10_Real_Estate

September 18, 2021

1 Regression Challenge

Predicting the selling price of a residential property depends on a number of factors, including the property age, availability of local amenities, and location.

In this challenge, you will use a dataset of real estate sales transactions to predict the price-per-unit of a property based on its features. The price-per-unit in this data is based on a unit measurement of 3.3 square meters.

> **Citation**: The data used in this exercise originates from the following study:
>
> *Yeh, I. C., & Hsu, T. K. (2018). Building real estate valuation models with comparative approach through case-based reasoning. Applied Soft Computing, 65, 260-271.*
>
> It was obtained from the UCI dataset repository (Dua, D. and Graff, C. (2019). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science).

1.1 Review the data

Run the following cell to load the data and view the first few rows.

[ ]:
```
import pandas as pd

# load the training dataset
data = pd.read_csv('real_estate.csv',parse_dates=True)
```

[ ]:
```
data.head()
```

[ ]:
```
   transaction_date  house_age  transit_distance  local_convenience_stores  \
0          2012.917       32.0          84.87882                        10
1          2012.917       19.5         306.59470                         9
2          2013.583       13.3         561.98450                         5
3          2013.500       13.3         561.98450                         5
4          2012.833        5.0         390.56840                         5

   latitude  longitude  price_per_unit
0  24.98298  121.54024            37.9
1  24.98034  121.53951            42.2
2  24.98746  121.54391            47.3
```

```
3  24.98746  121.54391            54.8
4  24.97937  121.54245            43.1
```

The data consists of the following variables:

- **transaction_date** - the transaction date (for example, 2013.250=2013 March, 2013.500=2013 June, etc.)
- **house_age** - the house age (in years)
- **transit_distance** - the distance to the nearest light rail station (in meters)
- **local_convenience_stores** - the number of convenience stores within walking distance
- **latitude** - the geographic coordinate, latitude
- **longitude** - the geographic coordinate, longitude
- **price_per_unit** house price of unit area (3.3 square meters)

1.2 Train a Regression Model

Your challenge is to explore and prepare the data, identify predictive features that will help predict the **price_per_unit** label, and train a regression model that achieves the lowest Root Mean Square Error (RMSE) you can achieve (which must be less than **7**) when evaluated against a test subset of data.

Add markdown and code cells as required to create your solution.

> **Note**: There is no single "correct" solution. A sample solution is provided in 02 - Real Estate Regression Solution.ipynb.

[ ]:
```
# Your code to explore data and train a regression model
```

1.3 Use the Trained Model

Save your trained model, and then use it to predict the price-per-unit for the following real estate transactions:

transaction_date	house_age	transit_distance	local_convenience_stores	latitude	longitude
2013.167	16.2	289.3248	5	24.98203	121.54348
2013.000	13.6	4082.015	0	24.94155	121.50381

[ ]:
```
# Your code to use the trained model
```

[ ]:
```
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
```

Note, price per unit is the dependent variable

[ ]:
```
data.isnull().sum().sum()
```

[ ]:
```
0
```

No null value

[ ]:
```
from datetime import datetime
import numpy as np
```

[ ]:
```
data['transaction_date2']=pd.to_datetime(data['transaction_date'])
```

[ ]:
```
#del data['transaction_date2']
```

[ ]:
```
data['price_per_unit']
```

[ ]:
```
0      37.9
1      42.2
2      47.3
3      54.8
4      43.1
       …
409    15.4
410    50.0
411    40.6
412    52.5
413    63.9
Name: price_per_unit, Length: 414, dtype: float64
```

[ ]:
```
data[data.columns[-1]]
```

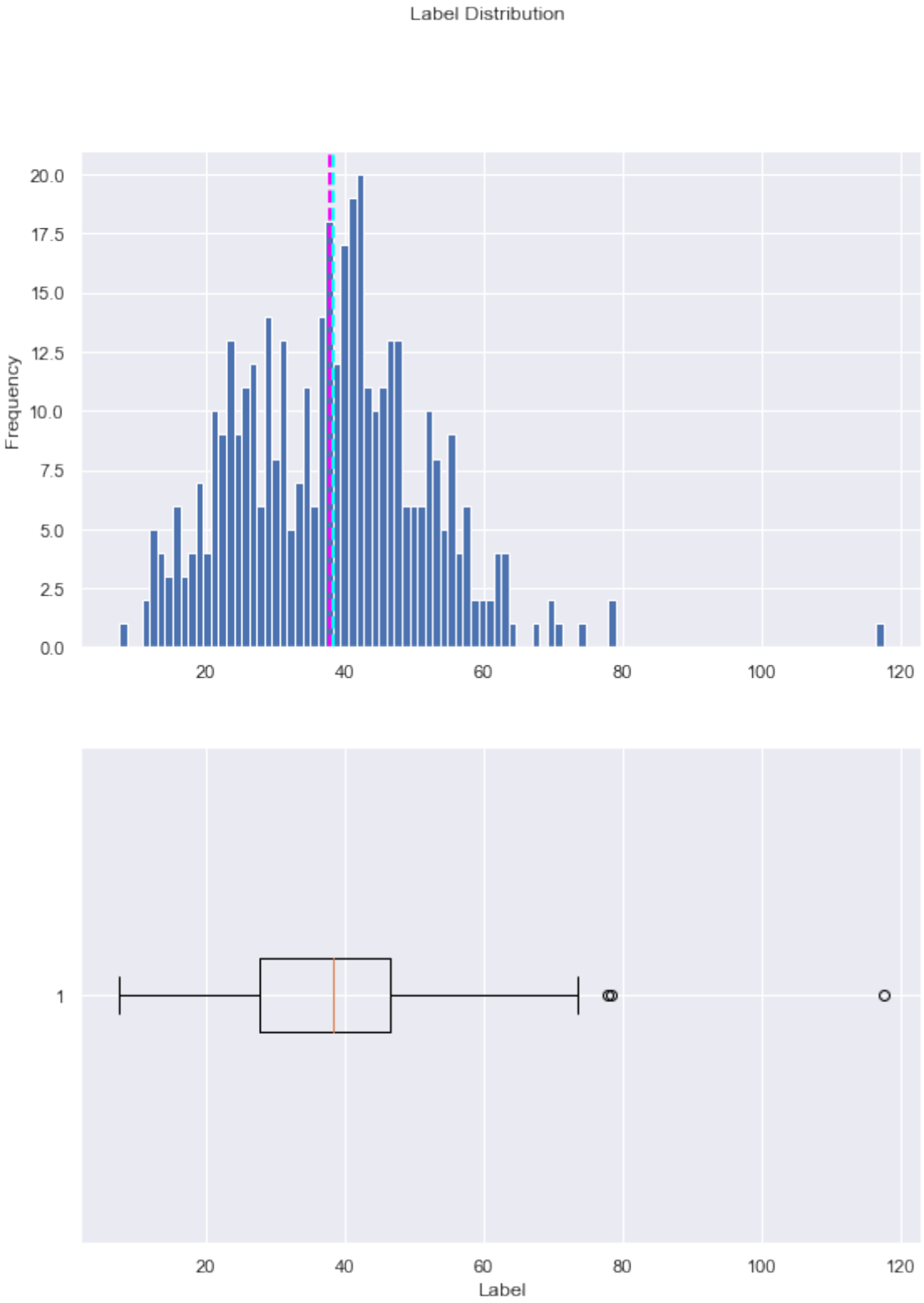
[ ]:
```
0      37.9
1      42.2
2      47.3
3      54.8
4      43.1
       …
409    15.4
410    50.0
411    40.6
412    52.5
```

```
413    63.9
Name: price_per_unit, Length: 414, dtype: float64
```

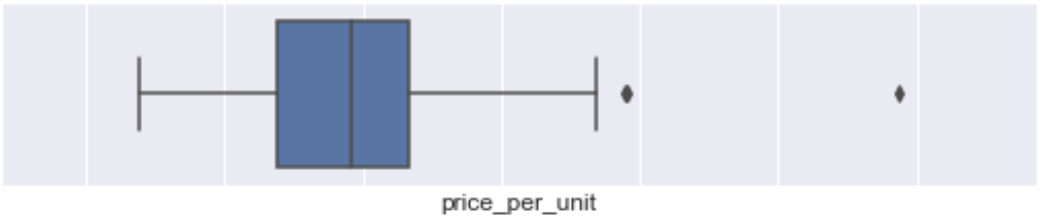
[ ]:
```
# import library & dataset
import seaborn as sns
```

