AN7914 Week 12 Python

May 3, 2024

1 Week 12 Python

1.1 Introduction to Regressions

- binary means (close vs far)

- simple linear regression (OLS)

- analysis of the results
- Log models
- Non-linear models

Dataset:

- hotels-vienna

1.2 Introduction to Regression

Import packages

[1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns


%matplotlib inline
plt.style.use('bmh')
#sns.set_style("dark")


pd.set_option('display.max_columns',50)
```

From OSF import hotel-vienna data

[2]:
```
hotels = pd.read_csv("https://osf.io/y6jvb/download")
```

[3]:
```
hotels
```

[3]:
```
    country city_actual  rating_count center1label center2label  \
0   Austria      Vienna          36.0  City centre    Donauturm
1   Austria      Vienna         189.0  City centre    Donauturm
2   Austria      Vienna          53.0  City centre    Donauturm
3   Austria      Vienna          55.0  City centre    Donauturm
4   Austria      Vienna          33.0  City centre    Donauturm
..      …          …            …          …            …
423 Austria      Vienna           2.0  City centre    Donauturm
424 Austria      Vienna         145.0  City centre    Donauturm
425 Austria      Vienna         112.0  City centre    Donauturm
426 Austria      Vienna         169.0  City centre    Donauturm
427 Austria      Vienna          80.0  City centre    Donauturm

   neighbourhood  price    city  stars  ratingta  ratingta_count  \
0     17. Hernals     81  Vienna    4.0       4.5           216.0
1     17. Hernals     81  Vienna    4.0       3.5           708.0
2      Alsergrund     85  Vienna    4.0       3.5           629.0
3      Alsergrund     83  Vienna    3.0       4.0            52.0
4      Alsergrund     82  Vienna    4.0       3.5           219.0
..            …      …     …     …       …             …
423        Wieden    109  Vienna    3.0       3.0            14.0
424        Wieden    185  Vienna    5.0       4.0           740.0
425        Wieden    100  Vienna    4.0       4.5          1006.0
426        Wieden     58  Vienna    3.0       3.0           135.0
427        Wieden    110  Vienna    3.5       NaN             NaN

    scarce_room  hotel_id  offer      offer_cat  year  month  weekend  \
0             1     21894      1  15-50% offer  2017     11        0
1             0     21897      1   1-15% offer  2017     11        0
2             0     21901      1  15-50% offer  2017     11        0
3             0     21902      1  15-50% offer  2017     11        0
4             1     21903      1  15-50% offer  2017     11        0
..          …        …      …           …     …     …       …
423           1     22404      1  50%-75% offer  2017     11        0
424           0     22406      1  15-50% offer  2017     11        0
425           1     22407      0    0% no offer  2017     11        0
426           0     22408      1  15-50% offer  2017     11        0
427           1     22409      1   1-15% offer  2017     11        0

    holiday  distance  distance_alter accommodation_type  nnights  rating
0         0       2.7             4.4          Apartment        1     4.4
1         0       1.7             3.8              Hotel        1     3.9
2         0       1.4             2.5              Hotel        1     3.7
3         0       1.7             2.5              Hotel        1     4.0
4         0       1.2             2.8              Hotel        1     3.9
```

```
..          …         …             …              …     …     …
423         0        1.5           3.8       Apartment       1     5.0
424         0        0.8           3.6           Hotel       1     4.3
425         0        1.0           3.7           Hotel       1     4.4
426         0        1.4           4.1           Hotel       1     3.2
427         0        0.7           3.5       Apartment       1     4.0

[428 rows x 24 columns]
```

[4]:
```python
hotels.shape
```

[4]:
```
(428, 24)
```

Apply filters: 3-4 stars, Vienna actual, without extreme prices

[5]:
```python
#hotels = (
#    hotels.loc[lambda x: x["accommodation_type"] == "Hotel"]
#    .loc[lambda x: x["city_actual"] == "Vienna"]
#    .loc[lambda x: x["stars"] >= 3]
#    .loc[lambda x: x["stars"] <= 4]
#    .loc[lambda x: x["stars"].notnull()]
#    .loc[lambda x: x["price"] <= 600]
#)
#hotels = hotels[(hotels["accommodation_type"] == "Hotel")
#                 & (hotels["city_actual"] == "Vienna")
#                 & (hotels["stars"] >= 3)
#                 & (hotels["stars"] <= 4)
#                 & (hotels["stars"].notnull())
#                 & (hotels["price"] <= 600)]


# Filter by accommodation_type
hotels = hotels[hotels["accommodation_type"] == "Hotel"]

# Filter by city_actual
hotels = hotels[hotels["city_actual"] == "Vienna"]

# Filter by stars between 3 and 4
hotels = hotels[(hotels["stars"] >= 3) & (hotels["stars"] <= 4)]

# Filter out null stars
hotels = hotels[hotels["stars"].notnull()]

# Filter by price <= 600
hotels = hotels[hotels["price"] <= 600]
```

Let’s break down each line of the code:

1. **Filter by accommodation_type**: This line selects rows from the DataFrame where the value in the "accommodation_type" column is equal to "Hotel". It uses boolean indexing to achieve this. The resulting DataFrame contains only rows where the accommodation type is "Hotel".
2. **Filter by city_actual**: Similar to the previous line, this line selects rows where the value in the "city_actual" column is equal to "Vienna". Again, it uses boolean indexing to filter the DataFrame based on the city.
3. **Filter by stars between 3 and 4**: Here, we filter the DataFrame to include only those rows where the value in the "stars" column is between 3 and 4, inclusive. We use a combination of two conditions separated by the logical AND (`&`) operator inside square brackets.
4. **Filter out null stars**: This line ensures that only rows with non-null values in the "stars" column are retained in the DataFrame. It filters out any rows where the "stars" column contains missing values (NaN).
5. **Filter by price <= 600**: Finally, this line selects rows where the value in the "price" column is less than or equal to 600. It again uses boolean indexing to filter the DataFrame based on the price criterion.

Each line of code progressively refines the DataFrame by applying additional filters based on specific criteria such as accommodation type, city, star rating, absence of null stars, and price. As a result, the final DataFrame (`hotels`) contains only those rows that satisfy all the specified conditions.

You can also use the commented out code in the previous to get the same result!

Summary statistics on price and distance

[6]:
```
hotels.filter(["price", "distance"]).describe(percentiles=[0.25, 0.5, 0.75, 0.
↪95]).T
```

[6]:

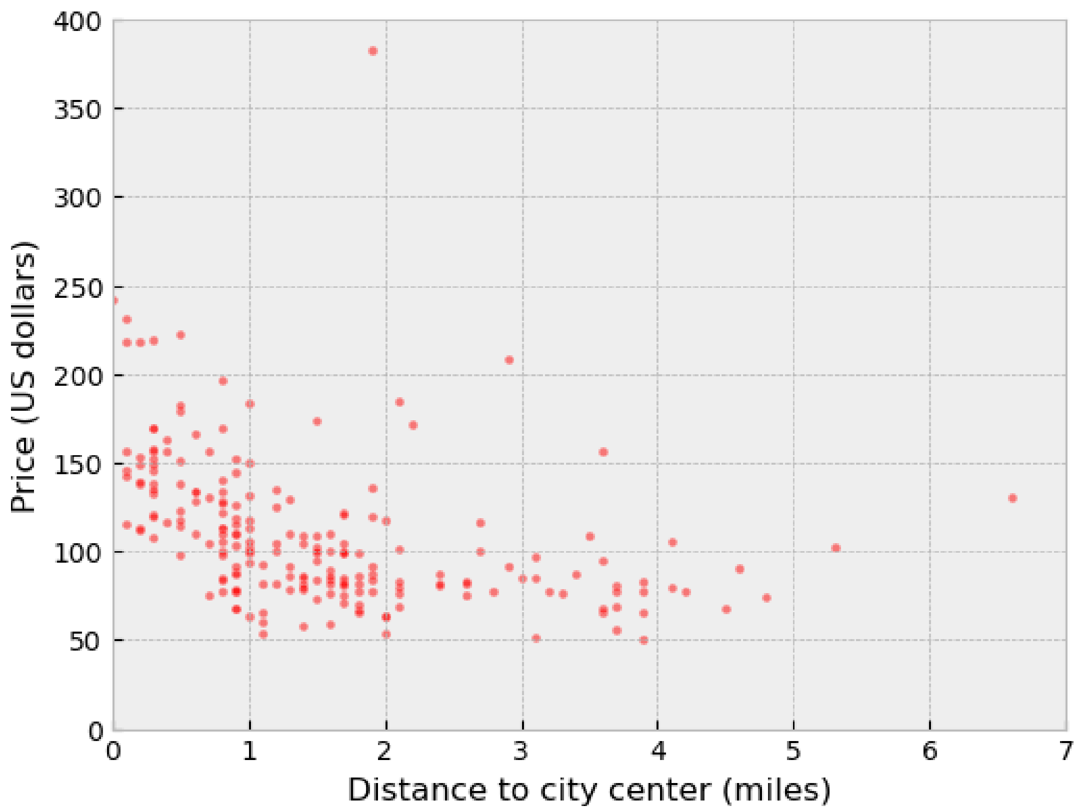
	count	mean	std	min	25%	50%	75%	95%	max
price	207.0	109.975845	42.221381	50.0	82.0	100.0	129.5	183.4	383.0
distance	207.0	1.529952	1.161507	0.0	0.8	1.3	1.9	3.9	6.6

This will return a pandas DataFrame with the following columns: count, mean, std, min, 25%, 50%, 75%, and 95% percentiles. The `T` method transposes the DataFrame so that the statistics are displayed as rows instead of columns.

Graphical investigation:
create a base scatter-plot between price and distance

[7]:
```
# Create scatterplot with seaborn
sns.scatterplot(data=hotels, x="distance", y="price", color="red", alpha=0.5,␣
↪s=12)

