```
import pandas as pd
```

Los datos Boston Housing contienen información de registros de censo en Boston2 para los cuales varias mediciones son tomadas (por ejemplo, tasa de crimen, radio estudiante/docente).

Información sobre el dataset: https://www.kaggle.com/code/avk256/the-boston-housing-dataset

```
Boston=pd.read_csv("https://raw.githubusercontent.com/reisanar/datasets/master/BostonHousing.csv")
Boston
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	MEDV	CAT_MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0	0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6	0
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	9.67	22.4	0
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	9.08	20.6	0
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	5.64	23.9	0
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	6.48	22.0	0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	7.88	11.9	0

506 rows × 14 columns

CRIM Crime rate
ZN Percentage of residential land zoned for lots over 25,000 ft2
INDUS Percentage of land occupied by nonretail business
CHAS Does tract bound Charles River (= 1 if tract bounds river, = 0 otherwise)
NOX Nitric oxide concentration (parts per 10 million)
RM Average number of rooms per dwelling
AGE Percentage of owner-occupied units built prior to 1940
DIS Weighted distances to five Boston employment centers
RAD Index of accessibility to radial highways
TAX Full-value property tax rate per USD 10000
PTRATIO Pupil-to-teacher ratio by town
LSTAT Percentage of lower status of the population
MEDV Median value of owner-occupied homes in $1000s
CAT.MEDV Is median value of owner-occupied homes in tract above USD 30,000 (CAT.MEDV = 1) or not (CAT.MEDV = 0)

Amtrak, a US railway company, routinely collects data on ridership. Here we focus on forecasting future ridership using the series of monthly ridership between January 1991 and March 2004.

```
Amtrak=pd.read_csv("https://raw.githubusercontent.com/reisanar/datasets/master/Amtrak.csv")
Amtrak
```

	Month	Ridership
0	01/01/1991	1708.917
1	01/02/1991	1620.586
2	01/03/1991	1972.715
3	01/04/1991	1811.665
4	01/05/1991	1974.964
...	...	...
154	01/11/2003	2076.054
155	01/12/2003	2140.677
156	01/01/2004	1831.508
157	01/02/2004	1838.006
158	01/03/2004	2132.446

159 rows × 2 columns

```
## Load the Amtrak data and convert them to be suitable for time series analysis
Amtrak['Date'] = pd.to_datetime(Amtrak.Month, format='%d/%m/%Y')
ridership_ts = pd.Series(Amtrak.Ridership.values, index=Amtrak.Date)
print(ridership_ts)
print(Amtrak)
```

```
Date
1991-01-01    1708.917
1991-02-01    1620.586
1991-03-01    1972.715
1991-04-01    1811.665
1991-05-01    1974.964
                ...
2003-11-01    2076.054
2003-12-01    2140.677
2004-01-01    1831.508
2004-02-01    1838.006
2004-03-01    2132.446
Length: 159, dtype: float64
          Month  Ridership       Date
0    01/01/1991   1708.917 1991-01-01
1    01/02/1991   1620.586 1991-02-01
2    01/03/1991   1972.715 1991-03-01
3    01/04/1991   1811.665 1991-04-01
4    01/05/1991   1974.964 1991-05-01
..          ...        ...        ...
154  01/11/2003   2076.054 2003-11-01
155  01/12/2003   2140.677 2003-12-01
156  01/01/2004   1831.508 2004-01-01
157  01/02/2004   1838.006 2004-02-01
158  01/03/2004   2132.446 2004-03-01

[159 rows x 3 columns]
```

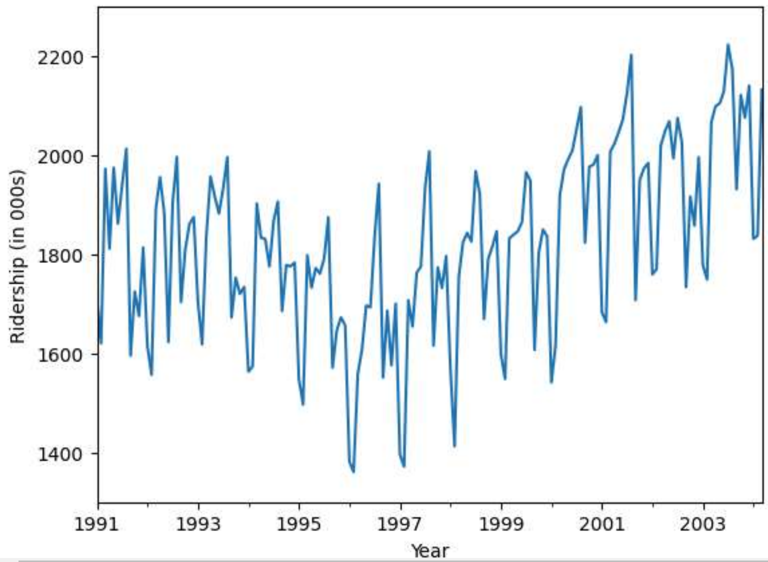
```
import matplotlib.pyplot as plt
```

Gráfico de Línea

Muestra una gráfica de línea para la serie de tiempo de pasajeros de tren de Amtrak durante el mes. Las gráficas de línea son utilizadas principalmente para mostrar series de tiempo.
La decisión del marco de tiempo a graficas, así como la escala temporal, deberían depender del horizonte de la tarea de predicción y de la naturaleza de los datos.

```
## line graph
ridership_ts.plot(ylim=[1300, 2300], legend=False)
plt.xlabel('Year') # set x-axis label
plt.ylabel('Ridership (in 000s)') # set y-axis label
```