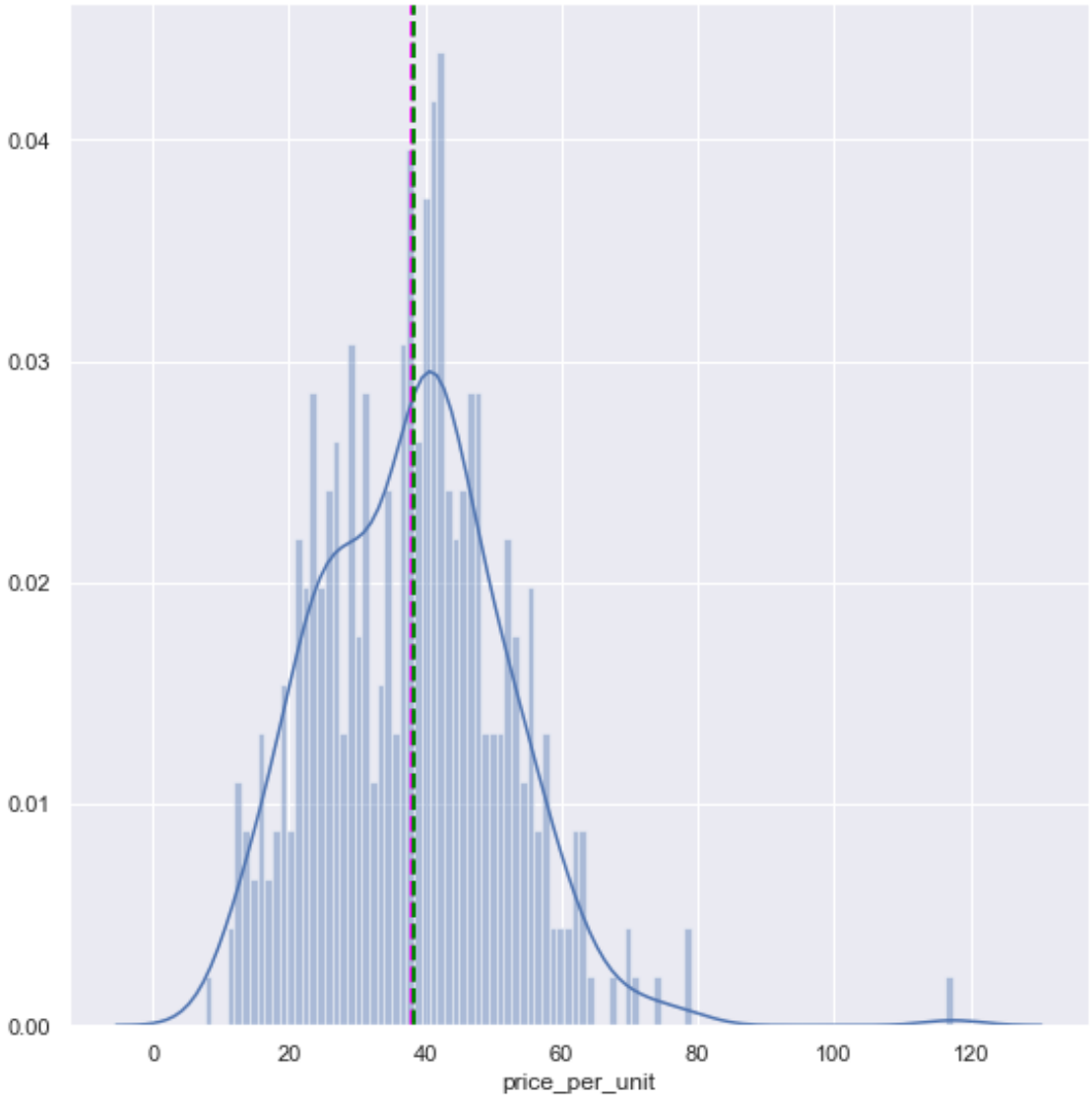
[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# Get the label column
label= data[data.columns[-1]]

# Create a figure for 2 subplots (2 rows, 1 column)
fig, ax = plt.subplots(2, 1, figsize = (9,12))

# Plot the histogram
ax[0].hist(label,bins=100)
ax[0].set_ylabel('Frequency')

# Add lines for the mean, median and mode
ax[0].axvline(label.mean(), color= 'magenta', linestyle='dashed', linewidth=2)
ax[0].axvline(label.median(), color='cyan', linestyle='dashed', linewidth=2)

# Plot the boxplot
ax[1].boxplot(label, vert=False)
ax[1].set_xlabel('Label')

# Add a title to the Figure
fig.suptitle('Label Distribution')

# Show the figure
fig.show()
```

```
C:\Users\aduzo\Anaconda3\lib\site-packages\ipykernel_launcher.py:27:
UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
```


Label Distribution
Frequency
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
20
40
60
80
100
120
1
20
40
60
80
100
120
Label

[ ]:
```
# set a grey background (use sns.set_theme() if seaborn version 0.11.0 or␣
 ↪above)
sns.set(style="darkgrid")

# creating a figure composed of two matplotlib.Axes objects (ax_box and ax_hist)
f, (ax_box, ax_hist) = plt.subplots(2, sharex=True,␣
 ↪gridspec_kw={"height_ratios": (.15, .85)},figsize = (9,12))

# assigning a graph to each ax
sns.boxplot(data["price_per_unit"],ax=ax_box)
sns.distplot(data["price_per_unit"],ax=ax_hist,bins=100)

ax_hist.axvline(data["price_per_unit"].mean(), color= 'magenta',␣
 ↪linestyle='dashed', linewidth=2)
ax_hist.axvline(data["price_per_unit"].median(), color='green',␣
 ↪linestyle='dashed', linewidth=2)

# Remove x axis name for the boxplot
#ax_box.set(xlabel='')
plt.show()
```





[ ]:
```
data["price_per_unit"].mean()
```

[ ]: 37.98019323671498

Remove outliers

[ ]:
```
# By visualising the plot above
data[data['price_per_unit']>70]
```

[ ]:
```
     transaction_date  house_age  transit_distance  local_convenience_stores  \
16           2013.250        0.0          292.9978                         6
105          2012.833        0.0          292.9978                         6
166          2013.417        0.0          292.9978                         6
220          2013.333       37.2          186.5101                         9
270          2013.333       10.8          252.5822                         1
312          2013.583       35.4          318.5292                         9

     latitude  longitude  price_per_unit
16   24.97744  121.54458            70.1
105  24.97744  121.54458            71.0
166  24.97744  121.54458            73.6
220  24.97703  121.54265            78.3
270  24.97460  121.53046           117.5
312  24.97071  121.54069            78.0
```

[ ]:
```
# Using IQR
def outlier_function(x):
    first= np.percentile(x, 25)
    third= np.percentile(x,75)
    iqr= third- first
    upper_threshold= third+ iqr*1.5
    lower_threshold= first- iqr*1.5
    outliers= {'upper_outliers': x[x> upper_threshold],
               'lower_outliers':x[x< lower_threshold]}
    return outliers
```

[ ]:
```
outlier_function(data[data.columns[-1]])
```

[ ]:
```
{'upper_outliers': 220     78.3
 270    117.5
 312     78.0
 Name: price_per_unit, dtype: float64,
 'lower_outliers': Series([], Name: price_per_unit, dtype: float64)}
```

[ ]:
```
def outlier_function_pd(x):
    first= np.percentile(x, 25)
    third= np.percentile(x,75)
    iqr= third- first
    upper_threshold= third+ iqr*1.5
    lower_threshold= first- iqr*1.5
    outliers= {'upper_outliers': x[x> upper_threshold],
               'lower_outliers':x[x< lower_threshold]}
    return outliers
```

[ ]:
```
outlier_function_pd(data['price_per_unit'])
```

[ ]:
```
{'upper_outliers': 220     78.3
 270    117.5
 312     78.0
 Name: price_per_unit, dtype: float64,
 'lower_outliers': Series([], Name: price_per_unit, dtype: float64)}
```

[ ]:
```
pd.concat([data[data['price_per_unit']>78],data[data['price_per_unit']<2]])
```

[ ]:
```
     transaction_date  house_age  transit_distance  local_convenience_stores  \
220          2013.333       37.2          186.5101                         9
270          2013.333       10.8          252.5822                         1

     latitude  longitude  price_per_unit
220  24.97703  121.54265            78.3
270  24.97460  121.53046           117.5
```

[ ]:
```
def outlier_function_pd(x):
    first= np.percentile(x, 25)
    third= np.percentile(x,75)
    iqr= third- first
    upper_threshold= third+ iqr*1.5
    lower_threshold= first- iqr*1.5
    outliers=pd.concat([x[x>upper_threshold],x[x<lower_threshold]])
    #outliers= {'upper_outliers': x[x> upper_threshold],
             #  'lower_outliers':x[x< lower_threshold]}
    return outliers
```

[ ]:
```
data[data['price_per_unit'].isin(outlier_function_pd(data['price_per_unit']))]
```

[ ]:
```
     transaction_date  house_age  transit_distance  local_convenience_stores  \
220          2013.333       37.2          186.5101                         9
270          2013.333       10.8          252.5822                         1
312          2013.583       35.4          318.5292                         9

