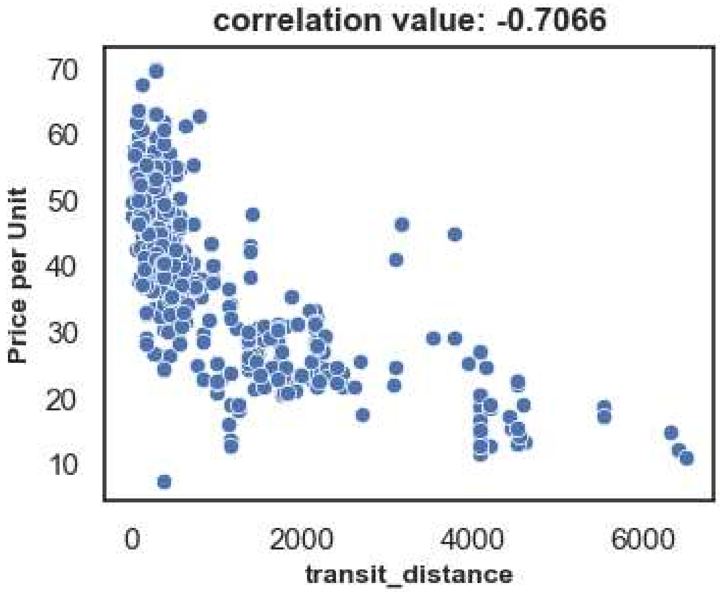


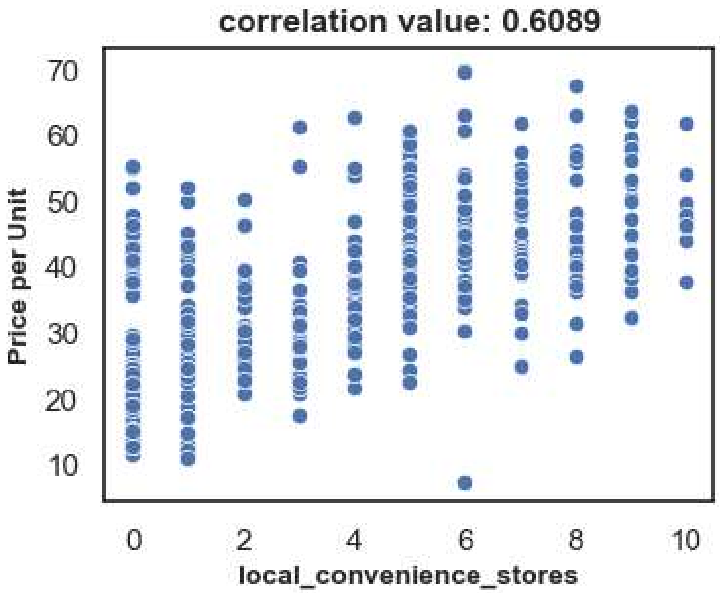
### Correlation between price and other features in the table

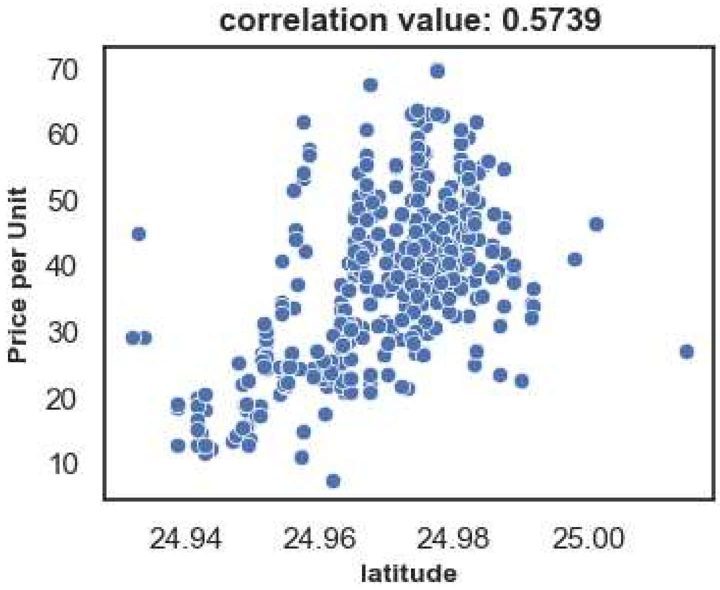
In [10]:
```
for col in re_data.columns[:-1]:
    plt.figure(figsize = [4, 3])
    correlation = re_data['price_per_unit'].corr(re_data[col])
    scatter_plot(re_data, col, 'price_per_unit', col, 'Price per Unit', f'correlation value: {correlation:.4f}')
```