# Set x and y limits
plt.xlim(0, 7)
plt.ylim(0, 400)

# Set x and y ticks
```

```
plt.xticks(range(0, 8))
plt.yticks(np.arange(0, 401, 50))

# Set axis labels
plt.xlabel("Distance to city center (miles)")
plt.ylabel("Price (US dollars)")

# Show the plot
plt.show()
```



1.2.1 Binary Variable

Close vs Far away hotels with a binary variable: - if further away from 2 miles, consider as ‘far’, otherwise ‘close’

[8]:
```
hotels["dist2"] = np.where(hotels["distance"] >= 2, "Far", "Close")
hotels["Eprice_cat2"] = hotels.groupby("dist2")["price"].transform("mean")
```

[ ]:
```
```

[9]:
```
hotels
```

[9]:

```
    country city_actual  rating_count center1label center2label  \
1   Austria      Vienna         189.0  City centre    Donauturm
2   Austria      Vienna          53.0  City centre    Donauturm
3   Austria      Vienna          55.0  City centre    Donauturm
4   Austria      Vienna          33.0  City centre    Donauturm
6   Austria      Vienna          57.0  City centre    Donauturm
..      …           …           …           …            …
420 Austria      Vienna          77.0  City centre    Donauturm
421 Austria      Vienna         572.0  City centre    Donauturm
422 Austria      Vienna          53.0  City centre    Donauturm
425 Austria      Vienna         112.0  City centre    Donauturm
426 Austria      Vienna         169.0  City centre    Donauturm

   neighbourhood  price    city  stars  ratingta  ratingta_count  \
1     17. Hernals     81  Vienna    4.0       3.5           708.0
2      Alsergrund     85  Vienna    4.0       3.5           629.0
3      Alsergrund     83  Vienna    3.0       4.0            52.0
4      Alsergrund     82  Vienna    4.0       3.5           219.0
6      Alsergrund    103  Vienna    4.0       3.5           251.0
..            …      …     …     …         …              …
420        Wieden    100  Vienna    3.0       4.0           149.0
421        Wieden     95  Vienna    4.0       4.0          1003.0
422        Wieden     73  Vienna    3.0       3.0           293.0
425        Wieden    100  Vienna    4.0       4.5          1006.0
426        Wieden     58  Vienna    3.0       3.0           135.0

    scarce_room  hotel_id  offer     offer_cat  year  month  weekend  \
1             0     21897      1   1-15% offer  2017     11        0
2             0     21901      1  15-50% offer  2017     11        0
3             0     21902      1  15-50% offer  2017     11        0
4             1     21903      1  15-50% offer  2017     11        0
6             1     21906      0   0% no offer  2017     11        0
..          …         …     …           …     …     …        …
420           1     22401      1   1-15% offer  2017     11        0
421           1     22402      1   1-15% offer  2017     11        0
422           1     22403      1   1-15% offer  2017     11        0
425           1     22407      0   0% no offer  2017     11        0
426           0     22408      1  15-50% offer  2017     11        0

    holiday  distance  distance_alter accommodation_type  nnights  rating  \
1         0       1.7             3.8              Hotel        1     3.9
2         0       1.4             2.5              Hotel        1     3.7
3         0       1.7             2.5              Hotel        1     4.0
4         0       1.2             2.8              Hotel        1     3.9
6         0       0.9             2.4              Hotel        1     3.9
..      …        …              …                  …      …      …
420       0       1.2             3.7              Hotel        1     4.0
```

```
421        0      1.5             3.9              Hotel         1      4.1
422        0      1.5             4.0              Hotel         1      3.4
425        0      1.0             3.7              Hotel         1      4.4
426        0      1.4             4.1              Hotel         1      3.2

     dist2  Eprice_cat2
1    Close   116.426752
2    Close   116.426752
3    Close   116.426752
4    Close   116.426752
6    Close   116.426752
..     …            …
420  Close   116.426752
421  Close   116.426752
422  Close   116.426752
425  Close   116.426752
426  Close   116.426752

[207 rows x 26 columns]
```

```
[ ]:
```

```
[ ]:
```

The first line of code is creating a new column named dist2, which is calculated using the `NumPy` `where()` function. The `where()` function takes three arguments: a *boolean* condition, a value to use if the condition is *true*, and a value to use if the condition is *false*. In this case, the boolean condition is whether the distance column is greater than or equal to 2. If the condition is true, the value in the new `dist2` column will be "Far". If the condition is false (i.e., the distance is less than 2), the value in the new `dist2` column will be "Close".

Alternative you could achive the same thing by doing this:

```
hotels["dist2"] = hotels["distance"].apply(lambda x: "Far" if x >= 2 else
"Close")
```

OR