```
Text(0, 0.5, 'Ridership (in 000s)')
```

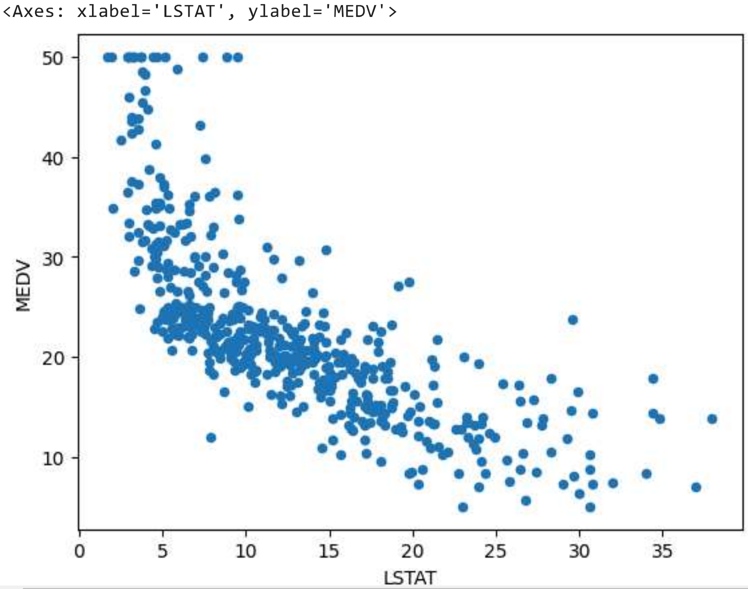


Gráfica de Dispersión

La variable de resultado está típicamente asociada con el eje y. En aprendizaje supervisado (para el propósito de reducción de datos o de clusterización), gráficos básicos que contienen relaciones (tales como gráficas de dispersión) son preferidas.
Para aprendizaje no supervisado, esta gráfica de dispersión particular, ayuda a estudiar la asociación entre dos variables numéricas en términos de superposición de información, así como identificar clusters de observaciones.

```
## scatter plot with axes names
Boston.plot.scatter(x='LSTAT', y='MEDV', legend=False)
```

```
<Axes: xlabel='LSTAT', ylabel='MEDV'>
```



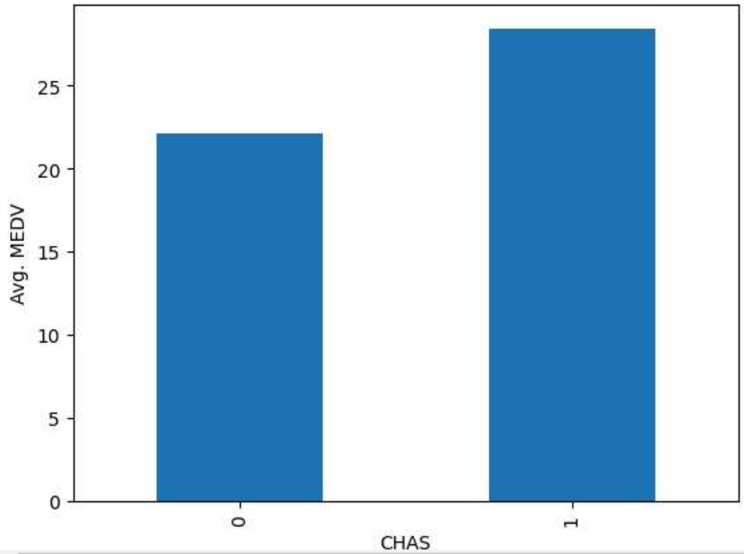
Gráficas de Barra

Son útiles para comparar una única estadística (por ejemplo, promedio, conteo, porcentaje) a lo largo de grupos. La altura de la barra (o la longitud, en una visualización horizontal) representa el valor de la estadística, y diferentes barras corresponden a diferentes grupos.

```
## barchart of CHAS vs. mean MEDV
# compute mean MEDV per CHAS = (0, 1)
```

```
ax = Boston.groupby('CHAS').mean().MEDV.plot(kind='bar')
ax.set_ylabel('Avg. MEDV')
```

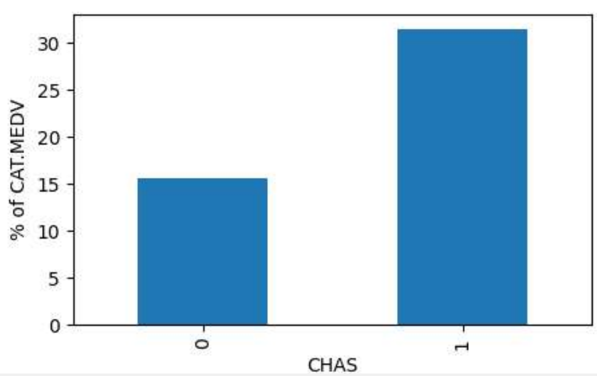
```
Text(0, 0.5, 'Avg. MEDV')
```



Esta gráfica nos muestra que las áreas bordeando el río Charles son más propensas a tener valores medios por encima de $30K.

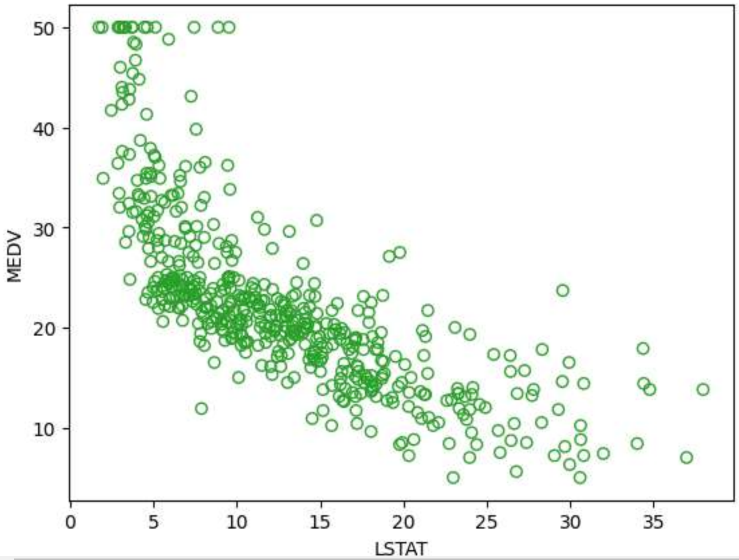
```
## barchart of CHAS vs. CAT_MEDV
dataForPlot = Boston.groupby('CHAS').mean()['CAT_MEDV'] * 100
ax = dataForPlot.plot(kind='bar', figsize=[5, 3])
ax.set_ylabel('% of CAT.MEDV')