     latitude  longitude  price_per_unit
220  24.97703  121.54265            78.3
270  24.97460  121.53046           117.5
312  24.97071  121.54069            78.0
```

from price_per_unit=78.0 is outlier

[ ]:
```
data[data['price_per_unit'].
 ↪isin(outlier_function_pd(data['price_per_unit']))==False]
```

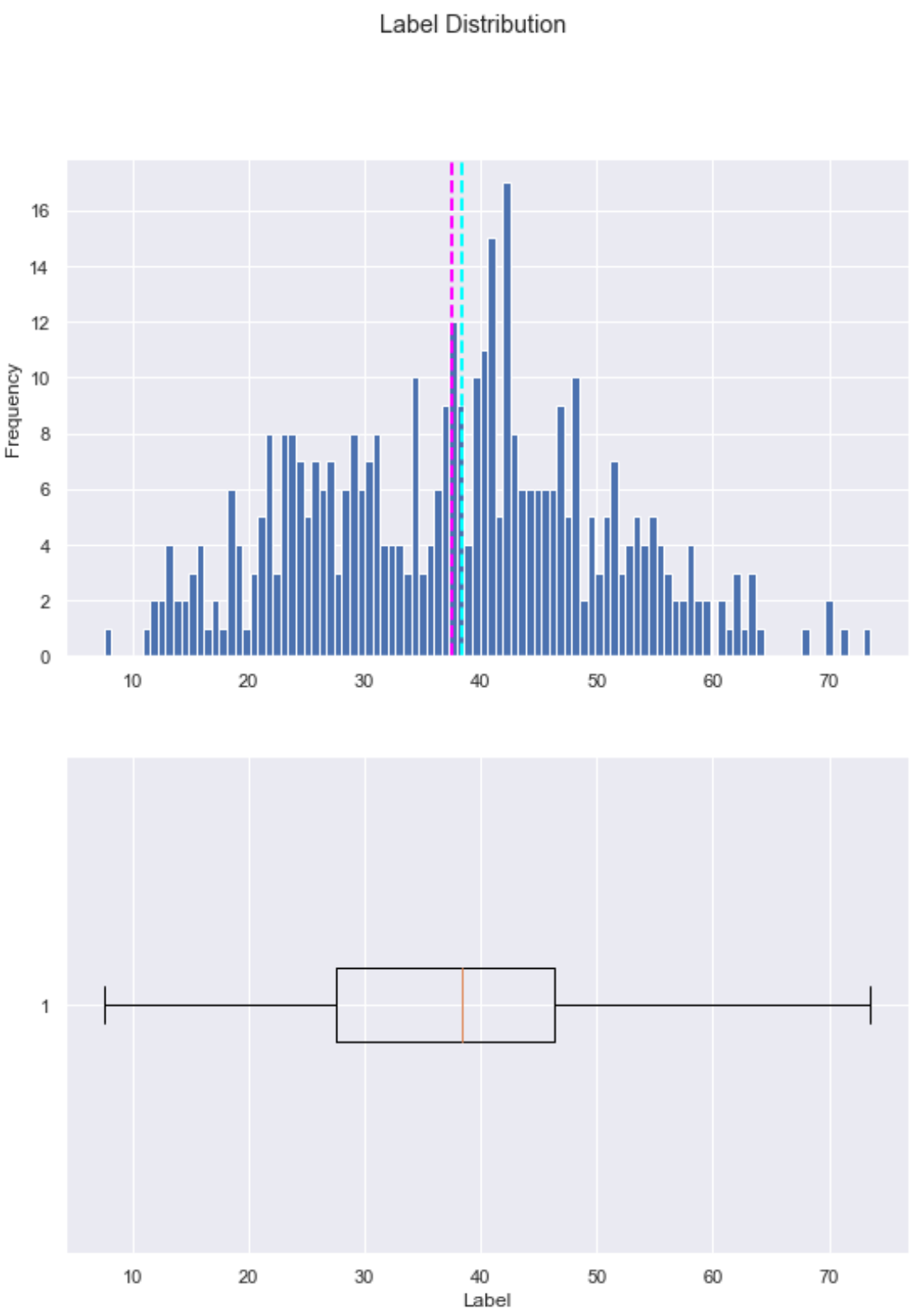
[ ]:
```
     transaction_date  house_age  transit_distance  local_convenience_stores  \
0          2012.917       32.0          84.87882                        10
```

```
1            2012.917       19.5         306.59470                        9
2            2013.583       13.3         561.98450                        5
3            2013.500       13.3         561.98450                        5
4            2012.833        5.0         390.56840                        5
..                ...        ...               ...                      ...
409          2013.000       13.7        4082.01500                        0
410          2012.667        5.6          90.45606                        9
411          2013.250       18.8         390.96960                        7
412          2013.000        8.1         104.81010                        5
413          2013.500        6.5          90.45606                        9

     latitude  longitude  price_per_unit
0    24.98298  121.54024            37.9
1    24.98034  121.53951            42.2
2    24.98746  121.54391            47.3
3    24.98746  121.54391            54.8
4    24.97937  121.54245            43.1
..        ...        ...             ...
409  24.94155  121.50381            15.4
410  24.97433  121.54310            50.0
411  24.97923  121.53986            40.6
412  24.96674  121.54067            52.5
413  24.97433  121.54310            63.9

[411 rows x 7 columns]
```

[ ]:
```
data= data[data['price_per_unit'].
 ↪isin(outlier_function_pd(data['price_per_unit']))==False]

# Get the label column
label = data[data.columns[-1]]

# Create a figure for 2 subplots (2 rows, 1 column)
fig, ax = plt.subplots(2, 1, figsize = (9,12))

# Plot the histogram
ax[0].hist(label, bins=100)
ax[0].set_ylabel('Frequency')

# Add lines for the mean, median, and mode
ax[0].axvline(label.mean(), color='magenta', linestyle='dashed', linewidth=2)
ax[0].axvline(label.median(), color='cyan', linestyle='dashed', linewidth=2)

# Plot the boxplot
ax[1].boxplot(label, vert=False)
ax[1].set_xlabel('Label')
```

```
# Add a title to the Figure
fig.suptitle('Label Distribution')

# Show the figure
fig.show()
```

```
C:\Users\aduzo\Anaconda3\lib\site-packages\ipykernel_launcher.py:25:
UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
```


Label Distribution
Frequency
0
2
4
6
8
10
12
14
16
10
20
30
40
50
60
70
1
10
20
30
40
50
60
70
Label

View numeric correlations

[ ]:
```
data.describe()
```

[ ]:
```
       transaction_date   house_age  transit_distance  \
count        411.000000  411.000000        411.000000
mean        2013.147019   17.638929       1089.953902
std            0.281884   11.354608       1264.697946
min         2012.667000    0.000000         23.382840
25%         2012.917000    8.950000        289.324800
50%         2013.167000   16.100000        492.231300
75%         2013.417000   27.800000       1455.798000
max         2013.583000   43.800000       6488.021000

       local_convenience_stores    latitude   longitude  price_per_unit
count                411.000000  411.000000  411.000000      411.000000
mean                   4.077859   24.968993  121.533328       37.591241
std                    2.932371    0.012446    0.015391       12.768915
min                    0.000000   24.932070  121.473530        7.600000
25%                    1.000000   24.962990  121.527600       27.500000
50%                    4.000000   24.971100  121.538630       38.400000
75%                    6.000000   24.977705  121.543395       46.300000
max                   10.000000   25.014590  121.566270       73.600000
```

[ ]:
```
data.nunique()
```

[ ]:
```
transaction_date            12
house_age                  233
transit_distance           256
local_convenience_stores    11
latitude                   231
longitude                  231
price_per_unit             267
dtype: int64
```

As seen from above, transaaction_date and local_convenience_stores looks like categorical variables as they have fewer values

[ ]:
```
print(f'transaction_date {data.transaction_date.unique()}')
print(f'local_convenience_stores {data.local_convenience_stores.unique()}')
```

```
transaction_date [2012.917 2013.583 2013.5   2012.833 2012.667 2013.417 2013.083
2013.333
 2013.25  2012.75  2013.    2013.167]
local_convenience_stores [10  9  5  3  7  6  1  4  2  8  0]
```

[ ]:
```
data[data.columns.difference(['transaction_date','local_convenience_stores'])]
```

[ ]:
```
     house_age  latitude  longitude  price_per_unit  transit_distance
0         32.0  24.98298  121.54024            37.9          84.87882
1         19.5  24.98034  121.53951            42.2         306.59470
2         13.3  24.98746  121.54391            47.3         561.98450
3         13.3  24.98746  121.54391            54.8         561.98450
4          5.0  24.97937  121.54245            43.1         390.56840
..         …       …         …                …               …
409       13.7  24.94155  121.50381            15.4        4082.01500
410        5.6  24.97433  121.54310            50.0          90.45606
411       18.8  24.97923  121.53986            40.6         390.96960
412        8.1  24.96674  121.54067            52.5         104.81010
413        6.5  24.97433  121.54310            63.9          90.45606