```
hotels["dist2"] = "Close"
```

```
hotels.loc[hotels["distance"] >= 2, "dist2"] = "Far"
```

The second line of code is creating a new column named `Eprice_cat2`, which is the mean price value grouped by the `dist2` column. The `groupby()` function is used to group the DataFrame by the values in the `dist2` column, and the `transform()` method is used to apply the `mean()` function to the price column within each group. The result is a new column with the mean price value for each group (i.e., "Close" and "Far").

Overall, this code is creating two new columns in the DataFrame that categorize the distance between each hotel and the city center as "Close" or "Far", and then calculates the mean price value for each category.

[10]:
```
hotels.melt(id_vars="dist2", value_vars=["distance", "price"],␣
↪value_name="price_value")
```

[10]:
```
    dist2  variable  price_value
0   Close  distance          1.7
1   Close  distance          1.4
2   Close  distance          1.7
3   Close  distance          1.2
4   Close  distance          0.9
..    …        …           …
409 Close     price        100.0
410 Close     price         95.0
411 Close     price         73.0
412 Close     price        100.0
413 Close     price         58.0

[414 rows x 3 columns]
```

Check the descriptives for the two categories:

[11]:
```
(
    hotels.melt(id_vars="dist2", value_vars=["distance", "price"],␣
↪value_name="price_value")
    .groupby(["dist2", "variable"])
    .agg(["mean", "std", "min", "max", "count"])
    .round(2)
)
```

[11]:
```
               price_value
                      mean    std   min    max count
dist2 variable
Close distance        0.99   0.54   0.0    1.9   157
      price         116.43  43.10  54.0  383.0   157
Far   distance        3.21   0.97   2.0    6.6    50
      price          89.72  32.09  50.0  208.0    50
```

The code is performing a series of operations on the `hotels` DataFrame, including:

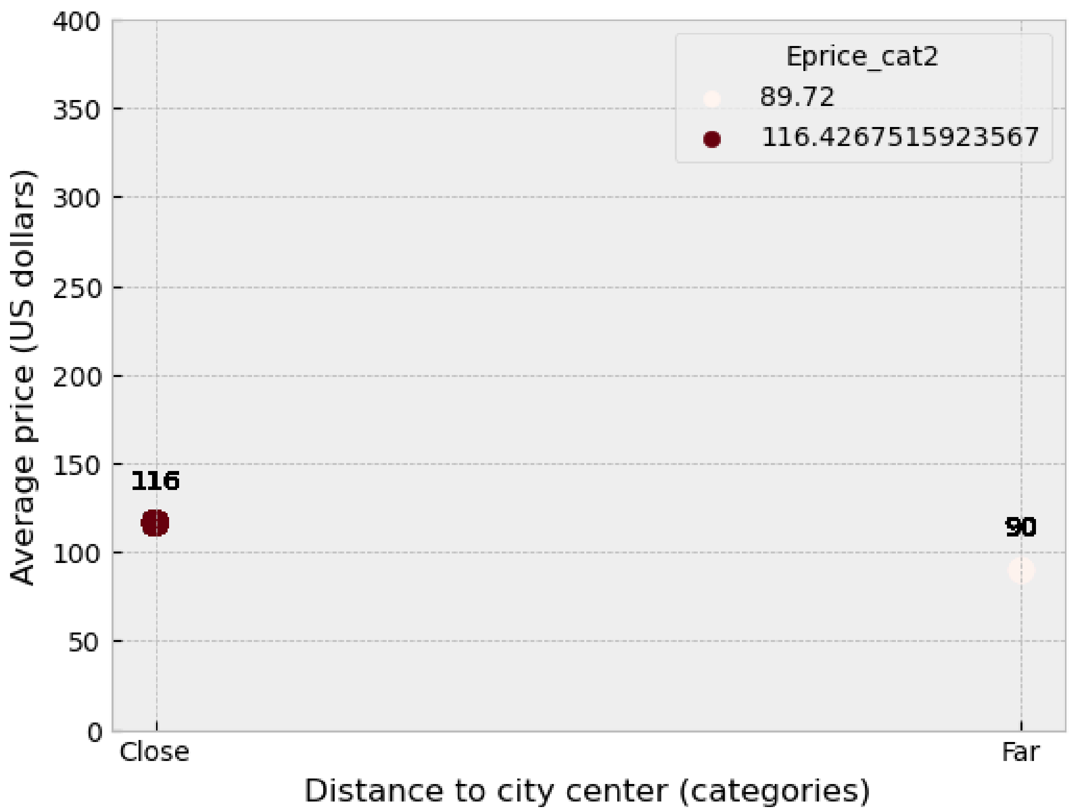
1. Using the `melt()` function to reshape the DataFrame so that the columns `distance` and `price` are "melted" into a single column named `value`, and a new column named `variable` is created to indicate the original column name of each value. The `id_vars` parameter is set to "dist2" to indicate that the `dist2` column should be kept as is.
2. Grouping the melted DataFrame by the `dist2` and `variable` columns using the `groupby()` method.
3. Aggregating the groups using the `agg()` method, which calculates several statistics for each group: mean, standard deviation, minimum, maximum, and count. These statistics are calculated for each of the original columns `distance` and `price` (now represented by the `variable` column).

4. Rounding the result to two decimal places using the `round()` method.

The result of these operations is a new DataFrame that shows the mean, standard deviation, minimum, maximum, and count of the `distance` and `price` variables for each `dist2` category. By grouping the data this way and calculating summary statistics, this code provides a way to compare the two categories based on these variables.

Plot the two categories

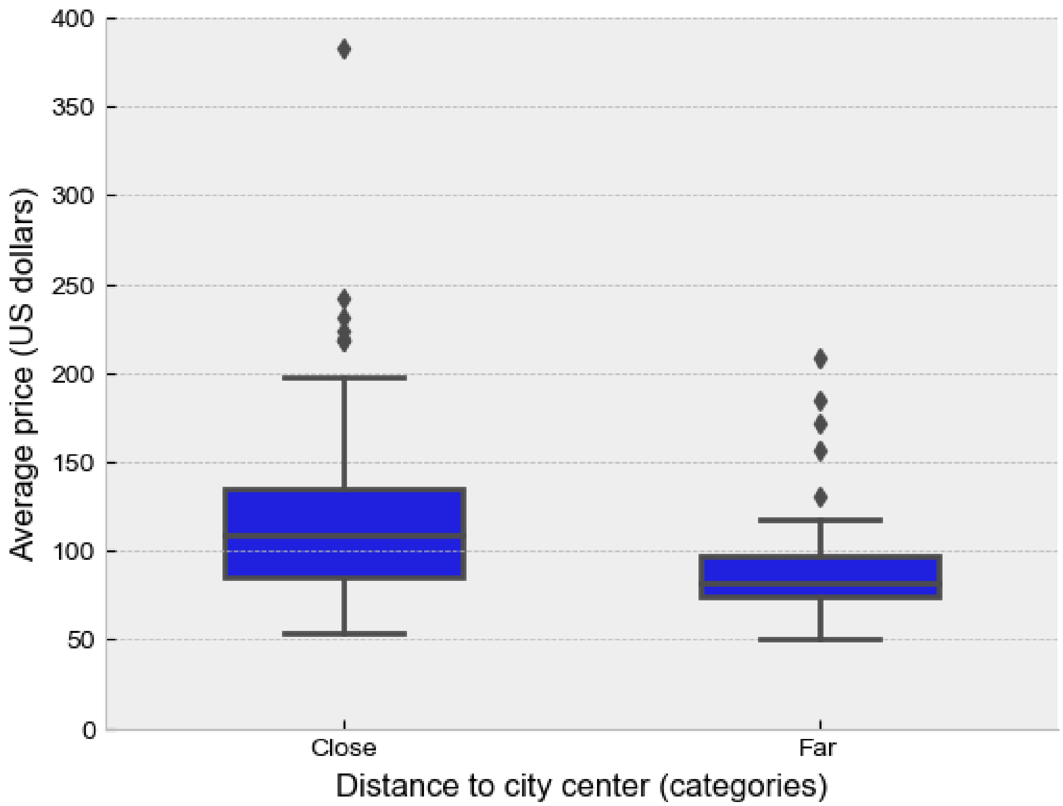
[12]:
```
# Create scatterplot with colored markers
sns.scatterplot(
    data=hotels,
    x="dist2",
    y="Eprice_cat2",
    hue="Eprice_cat2",
    palette="Reds",
    alpha=0.4,
    edgecolor="none",
    s=100,
)