#   CAT.MEDV   0    350
#   CAT.MEDV   1    156

# CHAS 0  50     156
# CHAS 1  106    156
```

```
Text(0, 0.5, '% of CAT.MEDV')
```



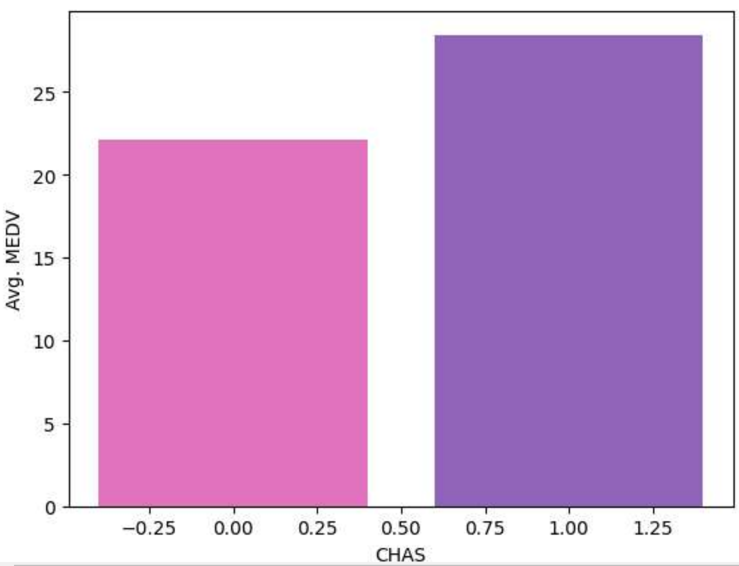

```
## Set the color of the points in the scatterplot and draw as open circles.
plt.scatter(Boston.LSTAT, Boston.MEDV, color='C2', facecolor='none')
plt.xlabel('LSTAT'); plt.ylabel('MEDV')
```

```
Text(0, 0.5, 'MEDV')
```



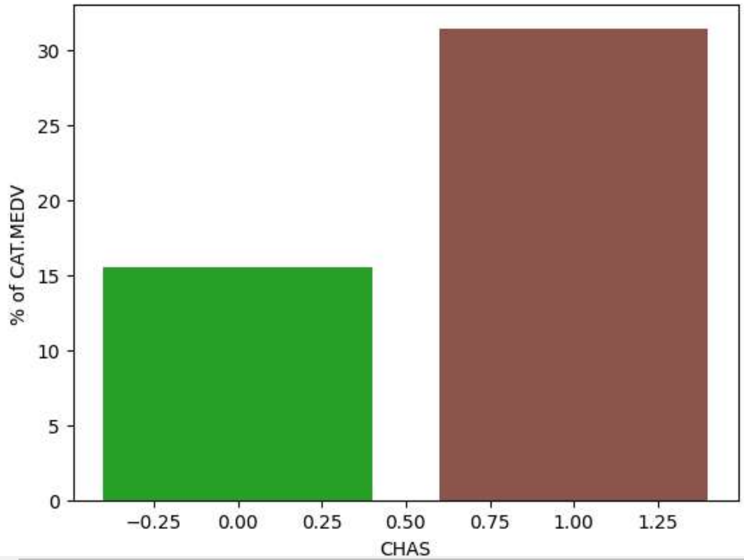

```
## barchart of CHAS vs. mean MEDV
# compute mean MEDV per CHAS = (0, 1)
dataForPlot = Boston.groupby('CHAS').mean().MEDV
fig, ax = plt.subplots()
ax.bar(dataForPlot.index, dataForPlot, color=['C6', 'C4'])
#ax.set_xticks([0, 1], False)
ax.set_xlabel('CHAS')
ax.set_ylabel('Avg. MEDV')
```

```
Text(0, 0.5, 'Avg. MEDV')
```




```
## barchart of CHAS vs. CAT.MEDV
dataForPlot = Boston.groupby('CHAS').mean()['CAT_MEDV'] * 100
fig, ax = plt.subplots()
ax.bar(dataForPlot.index, dataForPlot, color=['C2', 'C5'])
#ax.set_xticks([0, 1], False)
ax.set_xlabel('CHAS'); ax.set_ylabel('% of CAT.MEDV')
```

```
Text(0, 0.5, '% of CAT.MEDV')
```

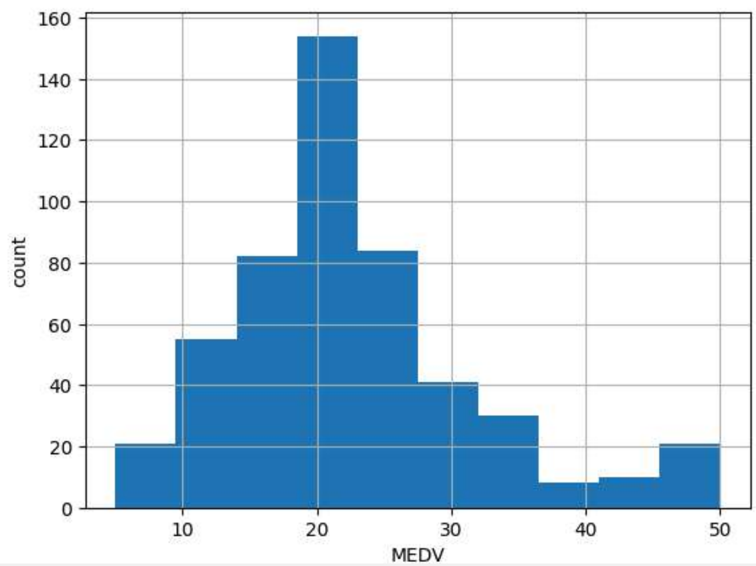


Histograma

Un histograma representa las frecuencias de todos los x valores en una serie de barras verticales conectadas. Por ejemplo, aquí se presentan más de 150 áreas donde el valor medio (MEDV) está entre $20K y $25K . Este histograma revela una distribución sesgada. Transformar la variable resultado a log(MEDV) podría mejorar los resultados de un predictor de regresión lineal.

```
## histogram of MEDV
ax = Boston.MEDV.hist()
ax.set_xlabel('MEDV'); ax.set_ylabel('count')
```