[411 rows x 5 columns]
```

[ ]:
```
correlation_data=data[data.columns.
↪difference(['transaction_date','local_convenience_stores'])].corr()
correlation_data
```

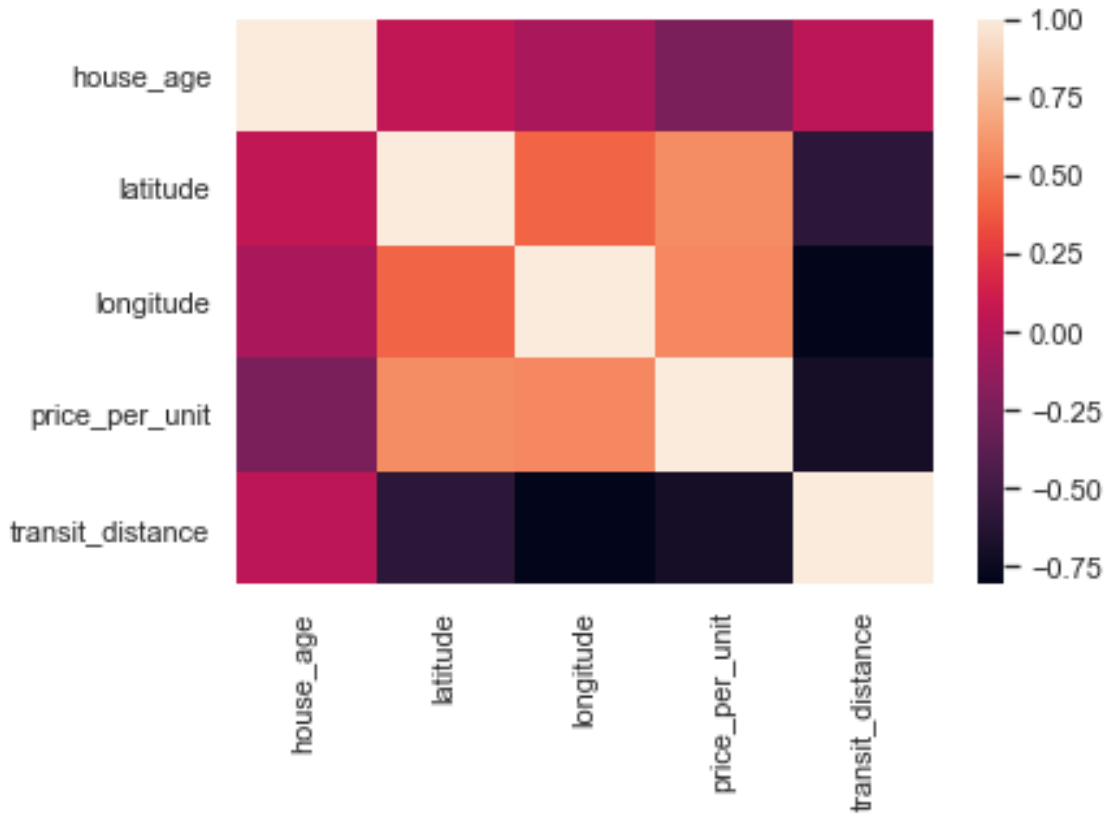
[ ]:
```
                house_age  latitude  longitude  price_per_unit  \
house_age        1.000000  0.052285  -0.053527       -0.242852
latitude         0.052285  1.000000   0.412657        0.571849
longitude       -0.053527  0.412657   1.000000        0.554585
price_per_unit  -0.242852  0.571849   0.554585        1.000000
transit_distance  0.030167 -0.590426  -0.806768       -0.701349

                transit_distance
house_age               0.030167
latitude               -0.590426
longitude              -0.806768
price_per_unit         -0.701349
transit_distance        1.000000
```

As seen price_per_unit is positively correlated with latitude and longitude and high negative correlation with transit_distance

- Thus, the higher the transit_distance reduces the price_per_unit

[ ]:
```
sns.heatmap(correlation_data)
```

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



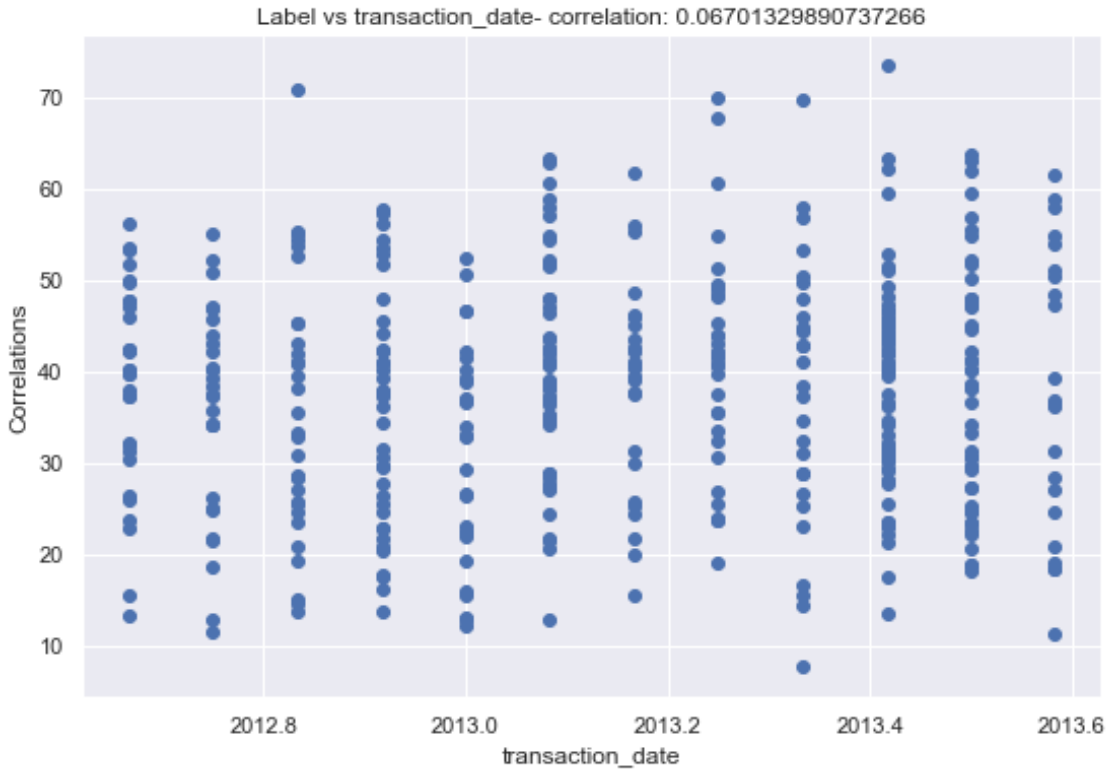
[ ]:
```
data[data.columns[0:-1]]
```

[ ]:
```
      transaction_date  house_age  transit_distance  local_convenience_stores  \
0             2012.917       32.0          84.87882                        10
1             2012.917       19.5         306.59470                         9
2             2013.583       13.3         561.98450                         5
3             2013.500       13.3         561.98450                         5
4             2012.833        5.0         390.56840                         5
..                 …          …                 …                       …
409           2013.000       13.7        4082.01500                         0
410           2012.667        5.6          90.45606                         9
411           2013.250       18.8         390.96960                         7
412           2013.000        8.1         104.81010                         5
413           2013.500        6.5          90.45606                         9