# Add labels to markers
for x, y, val in zip(hotels["dist2"], hotels["Eprice_cat2"],␣
 ↪hotels["Eprice_cat2"].round()):
    plt.text(x, y+16, int(val), ha="center", va="bottom", fontsize=10)

# Set axis limits and labels
plt.ylim(0, 400)
plt.yticks(np.arange(0, 401, 50))
plt.xlabel("Distance to city center (categories)")
plt.ylabel("Average price (US dollars)")

# Show plot
plt.show()
```



Task: Instead of a simple dot, use a box-plot, which shows the underlying (conditional) distribution better!

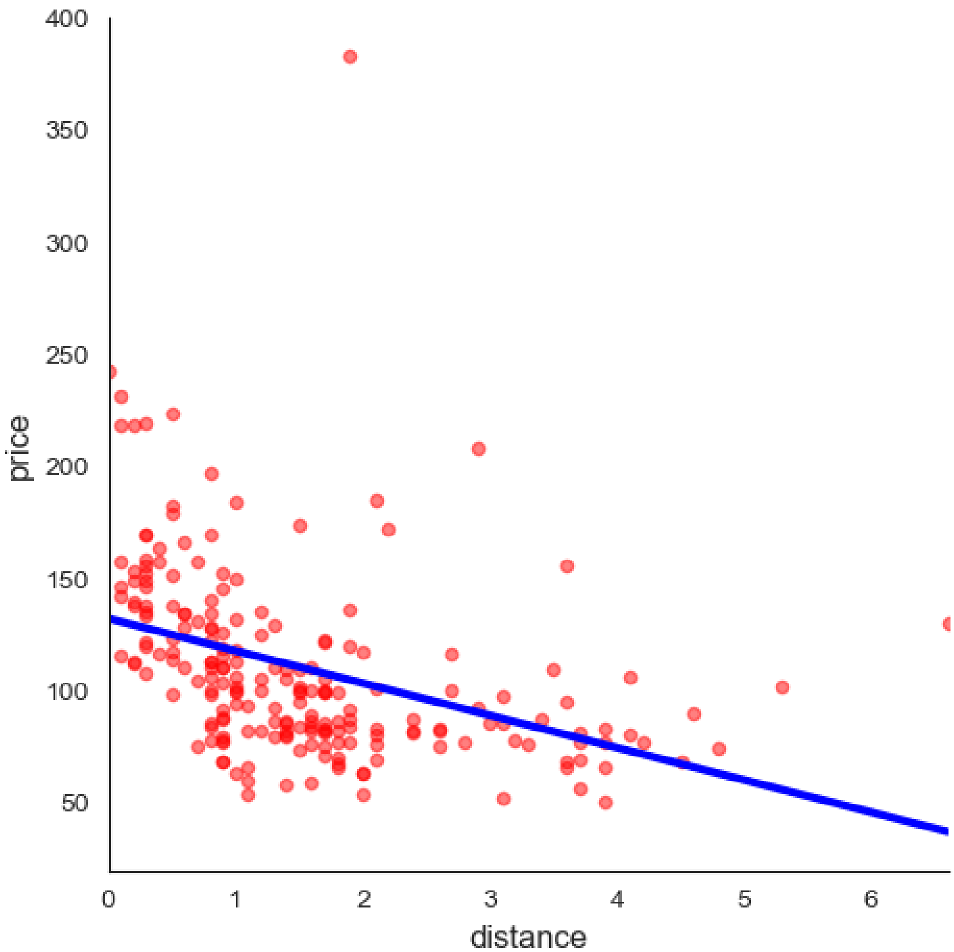
[13]:

```
sns.boxplot(data=hotels, x="dist2", y="price", color="blue", width=0.5)
#sns.errorbar(data=hotels, x="dist2", y="price", color="blue", fmt='none',␣
↪capsize=0.5)
sns.despine()
#
sns.set_style("white")
sns.set_palette("pastel")
#sns.set_context("talk")
plt.ylim(0, 400)
plt.yticks(np.arange(0, 401, 50))
plt.xlabel("Distance to city center (categories)")
plt.ylabel("Average price (US dollars)")
plt.show()
```



1.2.2 Simple Linear Regression

[14]:
```
sns.lmplot(x="distance", y="price", data=hotels, scatter_kws={"color": "red",␣
 ↪"alpha": 0.5, "s": 20}, line_kws={"color": "blue"}, ci=None)
plt.show()
```



This code generates a scatter plot with a linear regression line using the seaborn library (`sns`) and matplotlib (`plt`). Here's what each part of the code does:

- `sns.lmplot()`: This function creates a scatter plot with a linear regression line fit to the data. It takes several parameters:
  - `x="distance"`: Specifies the column from the DataFrame `hotels` to be used as the x-axis variable.
  - `y="price"`: Specifies the column from the DataFrame `hotels` to be used as the y-axis variable.
  - `data=hotels`: Specifies the DataFrame containing the data to be plotted.
  - `scatter_kws={"color": "red", "alpha": 0.5, "s": 20}`: Keyword arguments (`scatter_kws`) to customize the appearance of the scatter points. Here:
    * `"color": "red"`: Sets the color of the scatter points to red.
    * `"alpha": 0.5`: Sets the transparency of the scatter points to 0.5, making them semi-transparent.
    * `"s": 20`: Sets the size of the scatter points to 20.
  - `line_kws={"color": "blue"}`: Keyword arguments (`line_kws`) to customize the appearance of the regression line. Here:

    * "color": "blue": Sets the color of the regression line to blue.
  - ci=None: Specifies that no confidence interval should be shown around the regression line.
- plt.show(): This function displays the plot. It's necessary to call this function after creating the plot to actually show it on the screen.

In summary, the code creates a scatter plot of "distance" versus "price" from the hotels DataFrame, adds a linear regression line to the plot, customizes the appearance of the scatter points and regression line, and then displays the plot.

How to quantify linear regression:
Remember: $y = \alpha + \beta * x + \epsilon$

In Python, the statsmodels package is usually used to estimate regressions

[15]:
```
import statsmodels.formula.api as smf
#from mizani.formatters import percent_format
```

First we are going to run the following:

$price = \alpha + \beta * (distance) + \epsilon$

We use the statsmodels formula api, where you can give the equations as a string

Simple model, with homoskedastic SE

[16]:
```
simple_reg = smf.ols("price ~ distance", data=hotels).fit()
print(simple_reg.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.157
Model:                            OLS   Adj. R-squared:                  0.153
Method:                 Least Squares   F-statistic:                     38.20
Date:                Fri, 03 May 2024   Prob (F-statistic):           3.39e-09
Time:                        16:24:25   Log-Likelihood:                -1050.3
No. Observations:                 207   AIC:                             2105.
Df Residuals:                     205   BIC:                             2111.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    132.0170      4.474     29.511      0.000     123.197     140.837
distance     -14.4064      2.331     -6.181      0.000     -19.002      -9.811
==============================================================================
Omnibus:                      141.994   Durbin-Watson:                   1.479
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1560.025
Skew:                           2.497   Prob(JB):                         0.00
Kurtosis:                      15.488   Cond. No.                         3.78
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

The code performs simple linear regression between the variables `price` and `distance` using the `ols()` function from the `statsmodels.formula.api` module, which provides a formula-based interface to linear regression.