```
Text(0, 0.5, 'count')
```




```
# alternative plot with matplotlib
fig, ax = plt.subplots()
ax.hist(Boston.MEDV)
ax.set_axisbelow(True) # Show the grid lines behind the histogram
ax.grid(which='major', color='grey', linestyle='--')
ax.set_xlabel('MEDV'); ax.set_ylabel('count')
plt.show()
```

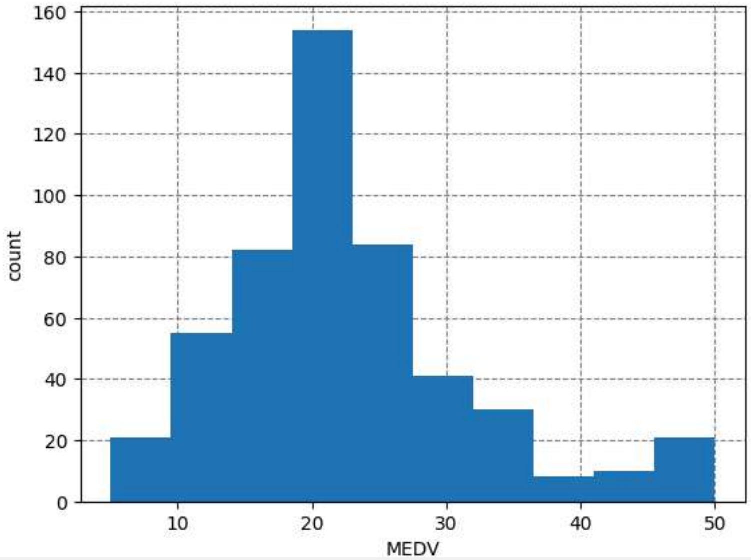
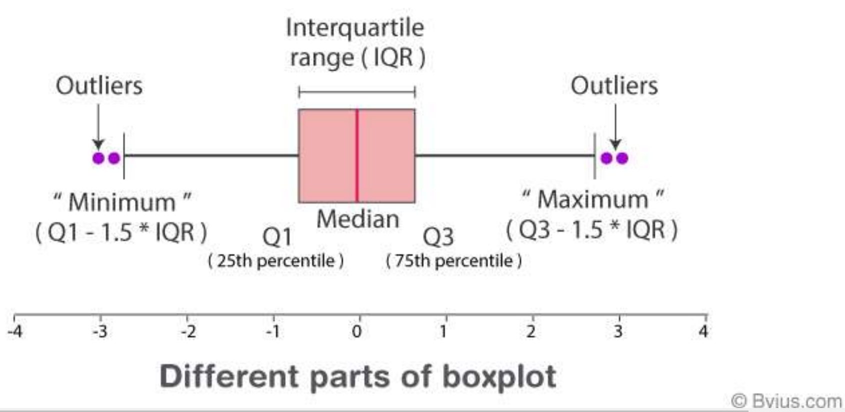


Diagrama de caja

```
from PIL import Image
import requests
url="https://cdn1.byjus.com/wp-content/uploads/2020/10/Box-Plot-and-Whisker-Plot-1.png"
Image.open(requests.get(url, stream=True).raw)
```

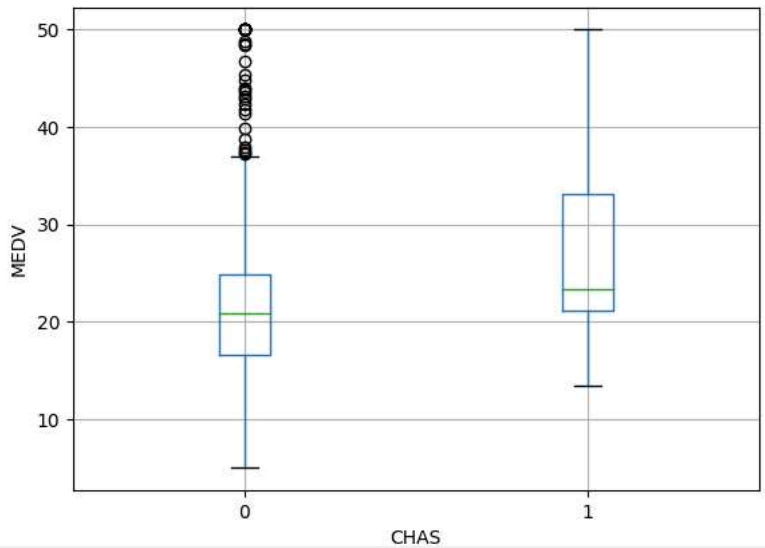


Un diagrama de caja representa la variable siendo graficada en el eje y (aunque la gráfica pueda potencialmente ser girada 90°, para que las cajas sean paralelas al eje x.

Hay dos diagramas de caja (llamados cajas lado a lado). Esta caja abarca el 50% de los datos, por ejemplo, en la caja de la derecha, la mitad de las áreas tienen valores medios (MEDV) entre $20,000 y $33,000. La línea horizontal dentro de la caja representa la media (50th percentil). El tope y el fondo de la caja representan los percentiles 75th y 25th percentiles, respectivamente. Las líneas que se extienden por encima y por debajo de la caja cubren el resto de los datos en el rango; los datos atípicos puedes estar representados como puntos o círculos. Algunas veces el promedio está marcado por un + (o signo similar).
Comparando el promedio y la mediana sirve para identificar el sesgo en los datos. Los diagramas de caja a menudo están organizados en series con una gráfica diferente para varios valores de una segunda variable, mostrada en el eje x.

```
## boxplot of MEDV for different values of CHAS
ax = Boston.boxplot(column='MEDV', by='CHAS')
ax.set_ylabel('MEDV')
plt.suptitle('') # Suppress the titles
plt.title('')
```

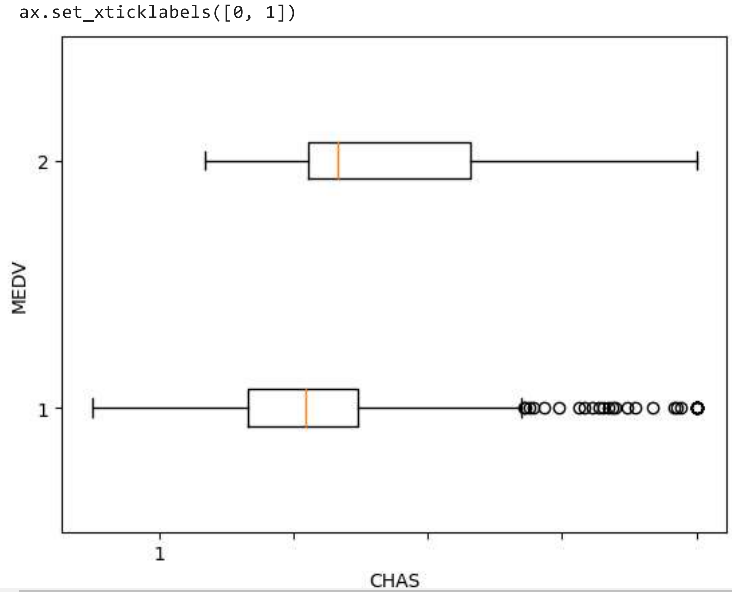
```
Text(0.5, 1.0, '')
```



Not only is the average MEDV for river-bounding homes higher than the non-river-bounding homes, but also the entire distribution is higher (median, quartiles, min, and max). We can also see that all riverbounding homes have MEDV above $10K, unlike non-river-bounding homes. This information is useful for identifying the potential importance of this predictor (CHAS).