     latitude  longitude
0    24.98298  121.54024
1    24.98034  121.53951
2    24.98746  121.54391
3    24.98746  121.54391
4    24.97937  121.54245
```

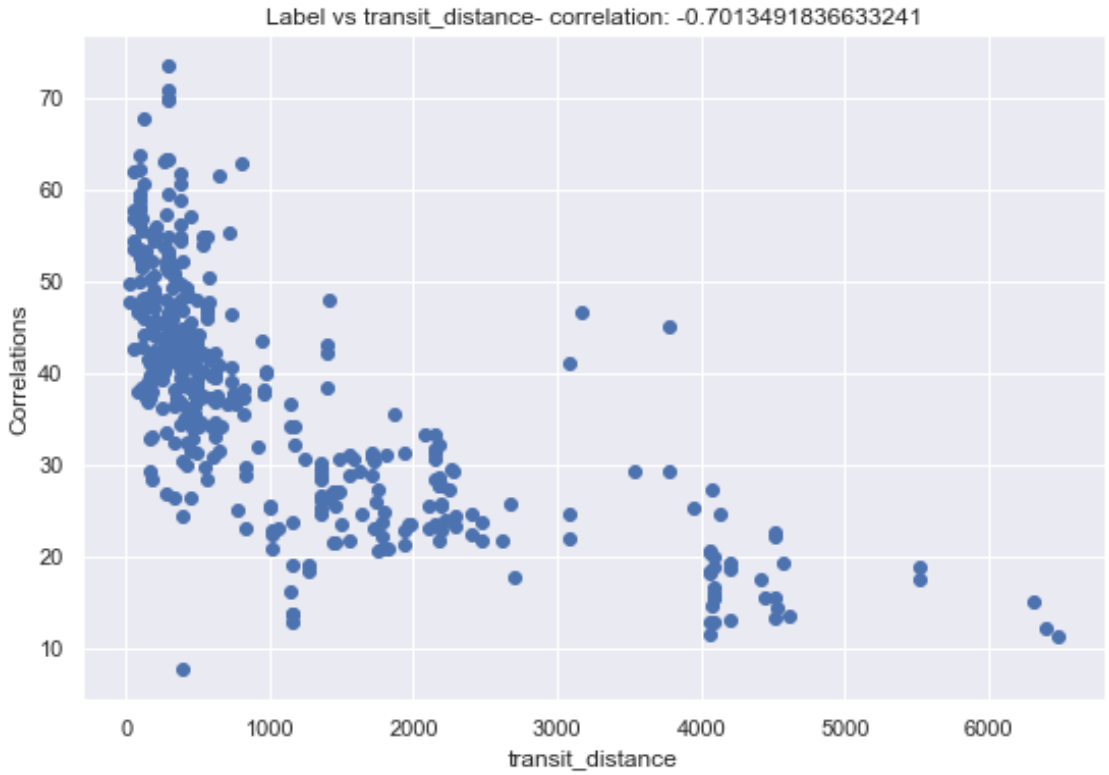
```
..         …         …
409   24.94155  121.50381
410   24.97433  121.54310
411   24.97923  121.53986
412   24.96674  121.54067
413   24.97433  121.54310

[411 rows x 6 columns]
```

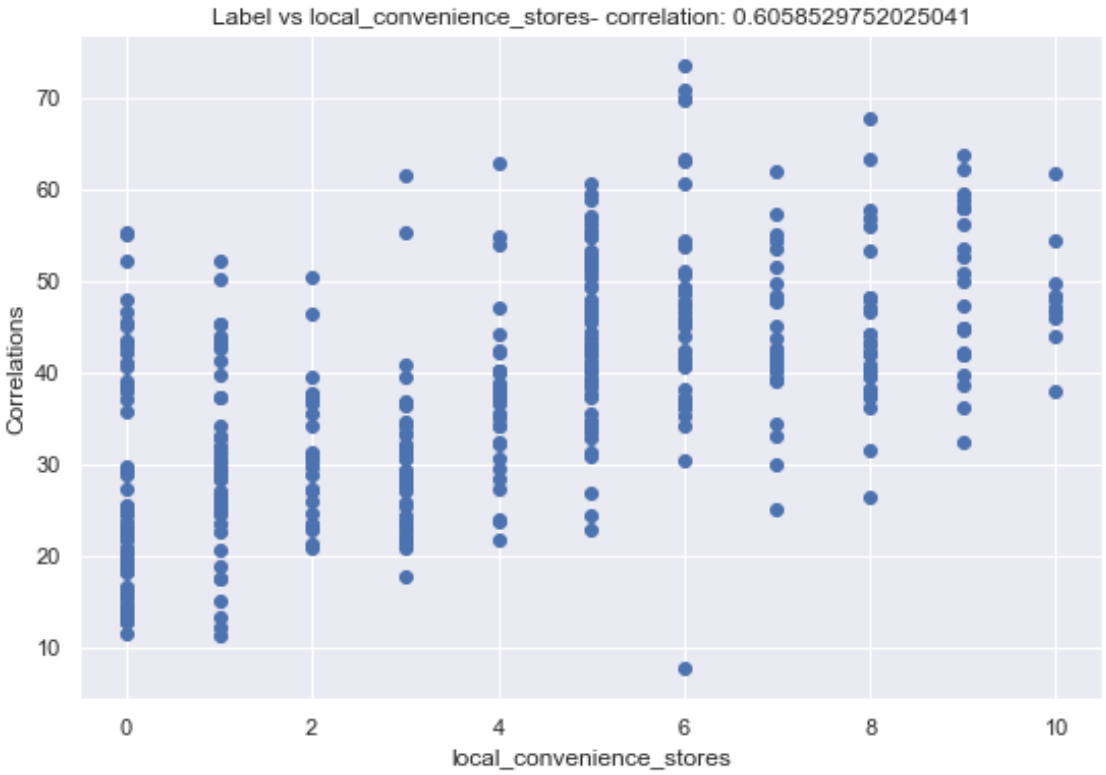
Checking correlation between label taht is price_per_unit and independent variables

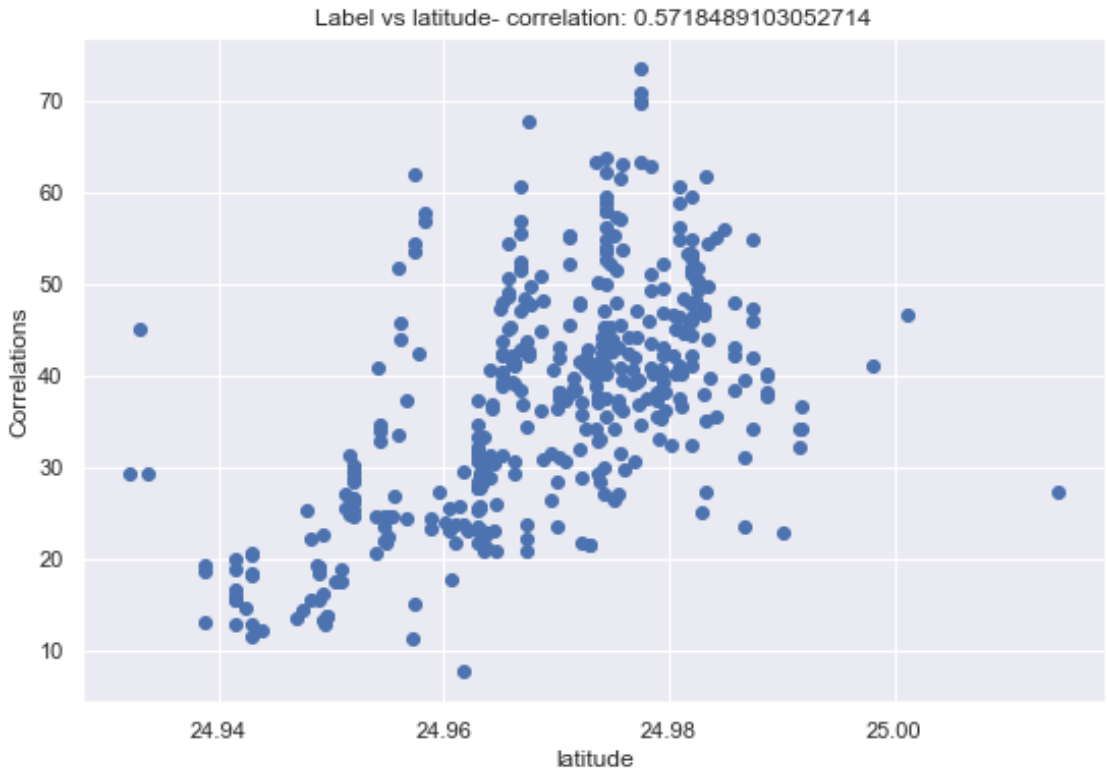
[ ]:
```
for col in data[data.columns[0:-1]]:
    fig = plt.figure(figsize=(9,6))
    ax =fig.gca()
    feature =data[col]
    correlation =feature.corr(label)
    plt.scatter(x=feature, y=label)
    plt.xlabel(col)
    plt.ylabel('Correlations')
    ax.set_title('Label vs '+ col + '- correlation: '+ str(correlation))