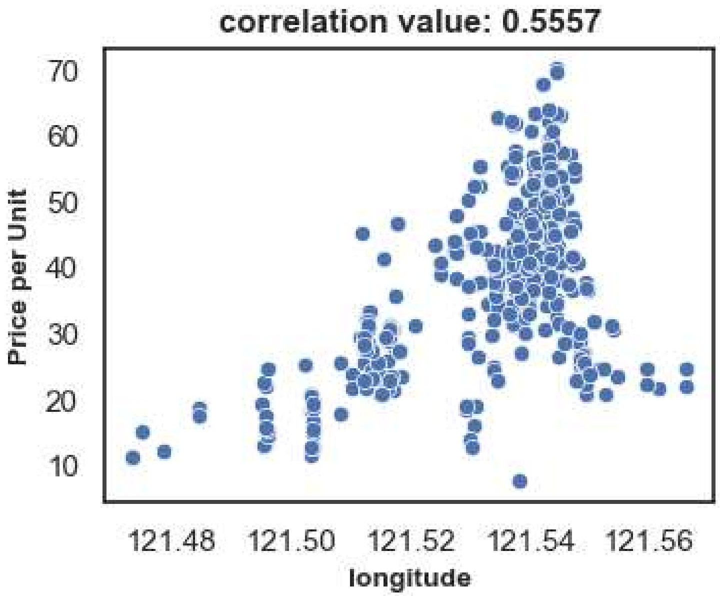




correlation value: -0.7066
Price per Unit
70
60
50
40
30
20
10
0
2000
4000
6000
transit_distance


correlation value: 0.6089
Price per Unit
70
60
50
40
30
20
10
0
2
4
6
8
10
local_convenience_stores



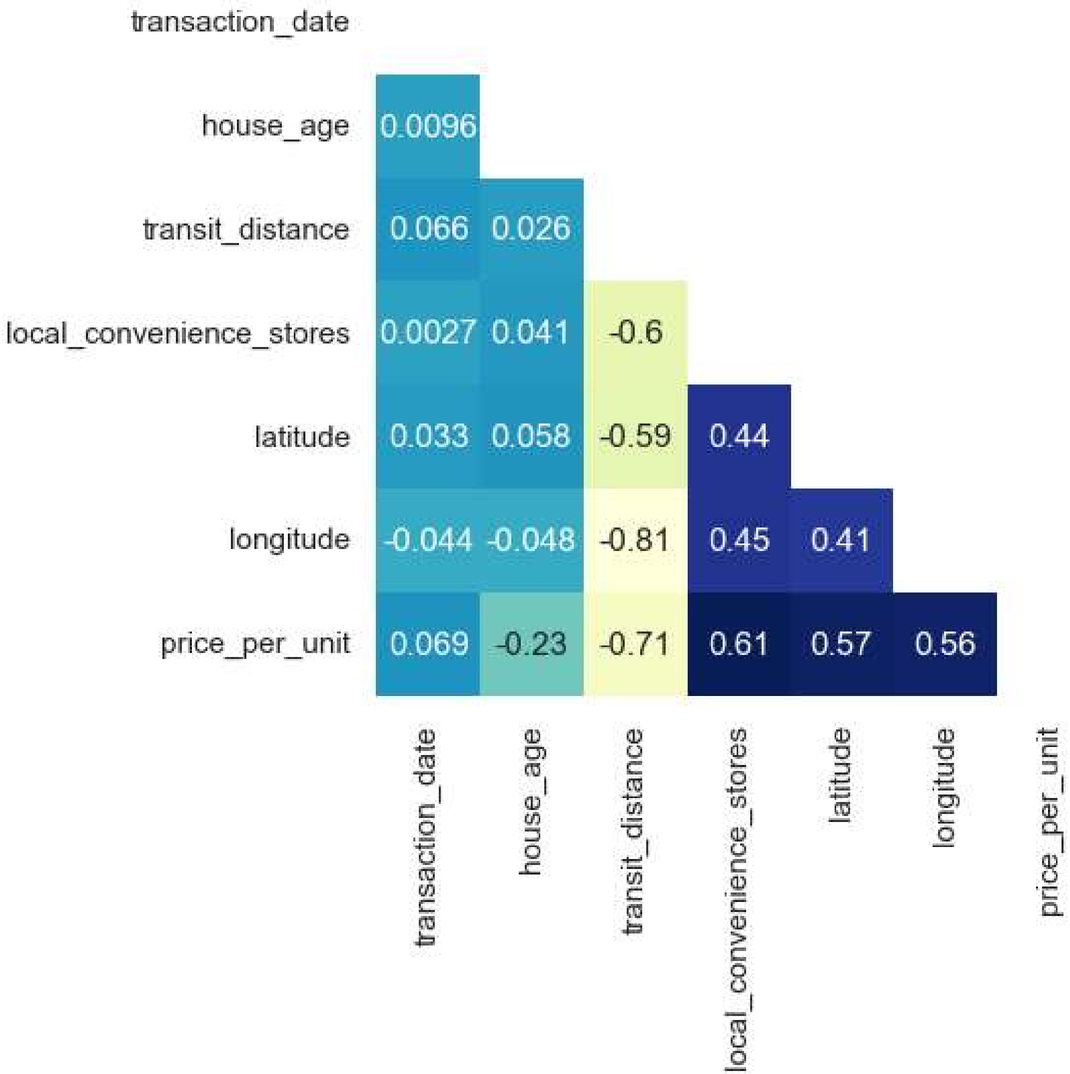
correlation value: 0.5739
Price per Unit
70
60
50
40
30
20
10
24.94
24.96
24.98
25.00
latitude