```
#alternative plot with matplotlib
dataForPlot = [list(Boston[Boston.CHAS==0].MEDV),
list(Boston[Boston.CHAS==1].MEDV)]
fig, ax = plt.subplots()
ax.boxplot(dataForPlot,vert=False)
#ax.set_xticks([1, 2], False)
ax.set_xticklabels([0, 1])
ax.set_xlabel('CHAS'); ax.set_ylabel('MEDV')
plt.show()
```

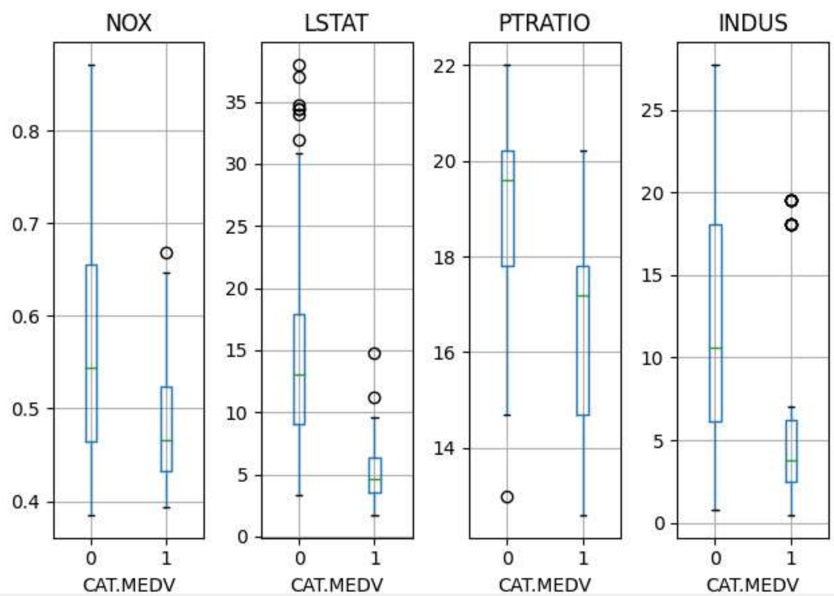
```
<ipython-input-17-6bcb6088e7f0>:7: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels([0, 1])
```



The main weakness of basic charts and distribution plots, in their basic form (i.e., using position in relation to the axes to encode values), is that they can only display two variables and therefore cannot reveal high-dimensional information. Each of the basic charts has two dimensions, where each dimension is dedicated to a single variable. In data mining, the data are usually multivariate by nature, and the analytics are designed to capture and measure multivariate information.

```
## side-by-side boxplots
fig, axes = plt.subplots(nrows=1, ncols=4)
```

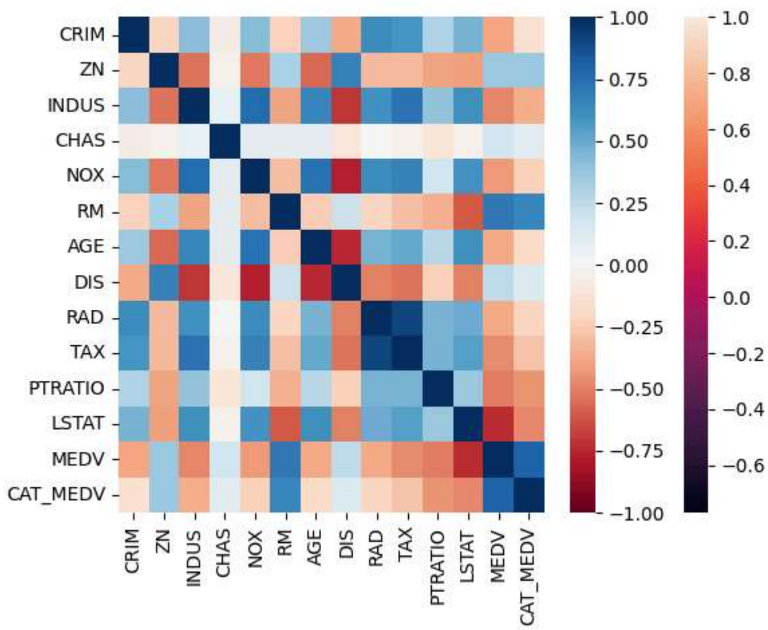
```
Boston.boxplot(column='NOX', by='CAT_MEDV', ax=axes[0])
Boston.boxplot(column='LSTAT', by='CAT_MEDV', ax=axes[1])
Boston.boxplot(column='PTRATIO', by='CAT_MEDV', ax=axes[2])
Boston.boxplot(column='INDUS', by='CAT_MEDV', ax=axes[3])
for ax in axes:
  ax.set_xlabel('CAT.MEDV')
plt.suptitle('') # Suppress the overall title
plt.tight_layout() # Increase the separation between the plots
```

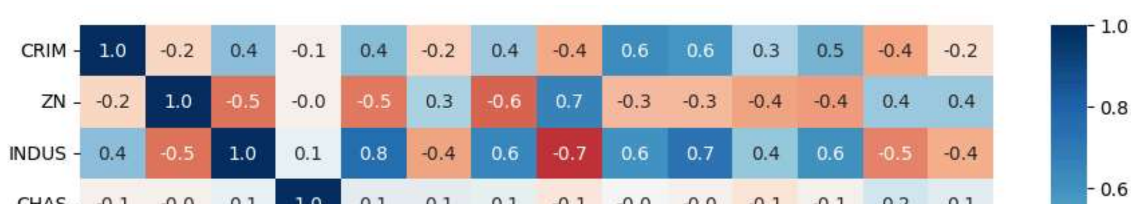


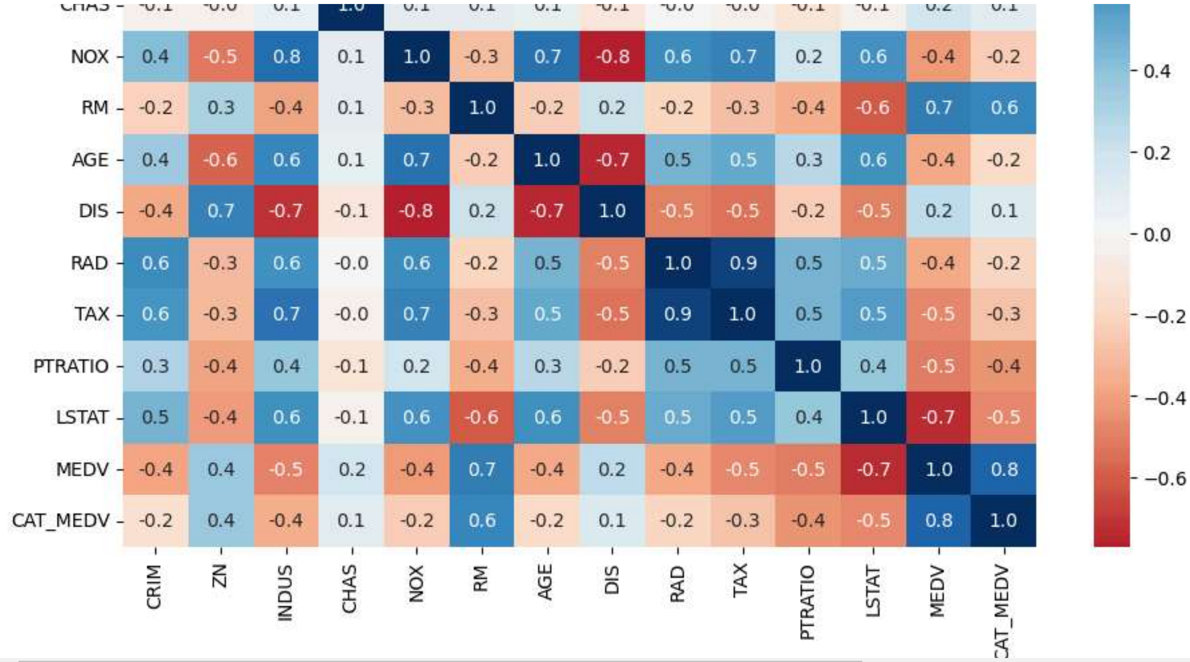
showing all the pairwise correlations between 13 variables (MEDV and 12 predictors). Darker shades correspond to stronger (positive or negative) correlation. It is easy to quickly spot the high and low correlations. The use of blue/red is used in this case to highlight positive vs. negative correlations.