plt.show()
```

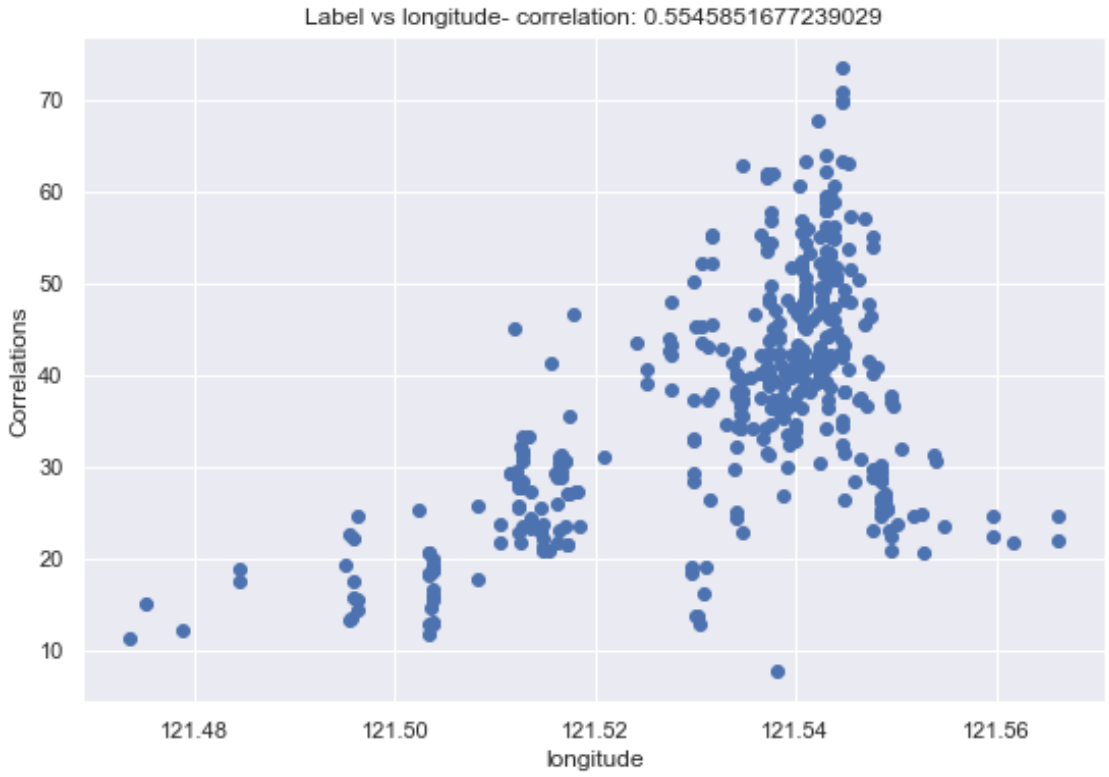



Label vs house_age- correlation: -0.24285150182676524
Correlations
house_age
70
60
50
40
30
20
10
0
10
20
30
40


Label vs transit_distance- correlation: -0.7013491836633241
Correlations
transit_distance
70
60
50
40
30
20
10
0
1000
2000
3000
4000
5000
6000


Label vs local_convenience_stores- correlation: 0.6058529752025041
Correlations
70
60
50
40
30
20
10
0
2
4
6
8
10
local_convenience_stores


Label vs latitude- correlation: 0.5718489103052714
Correlations
70
60
50
40
30
20
10
24.94
24.96
24.98
25.00
latitude


Label vs longitude- correlation: 0.5545851677239029
Correlations
70
60
50
40
30
20
10
121.48
121.50
121.52
121.54
121.56
longitude

1.3.1 View categorical features

(**transaction_date** and **local_convenience_stores** seem to be discrete values, so might work better if treated as categorical features)

[ ]:
```
data[['transaction_date', 'local_convenience_stores']]
```

[ ]:

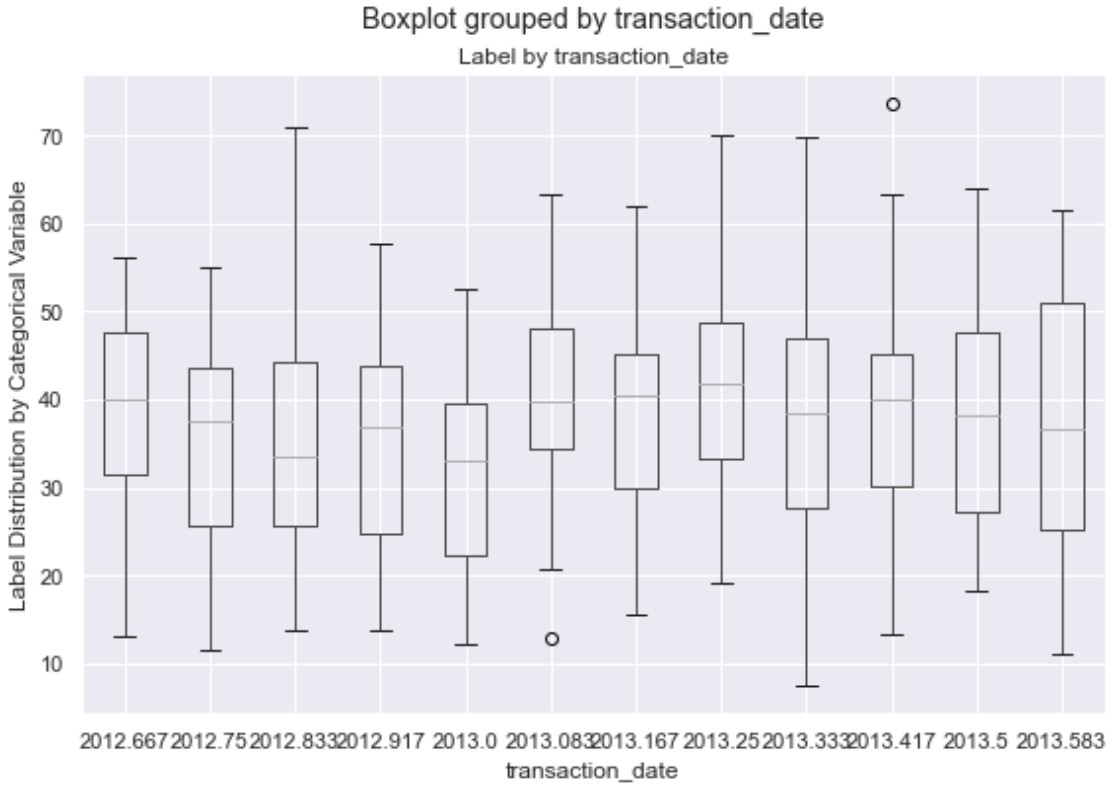
	transaction_date	local_convenience_stores
0	2012.917	10
1	2012.917	9
2	2013.583	5
3	2013.500	5
4	2012.833	5
..	…	…
409	2013.000	0
410	2012.667	9
411	2013.250	7
412	2013.000	5
413	2013.500	9

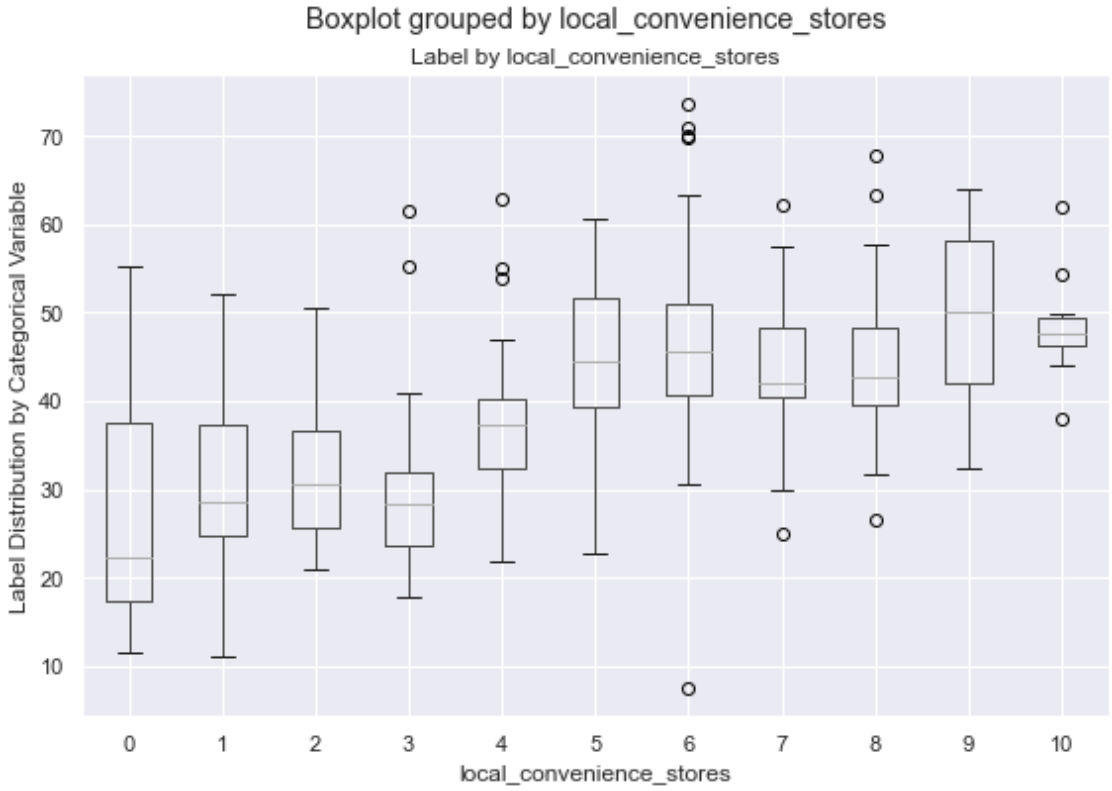
[411 rows x 2 columns]

[ ]:
```
for col in data[['transaction_date', 'local_convenience_stores']]:
    print(col)
```