Specifically, it specifies a linear model with `price` as the dependent variable and `distance` as the independent variable. The `.fit()` method is then called on the model to fit it to the data contained in the `hotels` dataframe.

Finally, the `.summary()` method is used to print a summary of the regression results, which includes important information such as the R-squared value, the coefficients and their standard errors, and p-values for hypothesis testing.

Simple model, with heteroskedastic robust SE

[17]:
```
hetero_rob_reg = smf.ols("price ~ distance", data=hotels).fit(cov_type ="HC3")
print(hetero_rob_reg.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.157
Model:                            OLS   Adj. R-squared:                  0.153
Method:                 Least Squares   F-statistic:                     28.10
Date:                Fri, 03 May 2024   Prob (F-statistic):           2.97e-07
Time:                        16:24:25   Log-Likelihood:                -1050.3
No. Observations:                 207   AIC:                             2105.
Df Residuals:                     205   BIC:                             2111.
Df Model:                           1
Covariance Type:                  HC3
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    132.0170      4.876     27.072      0.000     122.459     141.575
distance     -14.4064      2.718     -5.301      0.000     -19.733      -9.080
==============================================================================
Omnibus:                      141.994   Durbin-Watson:                   1.479
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1560.025
Skew:                           2.497   Prob(JB):                         0.00
Kurtosis:                      15.488   Cond. No.                         3.78
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC3)
```

The `cov_type` parameter is set to `"HC3"`, which indicates that we want to estimate the covariance matrix using the Huber-White estimator with small sample correction.

Its wasy to compare two regression output tables using the stargazer package

[18]:
```
#!pip install stargazer
from stargazer.stargazer import Stargazer
```

If you do not have the stargazer package. You have to go to your terminal or command prompt and type. `pip install stargazer`. This should definitely work for macbooks!

[ ]:
```

```

[ ]:
```

```

[19]:
```
table = Stargazer([simple_reg, hetero_rob_reg])

table.rename_covariates({"Intercept": "Constant"})
table.custom_columns(["Homoskedastic SE", "Heteroskedastic robust SE"], [1, 1])
table
```

[19]:
```
<stargazer.stargazer.Stargazer at 0x7f8be936f340>
```

[20]:
```
html = table.render_html()

with open("my_regression_table.html", "w") as f:
    f.write(html)

#open('lin_reg.html','w').write(table.render_html())
```

You can use either the one I implement here or the one commented out. If you want to now use this table in MS word or docx files. You just have to open the `my_regression_table.html` file an copy paste it! You will see it should be formated like a table in word.

1.2.3 Analysis of the Results

- price prediction of a model
- errors of predictions

It is easy to save the predicted values and residuals

[21]:
```
hotels["predprice"] = simple_reg.fittedvalues
hotels["e"] = simple_reg.resid
```

Get the hotel, which is the most underpriced

[22]:
```
hotels.sort_values(by="e").head(5)
```

[22]:
```
     country city_actual  rating_count center1label center2label  \
153  Austria      Vienna          63.0  City centre    Donauturm
10   Austria      Vienna         203.0  City centre    Donauturm
211  Austria      Vienna         242.0  City centre    Donauturm
```

```
426  Austria      Vienna         169.0  City centre    Donauturm
163  Austria      Vienna          43.0  City centre    Donauturm

    neighbourhood  price    city  stars  ratingta  ratingta_count  \
153    Josefstadt     54  Vienna    3.0       3.0            85.0
10     Alsergrund     60  Vienna    4.0       4.0           359.0
211  Leopoldstadt     63  Vienna    3.0       3.0           183.0
426        Wieden     58  Vienna    3.0       3.0           135.0
163    Josefstadt     68  Vienna    3.0       3.5           182.0

     scarce_room  hotel_id  offer     offer_cat  year  month  weekend  \
153            1     22080      1  15-50% offer  2017     11        0
10             1     21912      1   1-15% offer  2017     11        0
211            1     22152      1   1-15% offer  2017     11        0
426            0     22408      1  15-50% offer  2017     11        0
163            1     22090      1  15-50% offer  2017     11        0

     holiday  distance  distance_alter accommodation_type  nnights  rating  \
153        0       1.1             3.4              Hotel        1     3.2
10         0       1.1             2.7              Hotel        1     4.1
211        0       1.0             1.9              Hotel        1     3.4
426        0       1.4             4.1              Hotel        1     3.2
163        0       0.9             3.2              Hotel        1     3.7

     dist2  Eprice_cat2   predprice          e
153  Close   116.426752  116.169910 -62.169910
10   Close   116.426752  116.169910 -56.169910
211  Close   116.426752  117.610552 -54.610552
426  Close   116.426752  111.847984 -53.847984
163  Close   116.426752  119.051194 -51.051194
```

probably we are only interested in hotel_id, distance, price, prediction and error values:

[23]:
```
hotels.sort_values(by="e").head(1).
 ↪filter(["hotel_id","distance","price","predprice","e"])
```

[23]:
```
     hotel_id  distance  price  predprice         e
153     22080       1.1     54  116.16991 -62.16991
```

Interpret the result!

We can get the 5 most overpriced five hotels

[24]:
```
hotels.sort_values(by="e", ascending=False).head(5).filter(
    ["hotel_id", "distance", "price", "predprice", "e"]
)
```

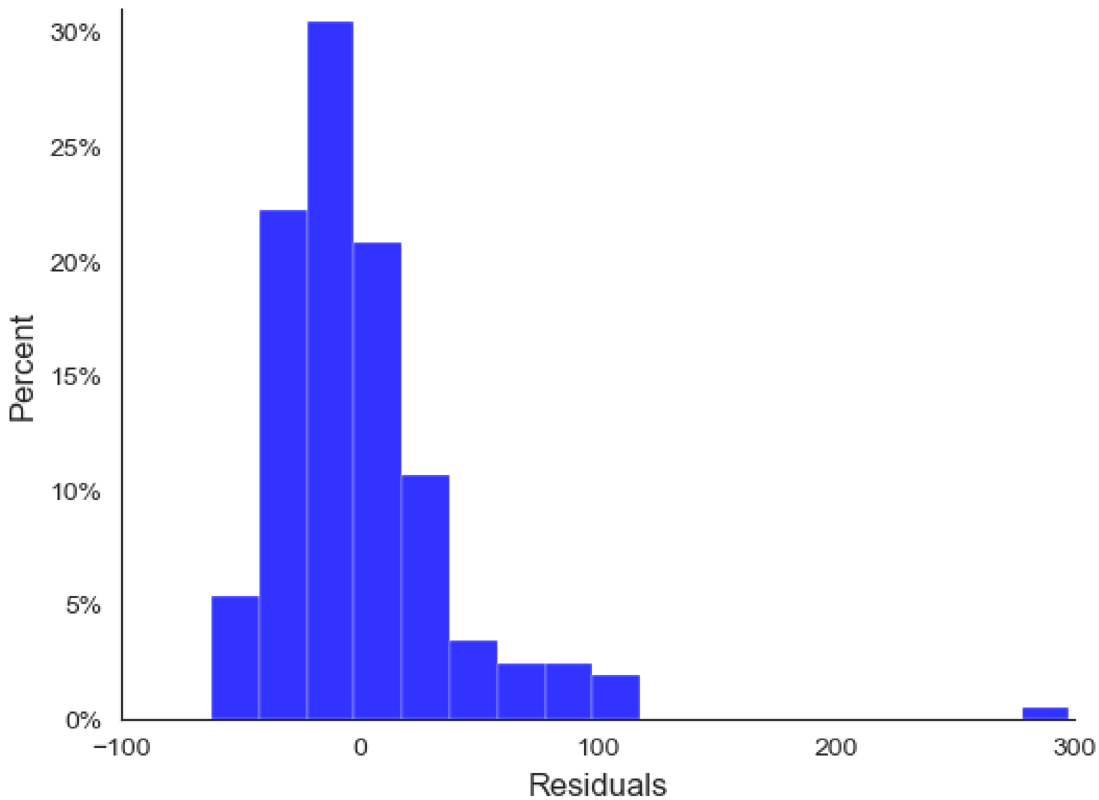
[24]:

	hotel_id	distance	price	predprice	e
247	22193	1.9	383	104.644774	278.355226
26	21930	2.9	208	90.238353	117.761647
128	22050	0.0	242	132.016973	109.983027
110	22031	0.1	231	130.576331	100.423669
129	22051	0.5	223	124.813762	98.186238

Checking the histogram of residuals:
we can better understand about how well we can predict the prices
notes:
- we picked previously the smallest and 5 largest values from here - on average we will have 0 error, as this is a property of the OLS estimator

[25]:

```
sns.histplot(data=hotels, x="e", binwidth=20, stat="probability", color="blue",␣
 ↪alpha=0.8, edgecolor="white", linewidth=0.2)
sns.despine()
#sns.set_style("white")
sns.set(rc={"figure.figsize":(6,4)})
sns.set_palette("deep")

plt.xlabel("Residuals")
plt.ylabel("Percent")
plt.xlim(-100, 300)
plt.ylim(0, 0.31)
plt.xticks(np.arange(-100, 301, 100))
plt.yticks(np.arange(0, 0.31, 0.05), labels=[f"{x:.0%}" for x in np.arange(0, 0.
 ↪31, 0.05)])
#plt.tight_layout()
plt.show()
```



1.3 Log models

Take log price

[26]:
```
hotels["lnprice"] = np.log(hotels["price"])
```

Correct distance2 measure: no closer than 0.05km

[27]:
```
hotels["distance2"] = np.where(hotels["distance"] < 0.05, 0.05,␣
↪hotels["distance"])
```

Take the log of distance2

[28]:
```
hotels["lndistance"] = np.log(hotels["distance2"])
```

Describe price and ln price

[29]:
```
hotels.filter(["price", "lnprice"]).describe(percentiles=[0.25, 0.5, 0.75, 0.
↪95]).T
```

[29]:
```
        count        mean        std        min        25%        50%  \
price   207.0  109.975845  42.221381  50.000000  82.000000  100.00000
```

```
lnprice  207.0    4.640219   0.336751   3.912023   4.406719    4.60517

                75%         95%         max
price    129.500000  183.400000  383.000000
lnprice    4.863673    5.211657    5.948035
```

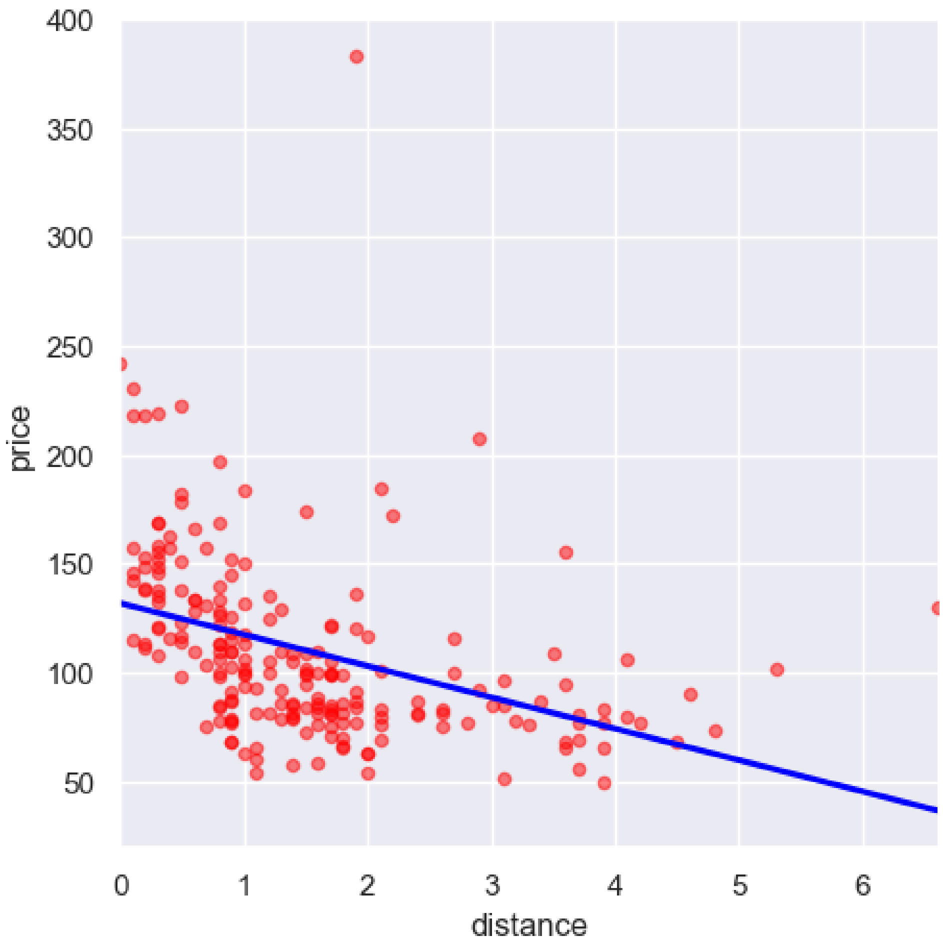
1.3.1 Running multiple regressions:

1. Level-level linear regression

[30]:
```python
reg1 = smf.ols("price ~ distance", data=hotels).fit()
print(reg1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.157
Model:                            OLS   Adj. R-squared:                  0.153
Method:                 Least Squares   F-statistic:                     38.20
Date:                Fri, 03 May 2024   Prob (F-statistic):           3.39e-09
Time:                        16:24:25   Log-Likelihood:                -1050.3
No. Observations:                 207   AIC:                             2105.
Df Residuals:                     205   BIC:                             2111.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    132.0170      4.474     29.511      0.000     123.197     140.837
distance     -14.4064      2.331     -6.181      0.000     -19.002      -9.811
==============================================================================
Omnibus:                      141.994   Durbin-Watson:                   1.479
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1560.025
Skew:                           2.497   Prob(JB):                         0.00
Kurtosis:                      15.488   Cond. No.                         3.78
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[31]:
```python
sns.lmplot(x="distance", y="price", data=hotels, scatter_kws={"color": "red",␣
↪"alpha": 0.5, "s": 20}, line_kws={"color": "blue"}, ci=None)