```
import seaborn as sns

## simple heatmap of correlations (without values)
corr = Boston.corr()
print(corr)
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns)
# Change the colormap to a divergent scale and fix the range of the colormap
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, vmin=-1,
vmax=1, cmap="RdBu")
# Include information about values (example demonstrate how to control the size of
# the plot
fig, ax = plt.subplots()
fig.set_size_inches(11, 7)
sns.heatmap(corr, annot=True, fmt=".1f", cmap="RdBu", center=0, ax=ax)
```

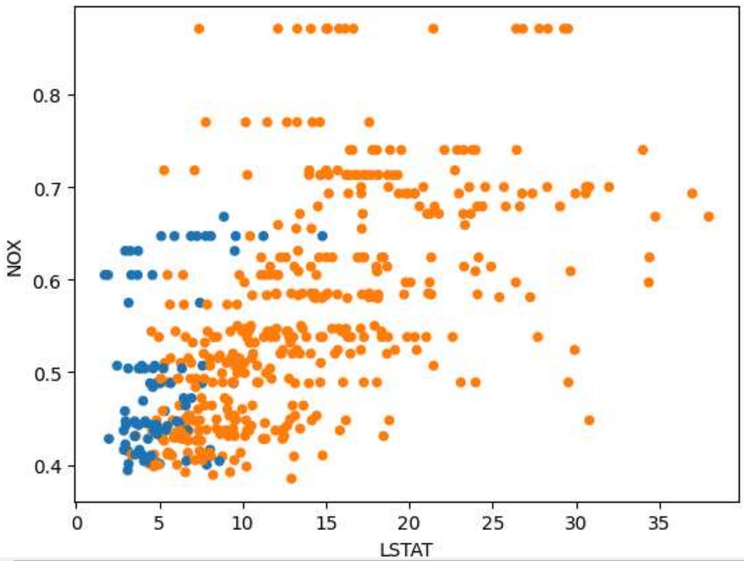
```
               CRIM        ZN     INDUS      CHAS       NOX        RM  \
CRIM      1.000000 -0.200469  0.406583 -0.055892  0.420972 -0.219247
ZN       -0.200469  1.000000 -0.533828 -0.042697 -0.516604  0.311991
INDUS     0.406583 -0.533828  1.000000  0.062938  0.763651 -0.391676
CHAS     -0.055892 -0.042697  0.062938  1.000000  0.091203  0.091251
NOX       0.420972 -0.516604  0.763651  0.091203  1.000000 -0.302188
RM       -0.219247  0.311991 -0.391676  0.091251 -0.302188  1.000000
AGE       0.352734 -0.569537  0.644779  0.086518  0.731470 -0.240265
DIS      -0.379670  0.664408 -0.708027 -0.099176 -0.769230  0.205246
RAD       0.625505 -0.311948  0.595129 -0.007368  0.611441 -0.209847
TAX       0.582764 -0.314563  0.720760 -0.035587  0.668023 -0.292048
PTRATIO   0.289946 -0.391679  0.383248 -0.121515  0.188933 -0.355501
LSTAT     0.455621 -0.412995  0.603800 -0.053929  0.590879 -0.613808
MEDV     -0.388305  0.360445 -0.483725  0.175260 -0.427321  0.695360
CAT_MEDV -0.151987  0.365296 -0.366276  0.108631 -0.232502  0.641265

                AGE       DIS       RAD       TAX   PTRATIO     LSTAT  \
CRIM      0.352734 -0.379670  0.625505  0.582764  0.289946  0.455621
ZN       -0.569537  0.664408 -0.311948 -0.314563 -0.391679 -0.412995
INDUS     0.644779 -0.708027  0.595129  0.720760  0.383248  0.603800
CHAS      0.086518 -0.099176 -0.007368 -0.035587 -0.121515 -0.053929
NOX       0.731470 -0.769230  0.611441  0.668023  0.188933  0.590879
RM       -0.240265  0.205246 -0.209847 -0.292048 -0.355501 -0.613808
AGE       1.000000 -0.747881  0.456022  0.506456  0.261515  0.602339
DIS      -0.747881  1.000000 -0.494588 -0.534432 -0.232471 -0.496996
RAD       0.456022 -0.494588  1.000000  0.910228  0.464741  0.488676
TAX       0.506456 -0.534432  0.910228  1.000000  0.460853  0.543993
PTRATIO   0.261515 -0.232471  0.464741  0.460853  1.000000  0.374044
LSTAT     0.602339 -0.496996  0.488676  0.543993  0.374044  1.000000
MEDV     -0.376955  0.249929 -0.381626 -0.468536 -0.507787 -0.737663
CAT_MEDV -0.191196  0.118887 -0.197924 -0.273687 -0.443425 -0.469911