```
transaction_date
local_convenience_stores
```

[ ]:
```
# plot a boxplot for the label by each categorical feature
for col in data[['transaction_date', 'local_convenience_stores']]:
    fig = plt.figure(figsize=(9, 6))
    ax = fig.gca()
    data.boxplot(column = 'price_per_unit', by = col, ax = ax)
    ax.set_title('Label by ' + col)
    ax.set_ylabel("Label Distribution by Categorical Variable")
plt.show()
```


Boxplot grouped by transaction_date
Label by transaction_date
Label Distribution by Categorical Variable
70
60
50
40
30
20
10
2012.667
2012.75
2012.833
2012.917
2013.0
2013.083
2013.167
2013.25
2013.333
2013.417
2013.5
2013.583
transaction_date

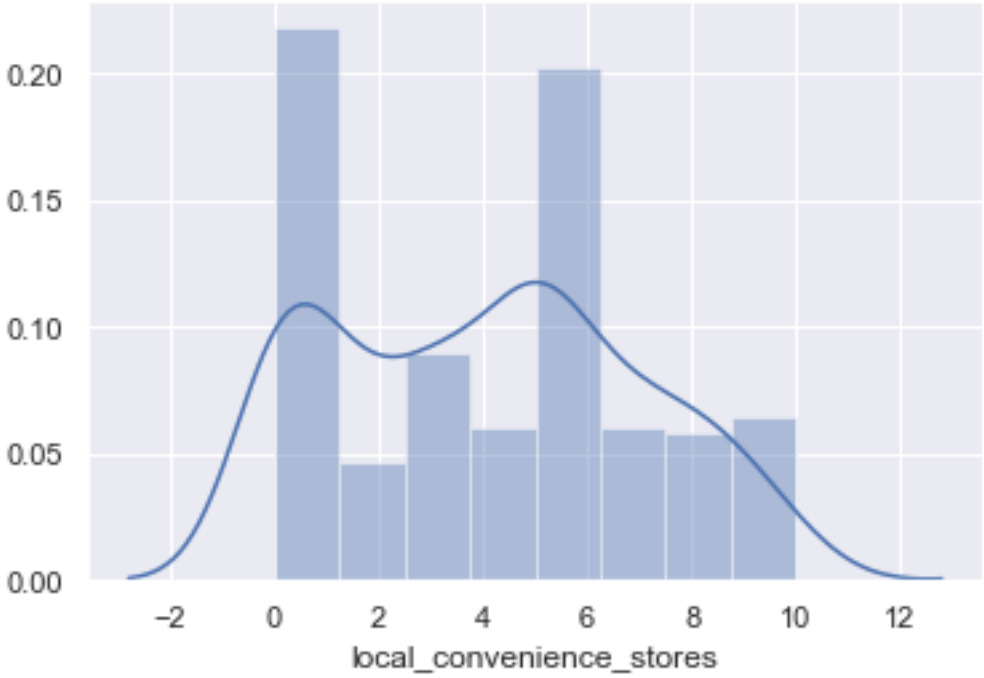

Boxplot grouped by local_convenience_stores
Label by local_convenience_stores
Label Distribution by Categorical Variable
70
60
50
40
30
20
10
0
1
2
3
4
5
6
7
8
9
10
local_convenience_stores

there are some up and down movements in transaction dates, with variance increases at both ends of the dates, but this doesnot seems to be so predictive.

For local_convinience_store, store 10 has very low variance, while stores 0 and 9 have high variance, the means of these local stores varies alot, this is a useful feature.

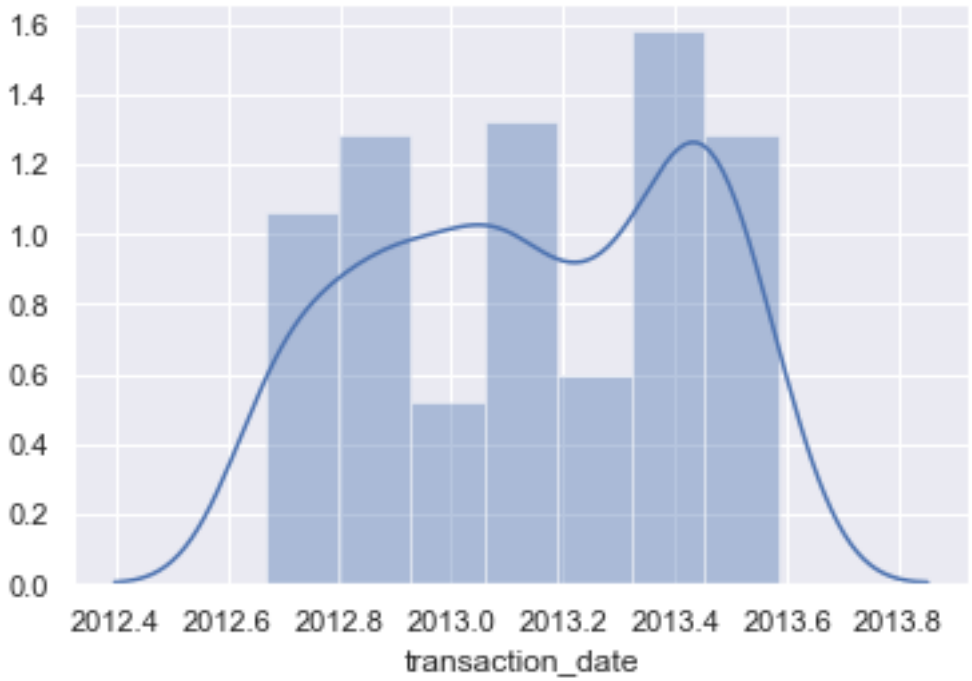
[ ]:
```
sns.distplot(data.local_convenience_stores)
```

[ ]:
```
<AxesSubplot:xlabel='local_convenience_stores'>
```



[ ]:
```
sns.distplot(data.transaction_date)
```

[ ]:
```
<AxesSubplot:xlabel='transaction_date'>
```



[ ]:
```
data.transaction_date.value_counts()
```

[ ]:
```
2013.417    58
2013.500    47
2013.083    46
2012.917    38
2013.250    32
2012.833    31
2012.667    30
2013.000    28
2012.750    27
2013.333    27
2013.167    25
2013.583    22
Name: transaction_date, dtype: int64
```

1.3.2 Separate features and label and split data for training and validation

(**transaction_date** doesn't seem to be very predictive, so omit it)

[ ]:
```
data.columns
```

[ ]:
```
Index(['transaction_date', 'house_age', 'transit_distance',
       'local_convenience_stores', 'latitude', 'longitude', 'price_per_unit'],
      dtype='object')
```

```
[ ]: data[data.columns[1:-1]].head() # 0,1,3,4 will be selected numeric features
```

```
[ ]:    house_age  transit_distance  local_convenience_stores  latitude  longitude
     0       32.0          84.87882                        10  24.98298  121.54024
     1       19.5         306.59470                         9  24.98034  121.53951
     2       13.3         561.98450                         5  24.98746  121.54391
     3       13.3         561.98450                         5  24.98746  121.54391
     4        5.0         390.56840                         5  24.97937  121.54245
```

data[data.columns[-1]] is the label