correlation value: 0.5557
Price per Unit
70
60
50
40
30
20
10
121.48
121.50
121.52
121.54
121.56
longitude

In [11]:

```
mask = np.triu(np.ones_like(re_data.corr()))
sb.heatmap(re_data.corr(), cmap = 'YlGnBu', annot = True, square = True, mask = mask, cbar = False);
```




The correlation values and the scatter plots above gives a clear evidence that `house age` , `transit distance` , `local convenience stores` , `latitude` and `longitude` all affect the price. `Transaction date` will be ignored since it has very negligible correlation with the price.

Checking the cardinality of local convenience stores:

In [12]:
```
print(f"Local convenience stores have {re_data['local_convenience_stores'].nunique()} unique values as listed below:")
display(re_data[['local_convenience_stores']].value_counts().sort_values())
```

```
Local convenience stores have 11 unique values as listed below:

local_convenience_stores
10                          10
9                           23
2                           24
8                           30
4                           31
7                           31
6                           35
1                           45
3                           46
0                           67
5                           67
dtype: int64
```

## Separate labels, features, split and normalize data

In [13]:
```
# Separate features and label
target_vector = re_data.columns[-1]
features = re_data.columns[1:-1]
X, y = re_data[features].values, re_data[target_vector].values

# split data: 70% for training and 30% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 0)
print(f'Training set: {X_train.shape[0]}\nTest set:{X_test.shape[0]}')
```

```
Training set: 286
Test set:123
```

## Build regression model

In [14]:
```
# Transform numeric columns
transformer = Pipeline(steps = [('scaler', StandardScaler())])
#numeric_features = ['house_age', 'transit_distance', 'latitude', 'longitude']
numeric_features = [0, 1, 2, 3, 4]
preprocess_data = ColumnTransformer(transformers = [('num', transformer, numeric_features)])
```

In [15]:
```
# Create a pipeline for Linear regression
model_reg = make_pipeline(transformer, preprocess_data, LinearRegression())
model_reg.fit(X_train, y_train)
```

Out[15]:
```
Pipeline(steps=[('pipeline', Pipeline(steps=[('scaler', StandardScaler())])),
                ('columntransformer',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('scaler',
                                                                   StandardScaler())]),
                                                  [0, 1, 2, 3, 4])])),
                ('linearregression', LinearRegression())])
```

In [16]:
```
# Pipeline form Ridge model
model_ridge = make_pipeline(transformer, preprocess_data, Ridge())
model_ridge.fit(X_train, y_train)
```

Out[16]:
```
Pipeline(steps=[('pipeline', Pipeline(steps=[('scaler', StandardScaler())])),
                ('columntransformer',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('scaler',
                                                                   StandardScaler())]),
                                                  [0, 1, 2, 3, 4])])),
                ('ridge', Ridge())])
```

In [17]:
```python
# Pipeline for Random forest regressor
model_forest = make_pipeline(transformer, preprocess_data, RandomForestRegressor())
model_forest.fit(X_train, y_train)
```

Out[17]:
```
Pipeline(steps=[('pipeline', Pipeline(steps=[('scaler', StandardScaler())])),
                ('columntransformer',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('scaler',
                                                                   StandardScaler())]),
                                                  [0, 1, 2, 3, 4])])),
                ('randomforestregressor', RandomForestRegressor())])
```

In [18]:
```python
# Pipeline for GB Boost
model_gb_boost = make_pipeline(transformer, preprocess_data, GradientBoostingRegressor())
model_gb_boost.fit(X_train, y_train)
```

Out[18]:
```
Pipeline(steps=[('pipeline', Pipeline(steps=[('scaler', StandardScaler())])),
                ('columntransformer',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('scaler',
                                                                   StandardScaler())]),
                                                  [0, 1, 2, 3, 4])])),
                ('gradientboostingregressor', GradientBoostingRegressor())])
```

## Evaluate Models

In [19]:

```
# Make predictions
reg_predictions = model_reg.predict(X_test)
ridge_predictions = model_ridge.predict(X_test)
forest_predictions = model_forest.predict(X_test)
gb_boost_predictions = model_gb_boost.predict(X_test)

# Get evaluation metrics
# Linear Regression
reg_mse = mean_squared_error(y_test, reg_predictions)
reg_rmse = np.sqrt(reg_mse)
reg_r2 = r2_score(y_test, reg_predictions)
print(f'Linear Regression results:\nMSE: {reg_mse}\nRMSE: {reg_rmse}\nR2_SCORE: {reg_r2}\n')

# Ridge
ridge_mse = mean_squared_error(y_test, ridge_predictions)
ridge_rmse = np.sqrt(ridge_mse)
ridge_r2 = r2_score(y_test, reg_predictions)
print(f'Ridge Regression results:\nMSE: {ridge_mse}\nRMSE: {ridge_rmse}\nR2_SCORE: {ridge_r2}\n')

# Random Forest
forest_mse = mean_squared_error(y_test, forest_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_r2 = r2_score(y_test, forest_predictions)
print(f'Random Forest Regression results:\nMSE: {forest_mse}\nRMSE: {forest_rmse}\nR2_SCORE: {forest_r2}\n')

# GB Boost
gb_boost_mse = mean_squared_error(y_test, gb_boost_predictions)
gb_boost_rmse = np.sqrt(gb_boost_mse)
gb_boost_r2 = r2_score(y_test, gb_boost_predictions)
print(f'Random Forest Regression results:\nMSE: {gb_boost_mse}\nRMSE: {gb_boost_rmse}\nR2_SCORE: {gb_boost_r2}')
```

```
Linear Regression results:
MSE: 43.01576372402771
RMSE: 6.558640386850594
R2_SCORE: 0.6781479656322735

Ridge Regression results:
MSE: 43.00464761992758
RMSE: 6.5577928924240645
R2_SCORE: 0.6781479656322735

Random Forest Regression results:
MSE: 26.975294438868193
RMSE: 5.193774584911074
R2_SCORE: 0.7981657736331546

Random Forest Regression results:
MSE: 26.323017673662722
RMSE: 5.130596229841394
R2_SCORE: 0.8030462310672954
```

## Use trained models to predict price

In [20]:

```
data = {'house_age': [16.2, 13.6],
        'transit_distance': [289.3248, 4082.015],
        'local_convenience_stores': [5, 0],
        'latitude': [24.98203, 24.94155],
        'longitude': [121.54348, 121.50381]}
predict_data = pd.DataFrame.from_dict(data)
predict_data
```

Out[20]:

| | house_age | transit_distance | local_convenience_stores | latitude | longitude |
|---|---|---|---|---|---|
| **0** | 16.2 | 289.3248 | 5 | 24.98203 | 121.54348 |
| **1** | 13.6 | 4082.0150 | 0 | 24.94155 | 121.50381 |

In [21]:

```
# Linear Regression
linear_price = model_reg.predict(predict_data)
print(f'Linear Regression predicted prices:')
for predicted_price in linear_price:
    print(predicted_price.round(2))

# Ridge
ridge_price = model_ridge.predict(predict_data)
print(f'\nRidge predicted prices:')
for predicted_price in ridge_price:
    print(predicted_price.round(2))

# Random Forest
forest_price = model_forest.predict(predict_data)
print(f'\nRandom Forest Regression predicted prices:')
for predicted_price in forest_price:
    print(predicted_price.round(2))

# Gradient Boosting
gb_price = model_gb_boost.predict(predict_data)
print(f'\nGradient Boosting Regressor predicted prices:')
for predicted_price in gb_price:
    print(predicted_price.round(2))
```

```
Linear Regression predicted prices:
45.59
15.26

Ridge predicted prices:
45.58
15.3

Random Forest Regression predicted prices:
49.27
15.82

Gradient Boosting Regressor predicted prices:
49.33
16.73
```

### Save models

In [22]:

```
reg = 'linear_regression_model.pkl'
ridge = 'ridge_regression_model.pkl'
forest = 'random_forest_regression_model.pkl'
gb = 'gradient_boosting_regressor_model.pkl'

joblib.dump(model_reg, reg)
joblib.dump(model_ridge, ridge)
joblib.dump(model_forest, forest)
joblib.dump(model_gb_boost, gb)
```

Out[22]:

```
['gradient_boosting_regressor_model.pkl']
```