               MEDV  CAT_MEDV
CRIM      -0.388305 -0.151987
ZN         0.360445  0.365296
INDUS     -0.483725 -0.366276
CHAS       0.175260  0.108631
NOX       -0.427321 -0.232502
RM         0.695360  0.641265
AGE       -0.376955 -0.191196
DIS        0.249929  0.118887
RAD       -0.381626 -0.197924
TAX       -0.468536 -0.273687
PTRATIO   -0.507787 -0.443425
LSTAT     -0.737663 -0.469911
MEDV       1.000000  0.789789
CAT_MEDV   0.789789  1.000000
<Axes: >
```








```
# Color the points by the value of CAT.MEDV
Boston.plot.scatter(x='LSTAT', y='NOX',
c=['C0' if c == 1 else 'C1' for c in Boston.CAT_MEDV])
# Plot first the data points for CAT.MEDV of 0 and then of 1
# Setting color to 'none' gives open circles
_, ax = plt.subplots()
for catValue, color in (0, 'C1'), (1, 'C0'):
  subset_df = Boston[Boston.CAT_MEDV == catValue]
  ax.scatter(subset_df.LSTAT, subset_df.NOX, color='none', edgecolor=color)
ax.set_xlabel('LSTAT')
ax.set_ylabel('NOX')
ax.legend(["CAT.MEDV 0", "CAT.MEDV 1"])
plt.show()
## panel plots
# compute mean MEDV per RAD and CHAS
dataForPlot_df = Boston.groupby(['CHAS','RAD']).mean()['MEDV']
# We determine all possible RAD values to use as ticks
ticks = set(Boston.RAD)
for i in range(2):
  for t in ticks.difference(dataForPlot_df[i].index):
    dataForPlot_df.loc[(i, t)] = 0
 # reorder to rows, so that the index is sorted
dataForPlot_df = dataForPlot_df[sorted(dataForPlot_df.index)]
# Determine a common range for the y axis
yRange = [0, max(dataForPlot_df) * 1.1]
fig, axes = plt.subplots(nrows=2, ncols=1)
dataForPlot_df[0].plot.bar(x='RAD', ax=axes[0], ylim=yRange)
dataForPlot_df[1].plot.bar(x='RAD', ax=axes[1], ylim=yRange)
axes[0].annotate('CHAS = 0', xy=(3.5, 45))
axes[1].annotate('CHAS = 1', xy=(3.5, 45))
plt.show()
```




```
import pandas as pd
df=pd.read_json("https://data.cityofnewyork.us/resource/h9gi-nx95.json")
df
```

In a missing value heatmap, rows correspond to records and columns to variables. We use a binary coding of the original dataset where 1 denotes a missing value and 0 otherwise. This new binary table is then colored such that only missing value cells (with value 1) are colored. Figure 3.5 shows an example of a missing value heatmap for a dataset on motor vehicle collisions.4 The missing data heatmap helps visualize the level and amount of “missingness” in the dataset. Some patterns of “missingness” easily emerge: variables that are missing for nearly all observations, as well as clusters of rows that are missing many values. Variables with little missingness are also visible. This information can then be used for determining how to handle the missingness (e.g., dropping some variables, dropping some records, imputing, or via other techniques).

```
import numpy as np
# given a dataframe df create a copy of the array that is 0 if a field contains a
# value and 1 for NaN
naInfo = np.zeros(df.shape)
naInfo[df.isna().values] = 1
naInfo = pd.DataFrame(naInfo, columns=df.columns)
fig, ax = plt.subplots()
fig.set_size_inches(13, 9)
ax = sns.heatmap(naInfo, vmin=0, vmax=1, cmap=["white", "#666666"], cbar=False, ax=ax)
ax.set_yticks([])
# draw frame around figure
rect = plt.Rectangle((0, 0), naInfo.shape[1], naInfo.shape[0], linewidth=1,

edgecolor='lightgrey', facecolor='none')

rect = ax.add_patch(rect)
rect.set_clip_on(False)
plt.xticks(rotation=80)
```

```
# Color the points by the value of CAT.MEDV
Boston.plot.scatter(x='LSTAT', y='NOX',
c=['C0' if c == 1 else 'C1' for c in Boston.CAT_MEDV])
# Plot first the data points for CAT.MEDV of 0 and then of 1
# Setting color to 'none' gives open circles
_, ax = plt.subplots()
for catValue, color in (0, 'C1'), (1, 'C0'):
  subset_df = Boston[Boston.CAT_MEDV == catValue]
  ax.scatter(subset_df.LSTAT, subset_df.NOX, color='none', edgecolor=color)
ax.set_xlabel('LSTAT')
ax.set_ylabel('NOX')
ax.legend(["CAT.MEDV 0", "CAT.MEDV 1"])
plt.show()
```

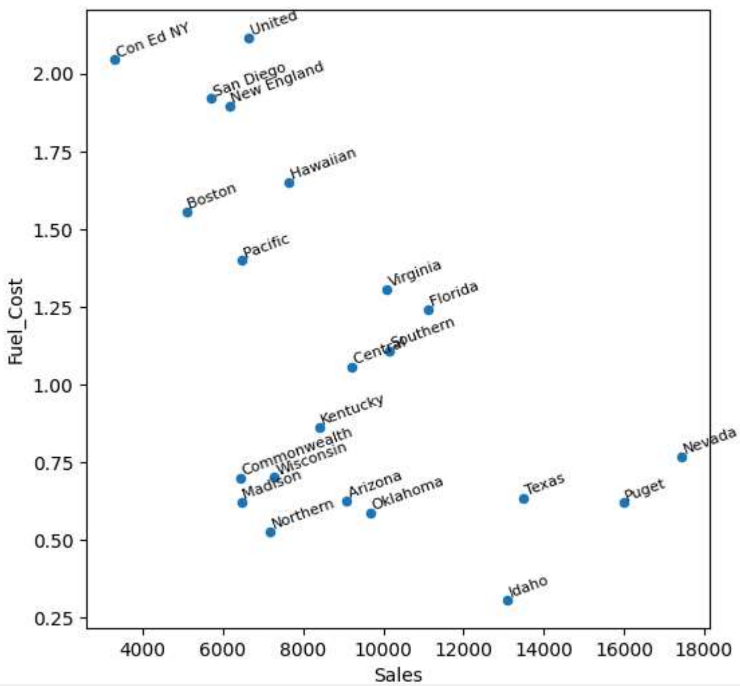
```
## panel plots
# compute mean MEDV per RAD and CHAS
dataForPlot_df = Boston.groupby(['CHAS','RAD']).mean()['MEDV']
# We determine all possible RAD values to use as ticks
ticks = set(Boston.RAD)
for i in range(2):
  for t in ticks.difference(dataForPlot_df[i].index):
    dataForPlot_df.loc[(i, t)] = 0
# reorder to rows, so that the index is sorted
dataForPlot_df = dataForPlot_df[sorted(dataForPlot_df.index)]
# Determine a common range for the y axis
yRange = [0, max(dataForPlot_df) * 1.1]
fig, axes = plt.subplots(nrows=2, ncols=1)
dataForPlot_df[0].plot.bar(x='RAD', ax=axes[0], ylim=yRange)
dataForPlot_df[1].plot.bar(x='RAD', ax=axes[1], ylim=yRange)
axes[0].annotate('CHAS = 0', xy=(3.5, 45))
axes[1].annotate('CHAS = 1', xy=(3.5, 45))
plt.show()
```

```
# Display scatterplots between the different variables
# The diagonal shows the distribution for each variable
df = Boston[['CRIM', 'INDUS', 'LSTAT', 'MEDV']]
axes = pd.plotting.scatter_matrix(df, alpha=0.5, figsize=(6, 6), diagonal='kde')
print(df.corr())
corr = df.corr()#.as_matrix()
#for i, j in zip(*plt.np.triu_indices_from(axes, k=1)):
#  axes[i, j].annotate('%.3f' %corr[i,j], (0.8, 0.8),
#  xycoords='axes fraction', ha='center', va='center')
#plt.show()
```

```
import calendar

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 7))
Amtrak_df = pd.read_csv("https://raw.githubusercontent.com/reisanar/datasets/master/Amtrak.csv")
Amtrak_df['Month'] = pd.to_datetime(Amtrak_df.Month, format='%d/%m/%Y')
Amtrak_df.set_index('Month', inplace=True)
# fit quadratic curve and display
quadraticFit = np.poly1d(np.polyfit(range(len(Amtrak_df)), Amtrak_df.Ridership, 2))
Amtrak_fit = pd.DataFrame({'fit': [quadraticFit(t) for t in range(len(Amtrak_df))]})
Amtrak_fit.index = Amtrak_df.index
ax = Amtrak_df.plot(ylim=[1300, 2300], legend=False, ax=axes[0][0])
Amtrak_fit.plot(ax=ax)
ax.set_xlabel('Year'); ax.set_ylabel('Ridership (in 000s)') # set x and y-axis label
# Zoom in 2-year period 1/1/1991 to 12/1/1992
ridership_2yrs = Amtrak_df.loc['1991-01-01':'1992-12-01']
ax = ridership_2yrs.plot(ylim=[1300, 2300], legend=False, ax=axes[1][0])
ax.set_xlabel('Year'); ax.set_ylabel('Ridership (in 000s)') # set x and y-axis label
# Average by month
byMonth = Amtrak_df.groupby(by=[Amtrak_df.index.month]).mean()
ax = byMonth.plot(ylim=[1300, 2300], legend=False, ax=axes[0][1])
ax.set_xlabel('Month'); ax.set_ylabel('Ridership (in 000s)') # set x and y-axis label
yticks = [-2.0,-1.75,-1.5,-1.25,-1.0,-0.75,-0.5,-0.25,0.0]
ax.set_xticks(range(1, 13))
ax.set_xticklabels([calendar.month_abbr[i] for i in range(1, 13)]);
# Average by year (exclude data from 2004)
byYear = Amtrak_df.loc['1991-01-01':'2003-12-01'].groupby(pd.Grouper(freq='A')).mean()
ax = byYear.plot(ylim=[1300, 2300], legend=False, ax=axes[1][1])
ax.set_xlabel('Year'); ax.set_ylabel('Ridership (in 000s)') # set x and y-axis label
plt.tight_layout()
plt.show()
```

In displays that are not overcrowded, the use of in-plot labels can be useful for better exploration of outliers and clusters. An example is shown in Figure 3.10 (a reproduction of Figure 15.1 with the addition of labels). The figure shows different utilities on a scatter plot that compares fuel cost with total sales. We might be interested in clustering the data, and using clustering algorithms with the labels, helps visualize these clusters and their members (e.g., Nevada and Puget are part of a clear cluster with low fuel costs and high sales).

```
utilities_df = pd.read_csv('https://raw.githubusercontent.com/GauthamBest/Training_Data/master/utilities.csv')
ax = utilities_df.plot.scatter(x='Sales', y='Fuel_Cost', figsize=(6, 6))
points = utilities_df[['Sales','Fuel_Cost','Company']]
_ = points.apply(lambda x:
ax.text(*x, rotation=20, horizontalalignment='left',
verticalalignment='bottom', fontsize=8), axis=1)
```




```
!pip install gmaps
```

Show hidden output

```
import gmaps
SCstudents = pd.read_csv("https://raw.githubusercontent.com/kwartler/Harvard_DataMining_Business_Student/master/Book
gmaps.configure(api_key=os.environ['GMAPS_API_KEY'])
fig = gmaps.figure(center=(39.7, -105), zoom_level=3)
fig.add_layer(gmaps.symbol_layer(SCstudents, scale=2,fill_color='red',
stroke_color='red'))
fig
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-2-d49f04f46e89> in <cell line: 1>()
----> 1 import gmaps
      2 SCstudents =
pd.read_csv("https://raw.githubusercontent.com/kwartler/Harvard_DataMining_Business_Student/master/BookDataSets/SC-US-students-
GPS-data-2016.csv")
      3 gmaps.configure(api_key=os.environ['GMAPS_API_KEY'])
      4 fig = gmaps.figure(center=(39.7, -105), zoom_level=3)
      5 fig.add_layer(gmaps.symbol_layer(SCstudents, scale=2,fill_color='red',

ModuleNotFoundError: No module named 'gmaps'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

```
import networkx as nx
import pandas as pd
import matplotlib.pylab as plt

```