```
[ ]: #data[data.columns[1:-1]].values, data[data.columns[-1]].values
```

```
[ ]: (array([[ 32.     ,  84.87882,  10.     ,  24.98298, 121.54024],
            [ 19.5    , 306.5947 ,   9.     ,  24.98034, 121.53951],
            [ 13.3    , 561.9845 ,   5.     ,  24.98746, 121.54391],
            …,
            [ 18.8    , 390.9696 ,   7.     ,  24.97923, 121.53986],
            [  8.1    , 104.8101 ,   5.     ,  24.96674, 121.54067],
            [  6.5    ,  90.45606,   9.     ,  24.97433, 121.5431 ]]),
     array([37.9, 42.2, 47.3, 54.8, 43.1, 32.1, 40.3, 46.7, 18.8, 22.1, 41.4,
            58.1, 39.3, 23.8, 34.3, 50.5, 70.1, 37.4, 42.3, 47.7, 29.3, 51.6,
            24.6, 47.9, 38.8, 27. , 56.2, 33.6, 47. , 57.1, 22.1, 25. , 34.2,
            49.3, 55.1, 27.3, 22.9, 25.3, 47.7, 46.2, 15.9, 18.2, 34.7, 34.1,
            53.9, 38.3, 42. , 61.5, 13.4, 13.2, 44.2, 20.7, 27. , 38.9, 51.7,
            13.7, 41.9, 53.5, 22.6, 42.4, 21.3, 63.2, 27.7, 55. , 25.3, 44.3,
            50.7, 56.8, 36.2, 42. , 59. , 40.8, 36.3, 20. , 54.4, 29.5, 36.8,
            25.6, 29.8, 26.5, 40.3, 36.8, 48.1, 17.7, 43.7, 50.8, 27. , 18.3,
            48. , 25.3, 45.4, 43.2, 21.8, 16.1, 41. , 51.8, 59.5, 34.6, 51. ,
            62.2, 38.2, 32.9, 54.4, 45.7, 30.5, 71. , 47.1, 26.6, 34.1, 28.4,
            51.6, 39.4, 23.1,  7.6, 53.3, 46.4, 12.2, 13. , 30.6, 59.6, 31.3,
            48. , 32.5, 45.5, 57.4, 48.6, 62.9, 55. , 60.7, 41. , 37.5, 30.7,
            37.5, 39.5, 42.2, 20.8, 46.8, 47.4, 43.5, 42.5, 51.4, 28.9, 37.5,
            40.1, 28.4, 45.5, 52.2, 43.2, 45.1, 39.7, 48.5, 44.7, 28.9, 40.9,
            20.7, 15.6, 18.3, 35.6, 39.4, 37.4, 57.8, 39.6, 11.6, 55.5, 55.2,
            30.6, 73.6, 43.4, 37.4, 23.5, 14.4, 58.8, 58.1, 35.1, 45.2, 36.5,
            19.2, 42. , 36.7, 42.6, 15.5, 55.9, 23.6, 18.8, 21.8, 21.5, 25.7,
            22. , 44.3, 20.5, 42.3, 37.8, 42.7, 49.3, 29.3, 34.6, 36.6, 48.2,
            39.1, 31.6, 25.5, 45.9, 31.5, 46.1, 26.6, 21.4, 44. , 34.2, 26.2,
            40.9, 52.2, 43.5, 31.1, 58. , 20.9, 48.1, 39.7, 40.8, 43.8, 40.2,
            38.5, 48.5, 42.3, 46. , 49. , 12.8, 40.2, 46.6, 19. , 33.4, 14.7,
            17.4, 32.4, 23.9, 39.3, 61.9, 39. , 40.6, 29.7, 28.8, 41.4, 33.4,
            48.2, 21.7, 40.8, 40.6, 23.1, 22.3, 15. , 30. , 13.8, 52.7, 25.9,
            51.8, 17.4, 26.5, 43.9, 63.3, 28.8, 30.7, 24.4, 53. , 31.7, 40.6,
            38.1, 23.7, 41.1, 40.1, 23. , 26.5, 40.5, 29.3, 41. , 49.7, 34. ,
            27.7, 44. , 31.1, 45.4, 44.8, 25.6, 23.5, 34.4, 55.3, 56.3, 32.9,
            51. , 44.5, 37. , 54.4, 24.5, 42.5, 38.1, 21.8, 34.1, 28.5, 16.7,
            46.1, 36.9, 35.7, 23.2, 38.4, 29.4, 55. , 50.2, 24.7, 53. , 19.1,
```

```
        24.7, 42.2, 42.8, 41.6, 27.3, 42. , 37.5, 49.8, 26.9, 18.6, 37.7,
        33.1, 42.5, 31.3, 38.1, 62.1, 36.7, 23.6, 19.2, 12.8, 15.6, 39.6,
        38.4, 22.8, 36.5, 35.6, 30.9, 36.3, 50.4, 42.9, 37. , 53.5, 46.6,
        41.2, 37.9, 30.8, 11.2, 53.7, 47. , 42.3, 28.6, 25.7, 31.3, 30.1,
        60.7, 45.3, 44.9, 45.1, 24.7, 47.1, 63.3, 40. , 48. , 33.1, 29.5,
        24.8, 20.9, 43.1, 22.8, 42.1, 51.7, 41.5, 52.2, 49.5, 23.8, 30.5,
        56.8, 37.4, 69.7, 53.3, 47.3, 29.3, 40.3, 12.9, 46.6, 55.3, 25.6,
        27.3, 67.7, 38.6, 31.3, 35.3, 40.3, 24.7, 42.5, 31.9, 32.2, 23. ,
        37.3, 35.5, 27.7, 28.5, 39.7, 41.2, 37.2, 40.5, 22.3, 28.1, 15.4,
        50. , 40.6, 52.5, 63.9]))
```

[ ]:
```
from sklearn.model_selection import train_test_split

# Separate features (columns 1 [house_age] to the last but one) and labels (the␣
 ↪last column)
X, y = data[data.columns[1:-1]].values, data[data.columns[-1]].values

# Split data 70%-30% into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
 ↪random_state=0)

print ('Training Set: %d, rows\nTest Set: %d rows' % (X_train.shape[0], X_test.
 ↪shape[0]))
```

```
Training Set: 287, rows
Test Set: 124 rows
```

1.3.3 Preprocess the data and train a model in a pipeline

Normalize the numeric features, then use a RandomForestRegressor to train a model.

[ ]:
```
# Train the model
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OrdinalEncoder
from sklearn.ensemble import RandomForestRegressor
import numpy as np

# Define preprocessing for numeric columns (scale them)
numeric_features = [0,1,3,4]
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())])

# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
```

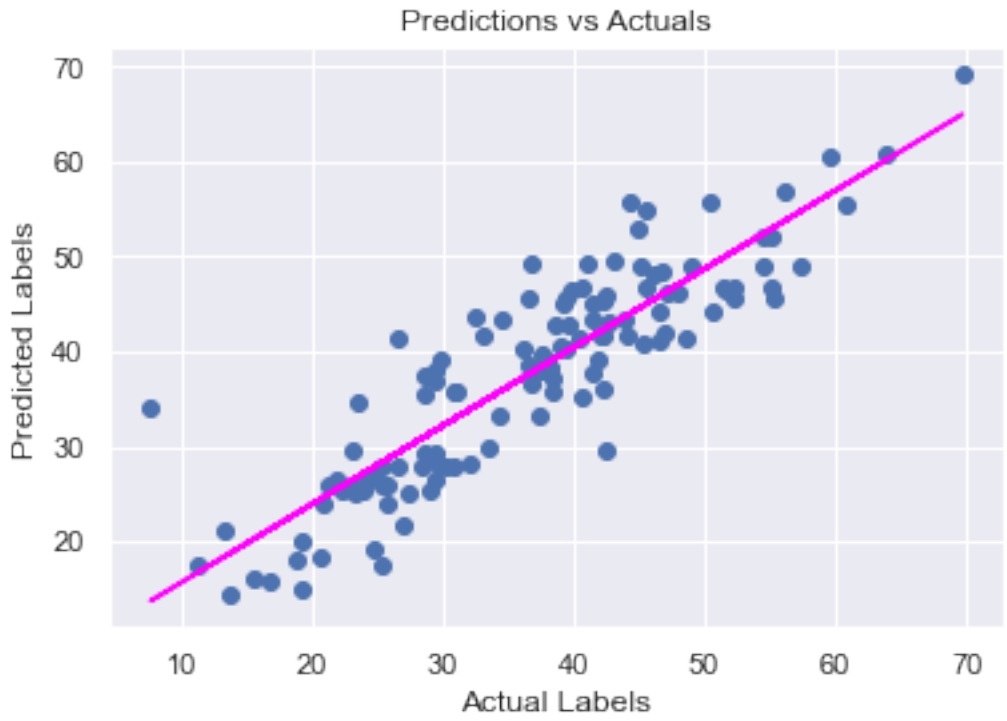
```
    ])

# Create preprocessing and training pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('regressor', RandomForestRegressor())])


# fit the pipeline to train a linear regression model on the training set
model = pipeline.fit(X_train, (y_train))
print (model)
```

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('scaler',
StandardScaler())]),
                                                  [0, 1, 3, 4])])),
                ('regressor', RandomForestRegressor())])
```

Evaluate the model

[ ]:
```
import matplotlib.pyplot as plt
from  sklearn.metrics import mean_squared_error, r2_score
%matplotlib inline

# Get predictions
predictions = model.predict(X_test)

# Display metrics
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Predictions vs Actuals')
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
plt.show()
```

```
MSE: 32.556988807323876
RMSE: 5.705873185352429
R2: 0.7697129912848865
```



1.4 Use the Trained Model

Save your trained model, and then use it to predict the price-per-unit for the following real estate transactions:

transaction_date	house_age	transit_distance	local_convenience_stores	latitude	longitude
2013.167	16.2	289.3248	5	24.98203	121.54348
2013.000	13.6	4082.015	0	24.94155	121.50381

[ ]:
```
import joblib

# Save the model as a pickle file
filename = 'real_estate_model.pkl'
joblib.dump(model, filename)

# Load the model from the file
loaded_model = joblib.load(filename)

# An array of features for each transaction (don't include the transaction date)
X_new = np.array([[16.2,289.3248,5,24.98203,121.54348],
                  [13.6,4082.015,0,24.94155,121.5038]])
```

```
# Use the model to predict unit price
results = loaded_model.predict(X_new)
print('Predictions:')
for prediction in results:
    print(round(prediction,2))
```

```
Predictions:
46.88
16.23
```