plt.show()
```



2. Level-log linear regression

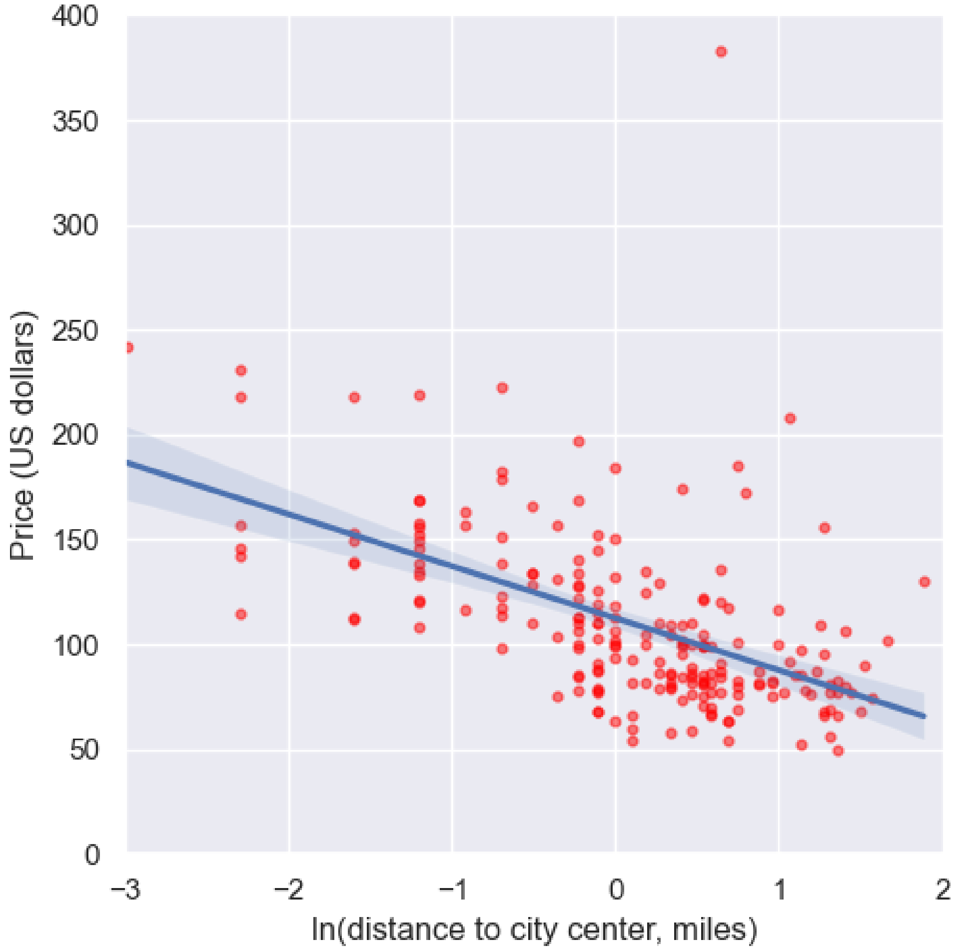
```
[32]: reg2 = smf.ols("price ~ lndistance", data=hotels).fit()
      print(reg2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.280
Model:                            OLS   Adj. R-squared:                  0.276
Method:                 Least Squares   F-statistic:                     79.58
Date:                Fri, 03 May 2024   Prob (F-statistic):           2.61e-16
Time:                        16:24:26   Log-Likelihood:                -1034.1
No. Observations:                 207   AIC:                             2072.
Df Residuals:                     205   BIC:                             2079.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
```

```
==============================================================================
Intercept     112.4171      2.512     44.757      0.000     107.465     117.369
lndistance    -24.7683      2.777     -8.921      0.000     -30.243     -19.294
==============================================================================
Omnibus:                      175.079   Durbin-Watson:                   1.612
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3501.545
Skew:                           3.084   Prob(JB):                         0.00
Kurtosis:                      22.182   Cond. No.                         1.16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[33]:

```
sns.lmplot(x="lndistance", y="price", data=hotels, scatter_kws={"color": "red",␣
 ↪"alpha": 0.5, "s": 12})
plt.xlim(-3, 2)
plt.ylim(0, 400)
plt.xlabel("ln(distance to city center, miles)")
plt.ylabel("Price (US dollars)")
plt.show()
```



3. Log-level linear regression

[34]:
```
reg3 = smf.ols("lnprice ~ distance", data=hotels).fit()
print(reg3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lnprice   R-squared:                       0.205
Model:                            OLS   Adj. R-squared:                  0.201
Method:                 Least Squares   F-statistic:                     52.90
Date:                Fri, 03 May 2024   Prob (F-statistic):           7.30e-12
Time:                        16:24:26   Log-Likelihood:                -44.160
No. Observations:                 207   AIC:                             92.32
Df Residuals:                     205   BIC:                             98.99
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
```

```
==============================================================================
Intercept      4.8411      0.035    139.720      0.000       4.773       4.909
distance      -0.1313      0.018     -7.273      0.000      -0.167      -0.096
==============================================================================
Omnibus:                       28.470   Durbin-Watson:                   1.564
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               47.450
Skew:                           0.746   Prob(JB):                     4.97e-11
Kurtosis:                       4.809   Cond. No.                         3.78
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

4. Log-log linear regression

[35]:
```python
reg4 = smf.ols("lnprice ~ lndistance", data=hotels).fit()
print(reg4.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lnprice   R-squared:                       0.334
Model:                            OLS   Adj. R-squared:                  0.330
Method:                 Least Squares   F-statistic:                     102.6
Date:                Fri, 03 May 2024   Prob (F-statistic):           8.18e-20
Time:                        16:24:26   Log-Likelihood:                -25.911
No. Observations:                 207   AIC:                             55.82
Df Residuals:                     205   BIC:                             62.49
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      4.6615      0.019    241.926      0.000       4.623       4.699
lndistance    -0.2158      0.021    -10.130      0.000      -0.258      -0.174
==============================================================================
Omnibus:                       47.573   Durbin-Watson:                   1.742
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              128.794
Skew:                           0.976   Prob(JB):                     1.08e-28
Kurtosis:                       6.335   Cond. No.                         1.16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

Checking and comparing all models:

[36]:
```
table = Stargazer([reg1, reg2, reg3, reg4])
table.rename_covariates({"Intercept": "Constant"})
table.custom_columns(["Level-Level", "Level-Log", "Log-Level", "Log-Log"], [1,␣
↪1, 1, 1])
table
```

[36]:
```
<stargazer.stargazer.Stargazer at 0x7f8be86bd370>
```

1.4 Polynomials

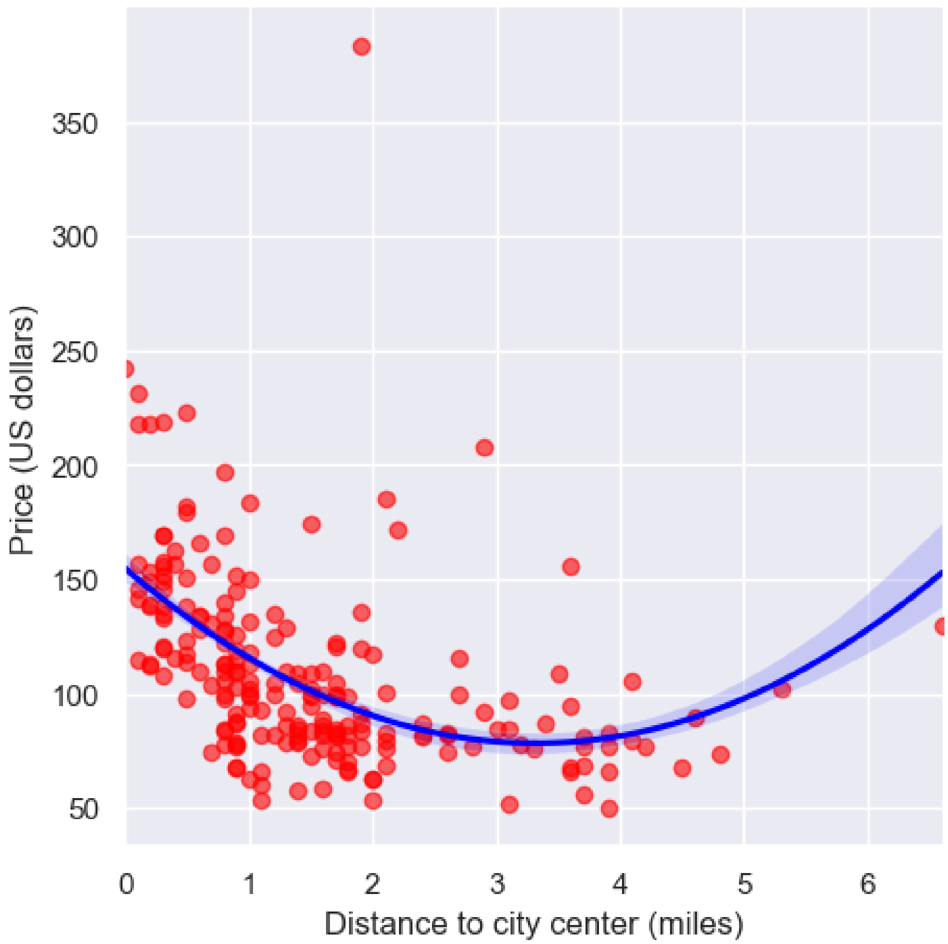
[37]:
```
hotels["dist_sq"] = hotels["distance"]**2
hotels["dist_cb"] = hotels["distance"]**3
```

5. Single squared

[38]:
```
reg5 = smf.ols("price ~ distance + dist_sq", data=hotels).fit()
print(reg5.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.258
Model:                            OLS   Adj. R-squared:                  0.251
Method:                 Least Squares   F-statistic:                     35.44
Date:                Fri, 03 May 2024   Prob (F-statistic):           6.18e-14
Time:                        16:24:26   Log-Likelihood:                -1037.1
No. Observations:                 207   AIC:                             2080.
Df Residuals:                     204   BIC:                             2090.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    154.8580      6.045     25.617      0.000     142.939     166.777
distance     -46.0140      6.394     -7.197      0.000     -58.620     -33.408
dist_sq        6.9277      1.316      5.263      0.000       4.332       9.523
==============================================================================
Omnibus:                      177.585   Durbin-Watson:                   1.551
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3582.408
Skew:                           3.149   Prob(JB):                         0.00
Kurtosis:                      22.382   Cond. No.                         23.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

```
[39]: sns.lmplot(x="distance", y="price", data=hotels, order=2, ci=68,␣
       ↪scatter_kws={"color": "red", "alpha": 0.6}, line_kws={"color": "blue"})

      plt.xlabel("Distance to city center (miles)")
      plt.ylabel("Price (US dollars)")
      plt.show()
```



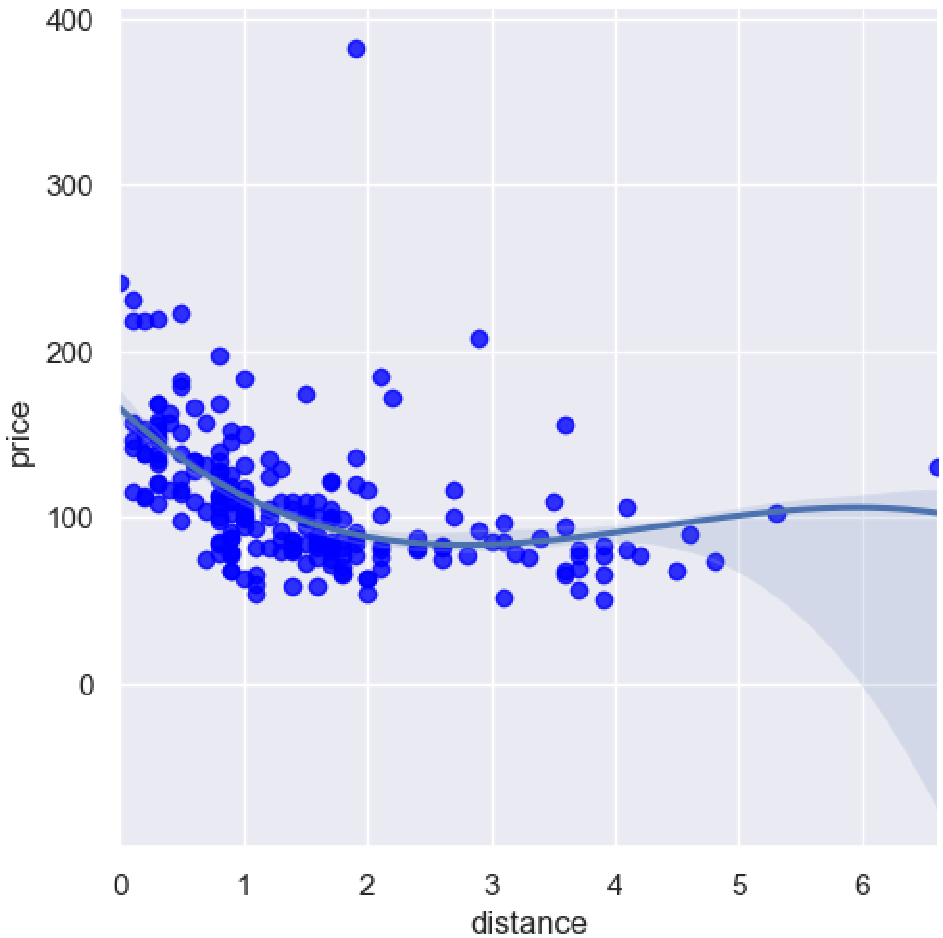
6. Squared and cubic

```
[40]: reg6 = smf.ols("price ~ distance + dist_sq + dist_cb", data=hotels).fit()
      print(reg6.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.276
Model:                            OLS   Adj. R-squared:                  0.265
Method:                 Least Squares   F-statistic:                     25.74
Date:                Fri, 03 May 2024   Prob (F-statistic):           3.75e-14
```

```
Time:                        16:24:26   Log-Likelihood:                -1034.6
No. Observations:                 207   AIC:                             2077.
Df Residuals:                     203   BIC:                             2091.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    165.6994      7.715     21.477      0.000     150.487     180.912
distance     -70.0162     12.496     -5.603      0.000     -94.656     -45.377
dist_sq       18.4468      5.332      3.460      0.001       7.933      28.960
dist_cb       -1.4070      0.632     -2.228      0.027      -2.652      -0.162
==============================================================================
Omnibus:                      184.317   Durbin-Watson:                   1.592
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4186.402
Skew:                           3.282   Prob(JB):                         0.00
Kurtosis:                      24.031   Cond. No.                         191.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[41]:
```
sns.lmplot(x="distance", y="price", data=hotels, ci=68, order=3,␣
 ↪scatter_kws={"color": "blue"})
plt.show()
```



Compare these non-linear models:

[42]:
```
table = Stargazer([reg1, reg5, reg6])
table.rename_covariates({"Intercept": "Constant"})
table.custom_columns(["Linear", "Squared", "Cubic"], [1, 1, 1])
table
```

[42]: <stargazer.stargazer.Stargazer at 0x7f8be9384be0>

[ ]: