Data Visualization

Data Mining for Business Analytics in Python

Shmueli, Bruce, Gedeck & Patel

 rev 10/17/19

Plots for Data Exploration

Basic Plots
Line Graphs
Bar Charts
Scatterplots

Distribution Plots
Boxplots
Histograms

Possible Goals in Visualization

1. Presentation (reporting & story-telling)
2. Exploration and preliminary analysis

Our focus here is mainly on #2.

The Term “graph”

1. Can mean a figure to represent data (bar graph, etc.)
2. A slightly more technical meaning is a data structure for connections among items or people

To avoid ambiguity, we try to use “plot” for the first meaning.

Python Libraries

`matplotlib` - oldest and most flexible

`seaborn`, `pandas` - wrappers around `matplotlib`; help create plots quickly, but knowledge of `matplotlib` allows better control over final plot

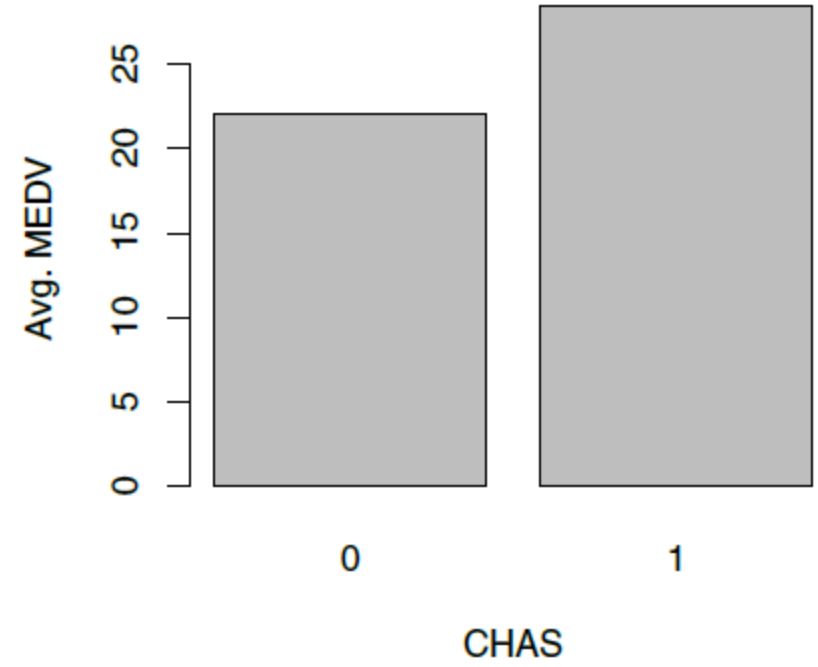
Load Data to Illustrate Plots

CRIM	Crime rate
ZN	Percentage of residential land zoned for lots over 25,000 ft^2
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 $10,000
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 $30,000 (CAT.MEDV = 1) or not (CAT.MEDV = 0)

```
## Boston housing data
housing_df = pd.read_csv('BostonHousing.csv')
housing_df = housing_df.rename(columns={'CAT. MEDV': 'CAT_MEDV'})
```

Bar Chart for Categorical Variable

Average median neighborhood value for neighborhoods that do and do not border the Charles River



Alternative Codes for Bar Chart

CHAS vs. mean MEDV

Using `pandas`:

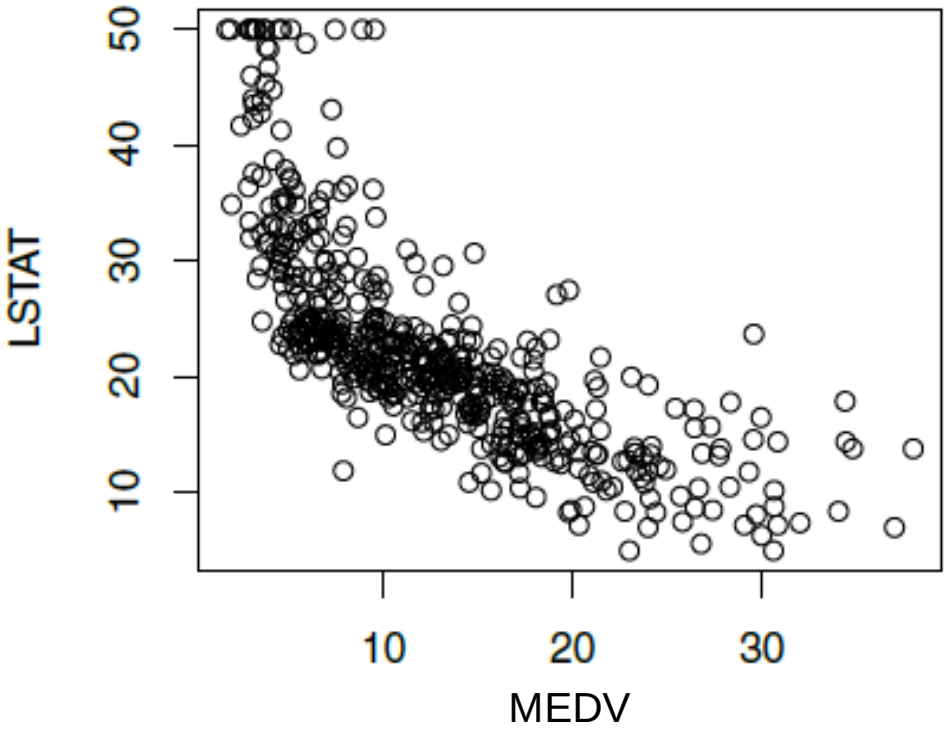
```
# compute mean MEDV per CHAS = (0, 1)
ax = housing_df.groupby('CHAS').mean().MEDV.plot(kind='bar')
ax.set_ylabel('Avg. MEDV')
```

Using `matplotlib`:

```
# compute mean MEDV per CHAS = (0, 1)
dataForPlot = housing_df.groupby('CHAS').mean().MEDV
fig, ax = plt.subplots()
ax.bar(dataForPlot.index, dataForPlot, color=['C5', 'C1'])
ax.set_xticks([0, 1], minor = False)
ax.set_xlabel('CHAS')
ax.set_ylabel('Avg. MEDV')
```

Scatterplot

Displays relationship between two numerical variables



Lower Socioeconomic Status: LSTAT is often used as a proxy for the socioeconomic status of a population in a specific area. It indicates the proportion of residents who may have lower incomes, education levels, or access to resources compared to the rest of the population. Higher LSTAT values suggest a larger percentage of the population with lower socioeconomic status.

Alternative Codes for Scatterplot

Using `pandas`:

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

Using `matplotlib`:

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

Load Data to Illustrate Plots

```
## Load, convert Amtrak data for time series analysis
import time
import pandas as pd
Amtrak_df = pd.read_csv('Amtrak.csv')
Amtrak_df['Date'] = pd.to_datetime(Amtrak_df.Month,
    format='%d/%m/%Y')
ridership_ts = pd.Series(Amtrak_df.Ridership.values,
    index=Amtrak_df.Date)
```

Alternative Codes for Line Plot

Using `pandas`:

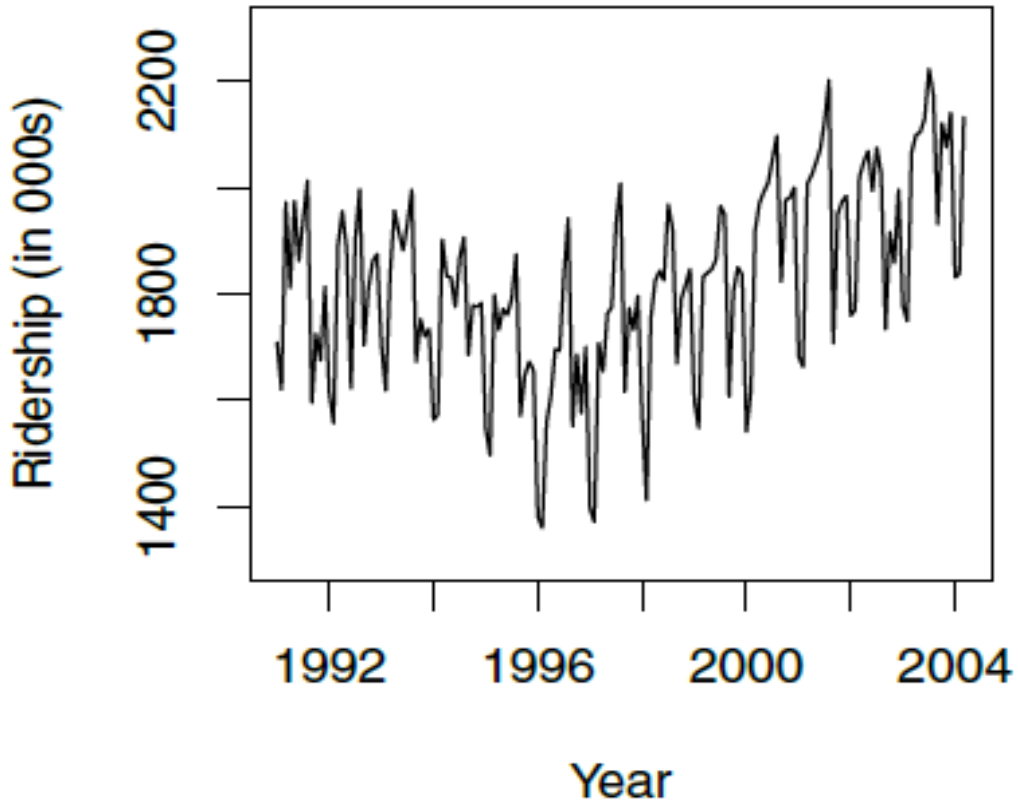
```
ridership_ts.plot(ylim=[1300, 2300], legend=False)
plt.xlabel('Year')  # set x-axis label
plt.ylabel('Ridership (in 000s)')  # set y-axis label
```

Using `matplotlib`:

```
plt.plot(ridership_ts.index, ridership_ts)
plt.xlabel('Year')  # set x-axis label
plt.ylabel('Ridership (in 000s)')  # set y-axis label
```

Line Plot for Time Series

Amtrak Ridership



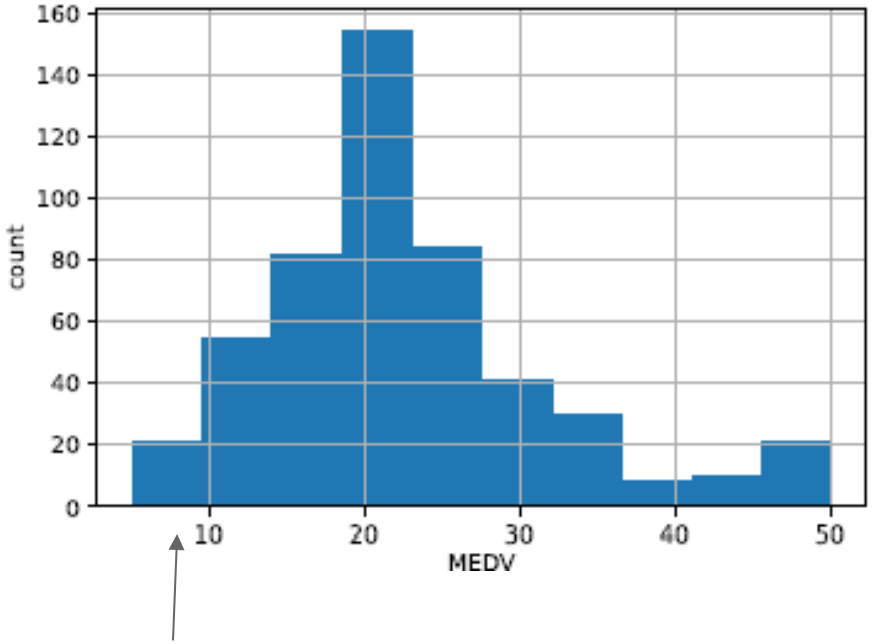
Distribution Plots

- Display “how many” of each value occur in a data set
- Or, for continuous data or data with many possible values, “how many” values are in each of a series of ranges or “bins”

Histograms

Boston Housing example:

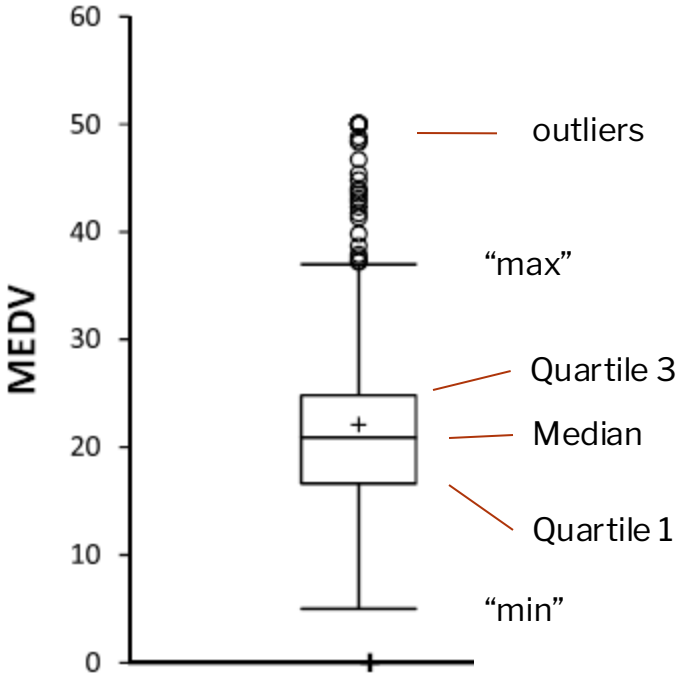
Histogram shows the distribution of the outcome variable (median house value)



About 20 neighborhoods had a median house value in the lowest bin, about $4000K to $9500 (these data are from mid-20th century)

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

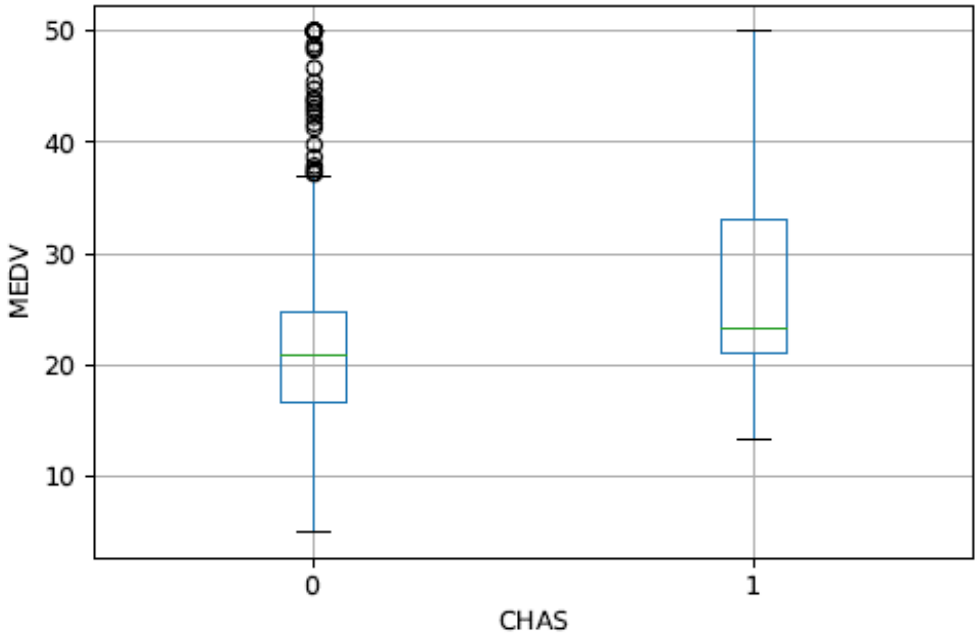
Box Plot



- Top outliers defined as those above Q3+1.5(Q3-Q1).
- “max” = maximum of non-outliers
- Analogous definitions for bottom outliers and for “min”
- Details may differ across software

Boxplots

Side-by-side boxplots are useful for comparing subgroups



Houses in neighborhoods on Charles river (1) are more valuable than those not (0)

```
ax = housing_df.boxplot(column='MEDV', by='CHAS')
ax.set_ylabel('MEDV')
plt.suptitle('')  # Suppress the titles
plt.title('')
```

Heat Maps

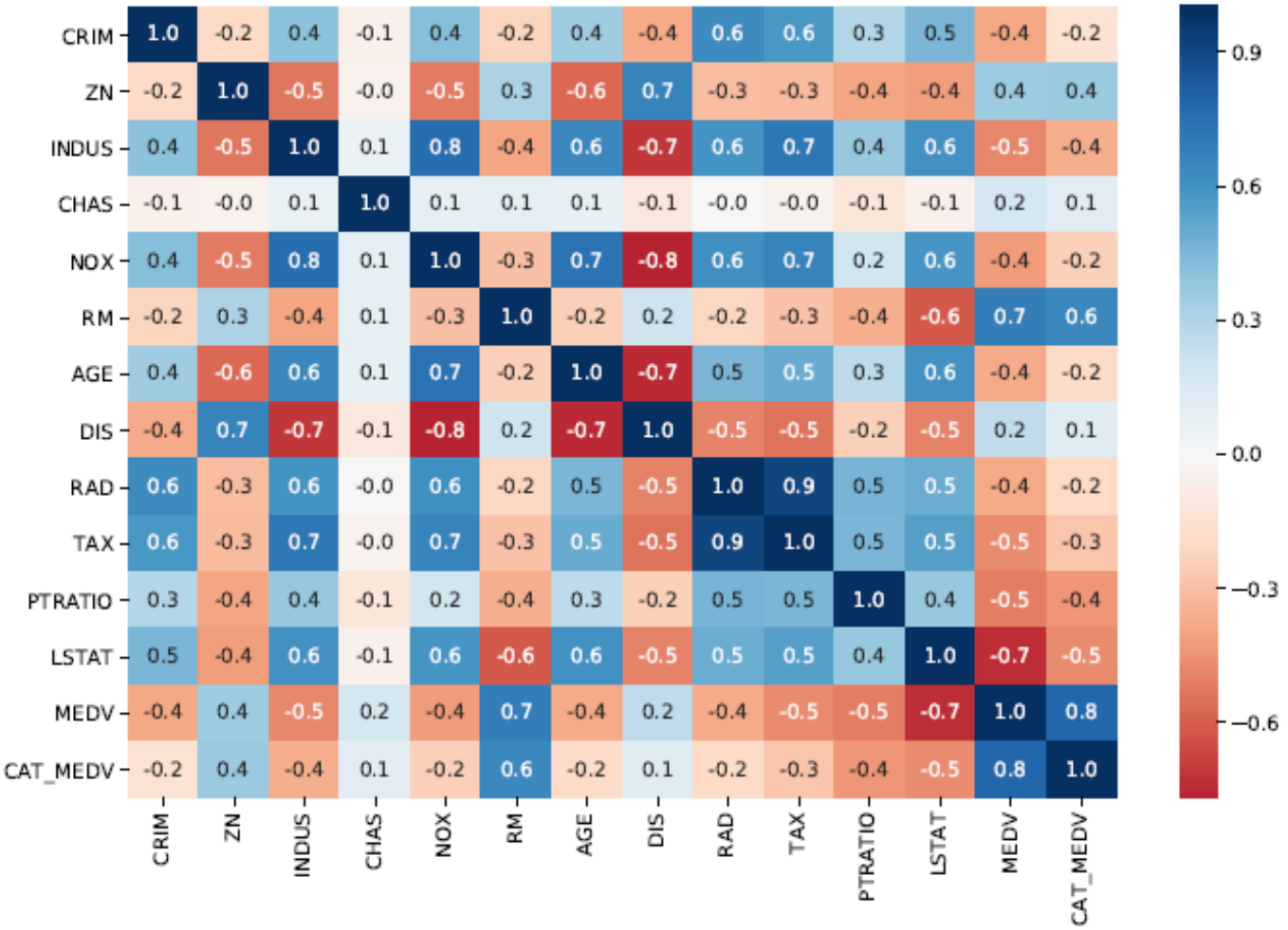
Color conveys information

In data mining, used to visualize

- Correlations
- Missing Data

Heatmap to highlight correlations

Darker and bluer = stronger positive correlation
Darker & redder = stronger negative correlation



Code for Heat Map

```
## simple heatmap of correlations (without values)
import seaborn as sns
corr = housing_df.corr()
sns.heatmap(corr, xticklabels=corr.columns,
  yticklabels=corr.columns)
# Change to divergent scale and fix the range
sns.heatmap(corr, xticklabels=corr.columns,
  yticklabels=corr.columns, vmin=-1, vmax=1,
  cmap="RdBu")
# Include information about values (example
  demonstrates 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)
```

Code for Heat Map

- fmt = format - fmt=".1f" formats the numbers in the cells to one decimal place.
- annot=True displays the values inside each cell of the heatmap.
- when ax is more than one, determining ax=ax[0] tells the function which ax we want to have heatmap on.
- center parameter is used to set the value that will be at the center of the color scale in the heatmap. In this case, center=0 means that the color scale will be centered around the value 0. Values greater than 0 will be displayed in colors on one side of the scale (e.g., a shade of red), while values less than 0 will be displayed in colors on the other side of the scale (e.g., a shade of blue). This helps in visualizing positive and negative correlations in the heatmap. The colors will vary in intensity as they move away from the center value of 0.

Multidimensional Visualization

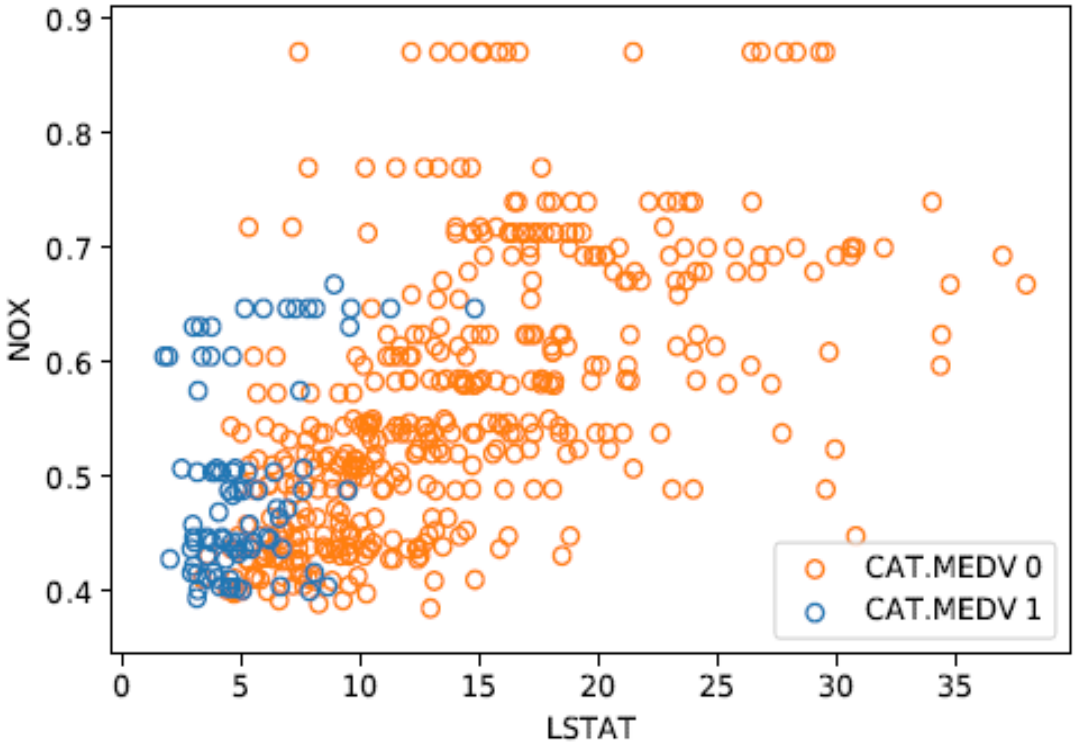
Scatterplot with Color Added

Boston Housing

NOX vs. LSTAT

low median value

high median value

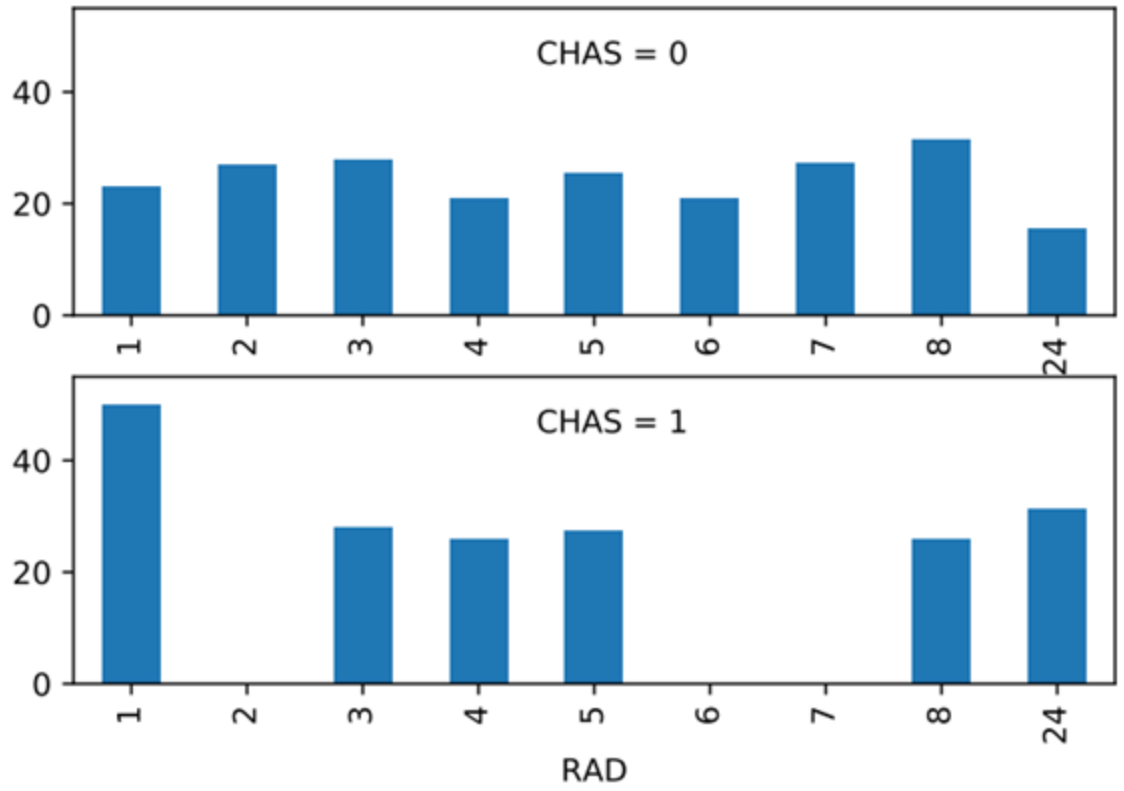


Code for Scatterplot with Color Added

```
# Color the points by the value of CAT.MEDV (pandas library)
housing_df.plot.scatter(x='LSTAT', y='NOX',
    color=['C0' if c == 1 else 'C1' for c in
housing_df.CAT_MEDV])
# Unlike Matplotlib, the Pandas .plot() API doesn't
automatically recognize custom color assignments for the
legend. So, it only shows a single legend entry.
# Plot first the data points for CAT.MEDV of 0 and then of 1
(matplotlib library)
# Setting color to 'none' gives open circles
_, ax = plt.subplots()
for catValue, color in (0, 'C1'), (1, 'C0'):
    subset_df = housing_df[housing_df.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()
```

Bar Plot - 3 Variables

MEDV (y-axis), CHAS and RAD

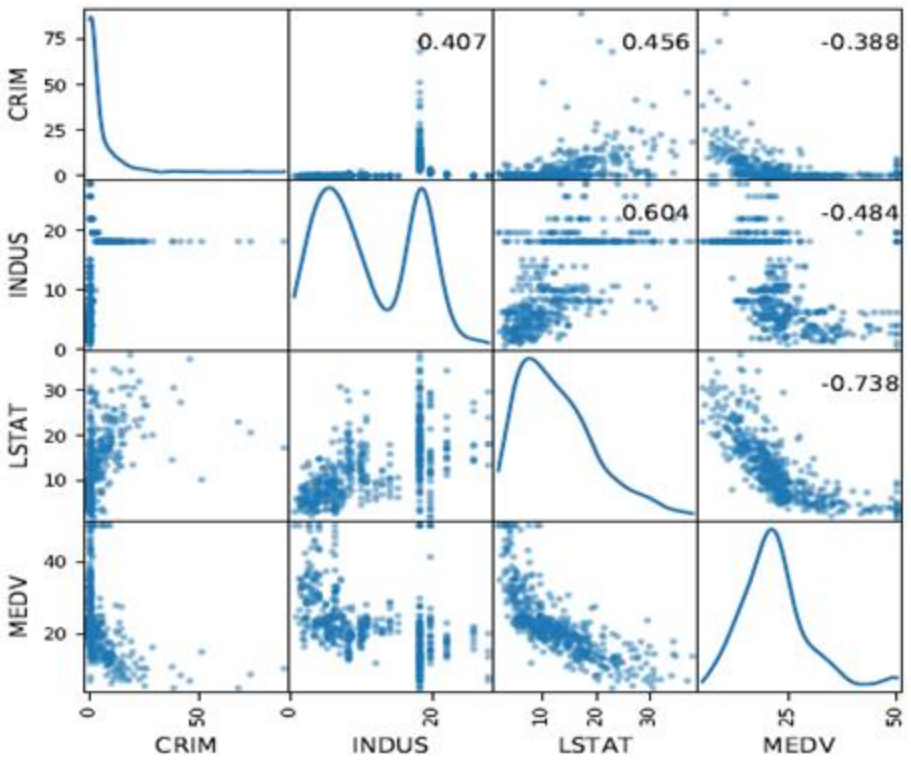


RAD = The ease with which a highway may be reached from a particular place.

* Larger RAD index denotes better accessibility.

Matrix Scatterplot

Diagonal plot is the frequency distribution for the variable




```
from pandas.plotting import scatter_matrix

df = housing_df[['CRIM', 'INDUS', 'LSTAT', 'MEDV']]
axes = scatter_matrix(df, alpha=0.5, figsize=(6, 6), diagonal='kde')
corr = df.corr().values
for i, j in zip(*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()
```

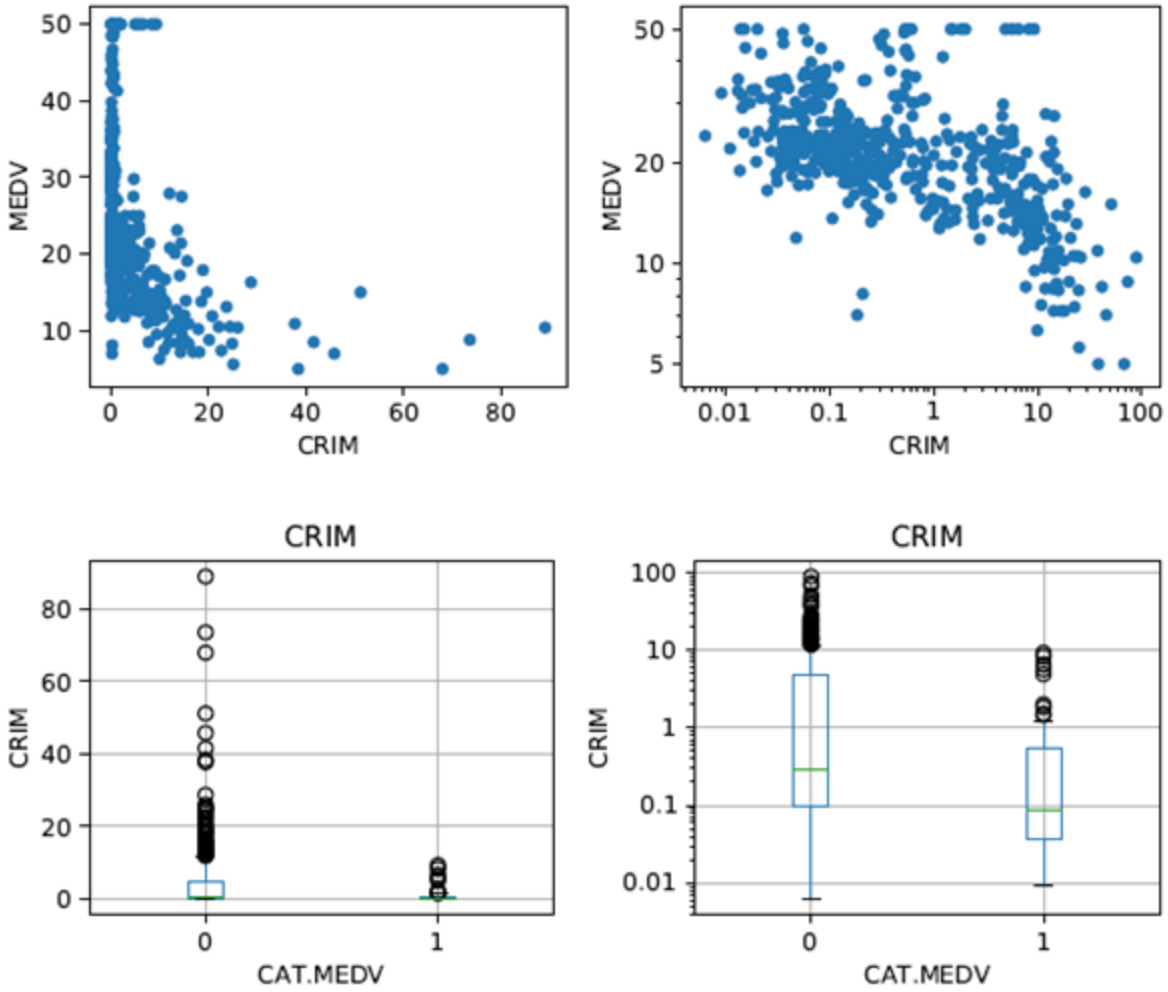
Matrix Scatterplot

- The function “scatter_matrix” comes from the pandas.plotting module, which is part of the Pandas library.
- plt is an alias for Matplotlib, commonly used for plotting in Python.
- “alpha” is a parameter to control the transparency of the points or markers.
- “KDE” stands for Kernel Density Estimation. It is a non-parametric method for estimating the probability density function (PDF) of a continuous random variable. KDE is used to visualize the distribution of data in a way that is smoother and less dependent on specific data points compared to traditional histograms. Alternatives: ‘hist’, ‘both’, ‘None’.
- plt.np.triu_indices_from(axes, k=1):
    1. triu_indices_from is a function in NumPy used to get the indices of the upper triangular part of a matrix or grid.
    2. axes is assumed to be a 2D array or grid, and k=1 specifies that you want to exclude the diagonal elements (the main diagonal) while obtaining indices.
    3. The result of this function call is a tuple containing two arrays: one for the row indices i and one for the column indices j.
- zip(*...) is used to unpack the two arrays obtained from triu_indices_from and create pairs of indices (i, j).
- The purpose of this code is to efficiently iterate over pairs of indices (i, j) corresponding to the upper triangular part of a square matrix or grid. This can be useful in various situations, such as when you want to compute pairwise interactions or correlations between elements in the upper triangular part without redundancy.
- In Matplotlib, xycoords='axes fraction' specifies that the coordinates for the annotation should be relative to the axes, rather than data or figure coordinates. (0, 0) is the bottom-left corner, and (1, 1) is the top-right corner of the axes. Another alternative: 'figure points': Coordinates are specified in points (1/72 inch) relative to the figure.

Manipulation

- Transformations include changing the numeric scale of a variable, binning numerical variables, and condensing categories in categorical variables. The following manipulations support the preprocessing step by revealing patterns and their nature.
- Rescaling
- Aggregation
- Zooming
- Filtering

Rescaling “CRIM” to log scale (on right) “uncrowds” the data



Rescaling "CRIM" to log scale

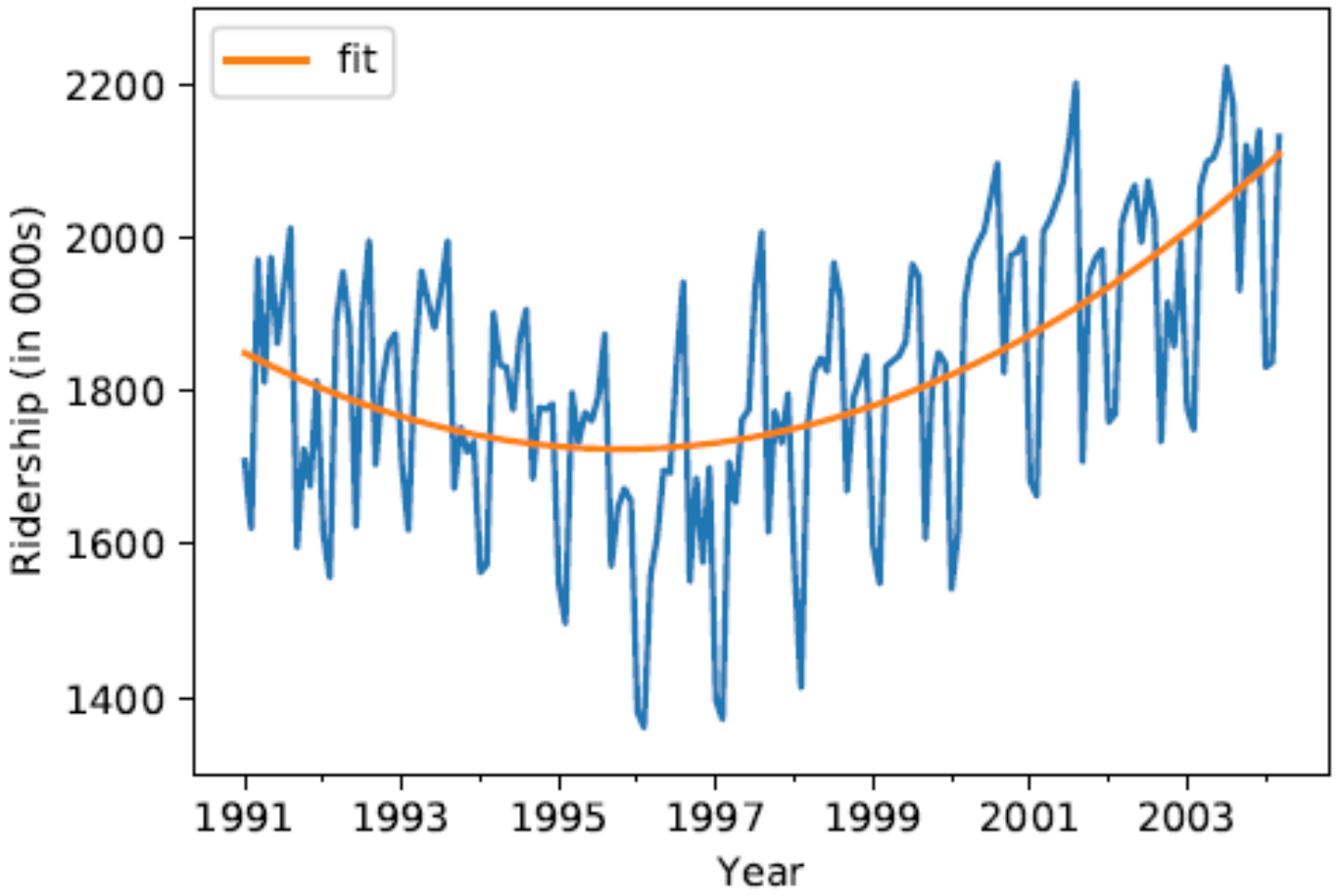
```
# Avoid the use of scientific notation for the log axis
plt.rcParams['axes.formatter.min_exponent'] = 4

## scatter plot: regular and log scale
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(7, 4))
# regular scale
housing_df.plot.scatter(x='CRIM', y='MEDV', ax=axes[0])
# log scale
ax = housing_df.plot.scatter(x='CRIM', y='MEDV', logx=True, logy=True,
     ax=axes[1])
ax.set_yticks([5, 10, 20, 50])

## boxplot: regular and log scale
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(7, 3))
# regular scale
ax = housing_df.boxplot(column='CRIM', by='CAT_MEDV', ax=axes[0])
ax.set_xlabel('CAT.MEDV'); ax.set_ylabel('CRIM')
# log scale
ax = housing_df.boxplot(column='CRIM', by='CAT_MEDV', ax=axes[1])
ax.set_xlabel('CAT.MEDV'); ax.set_ylabel('CRIM'); ax.set_yscale('log')
# suppress the title
axes[0].get_figure().suptitle(''); plt.tight_layout(); plt.show()
```

Rescaling “CRIM” to log scale

- plt.rcParams (Runtime Configuration Parameters) = Adjusting the formatting of axis labels when they involve small or large numbers, such as scientific notation.
- 'axes.formatter.min_exponent' is a parameter in Matplotlib that controls the minimum exponent used for formatting tick labels in scientific notation. By setting it to 4, we are specifying that Matplotlib should use scientific notation when the exponent is less than -4 or greater than or equal to 4. This means that if we have very small or very large numbers on the plot, Matplotlib will represent them in scientific notation when their exponents fall outside the range of -4 to 4.
- plt.tight_layout() function in Matplotlib is used to automatically adjust the spacing between subplots or elements within a figure to ensure that they fit nicely within the available space. It helps prevent elements like axis labels, titles, or other annotations from overlapping or being cut off when we have multiple subplots in a single figure.

Aggregation

Amtrak Ridership – Monthly Data – Curve Added



Amtrak Ridership – Monthly Data – Quadratic fit

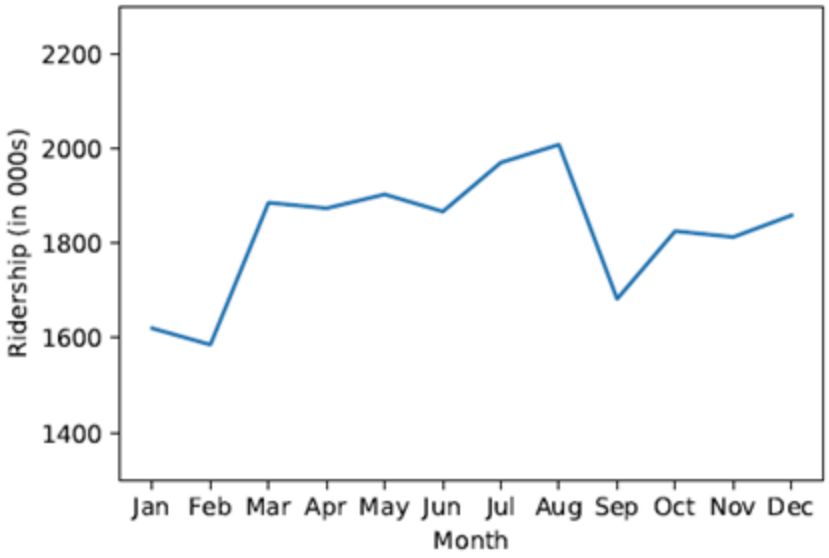
```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 7))

Amtrak_df = pd.read_csv('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) #Overlay a new plot onto the previous plot
ax.set_xlabel('Year'); ax.set_ylabel('Ridership (in 000s)')  # set x
   and y-axis label
```

Amtrak Ridership – Monthly Data – Quadratic fit

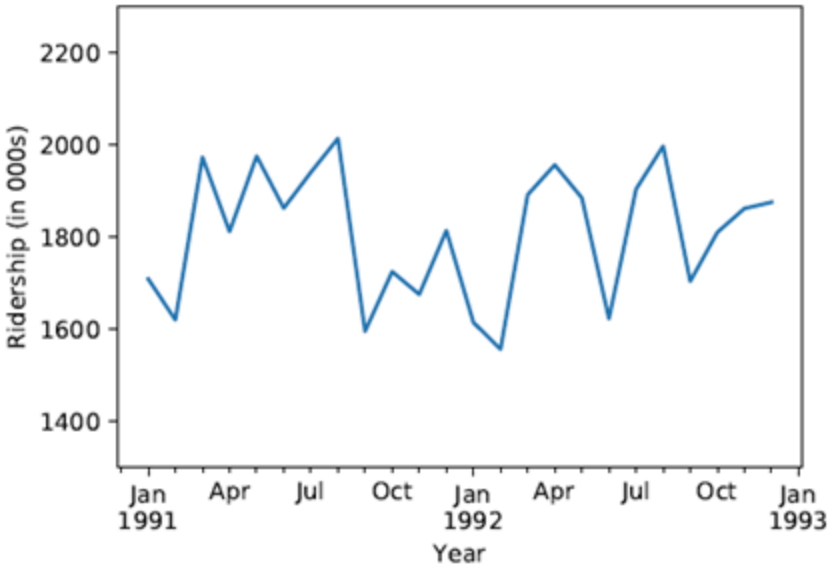
- Amtrak_df.set_index('Month', inplace=True) is used to set the "Month" column in the DataFrame Amtrak_df as the index of the DataFrame. When you set the index in this way with inplace=True, it modifies the DataFrame in place, meaning it directly changes the DataFrame without the need to assign the result to a new variable.
- The np.polyfit function will calculate the coefficients of the quadratic polynomial that best fits the data points. The result will be an array of coefficients, where the first element represents the coefficient of the quadratic term, the second element represents the coefficient of the linear term, and the third element represents the coefficient of the constant term.
- range(len(Amtrak_df)) generates a sequence of integers from 0 to one less than the length of the DataFrame Amtrak_df. These integers represent the x-values (independent variable) for the curve fitting.
- Amtrak_df.Ridership is the y-values (dependent variable) that you want to fit the curve to. It represents the Ridership data from the DataFrame.
- “2” specifies that you want to fit a quadratic polynomial (degree 2).
- np.poly1d(...) takes the coefficients obtained from the polyfit function and creates a polynomial function using NumPy's poly1d constructor. This allows you to work with the fitted polynomial function as if it were a regular Python function. The polynomial function is in the form: $f(x)=ax^2+bx+c$.
- Higher-degree polynomials can capture more complex patterns but may not generalize well to new data points (overfitting).
- [quadraticFit(t) for t in range(len(Amtrak_df))] This list comprehension iterates over a range of values from 0 to len(Amtrak_df) - 1, where each value represents an x-value or index. For each index t, it calculates the corresponding fitted y-value by evaluating the quadraticFit function at that index. This effectively computes the fitted values for the entire range of data points.
- pd.DataFrame({'fit': ...}) creates a new pandas DataFrame with one column labeled 'fit' and populates it with the fitted values obtained in previous step.
- If you wonder why degree 2 not higher, you can use the following directions => “R-squared (Coefficient of Determination)”: This statistic measures how well the model explains the variability of the data. Higher values indicate a better fit. “Residual Analysis”: Analyze the residuals (differences between observed and fitted values). For a good fit, residuals should be randomly scattered around zero. “Comparison with Higher-Degree Polynomials”: Fit polynomials of higher degrees (e.g., cubic, quartic) and compare the fit quality using metrics such as R-squared, adjusted R-squared, or Akaike Information Criterion (AIC).

Amtrak Ridership – Different Aggregations

Monthly Averages



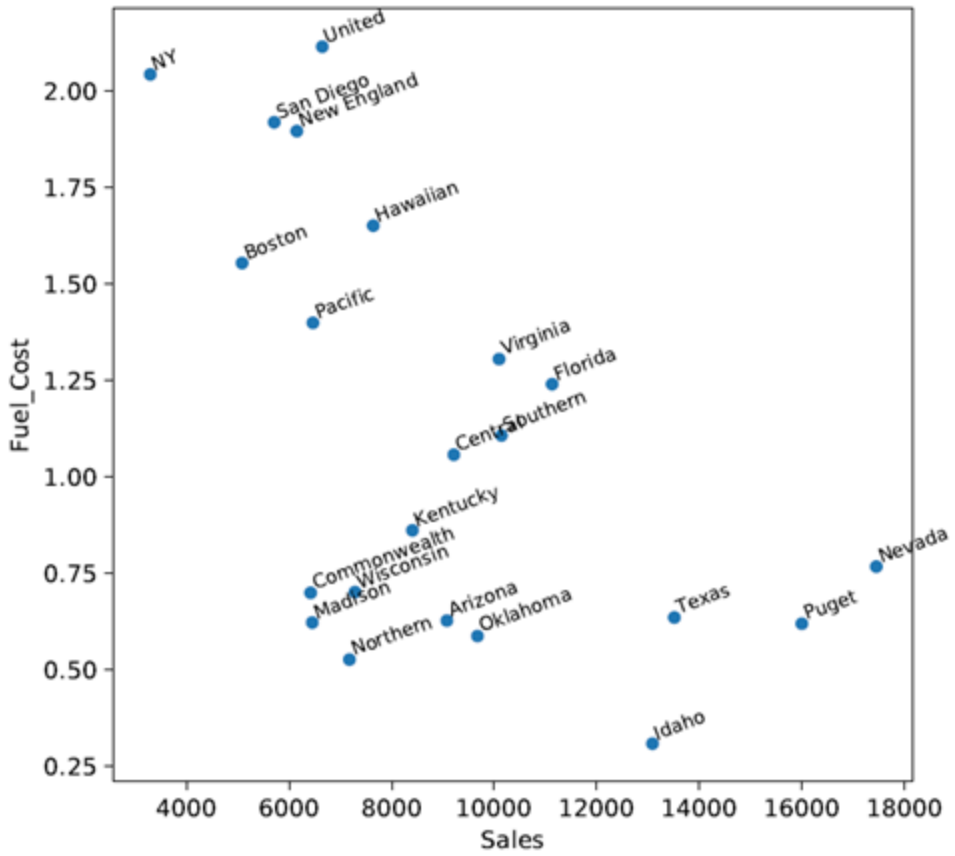
Zoom In



Amtrak Ridership - More Options

```
# Average by month
byMonth = Amtrak_df.groupby(by=[Amtrak_df.index.month]).mean()
# Amtrak_df.index.month is a way to extract and group by the month component of the DatetimeIndex
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
ax.set_xticks(range(1, 13))
ax.set_xticklabels([calendar.month_abbr[i] for i in range(1, 13)]);
# This creates a list of abbreviated month names (e.g., 'Jan', 'Feb', 'Mar', etc.) using Python's built-in calendar module.

# 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
```

Scatter Plot with Labels
(for displays that are not overcrowded)




```
utilities_df = pd.read_csv('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)
```

Scatter Plot with Labels
(for displays that are not overcrowded)

- When you use single brackets like utilities_df['Sales'], you are selecting a single column from the DataFrame, and the result will be a pandas Series. Pandas allows single brackets for selecting only one column at a time.
- When you use double brackets like utilities_df[['Sales', 'Fuel_Cost', 'Company']], you are selecting multiple columns from the DataFrame. The result is a pandas DataFrame. The outer brackets are for indexing, and the inner list specifies the columns you want.
- The .apply() method in pandas is a powerful function that allows you to apply a custom function or operation to elements of a DataFrame or Series. It can be used in various scenarios to transform, manipulate, or analyze data. It is used to apply a function to each row (or axis=1) of the DataFrame. In this case, you are applying a lambda function to each row.
- ax.text() is a method of a Matplotlib Axes (ax) to add a text annotation to the plot. The *x syntax is used to unpack the values in the row x, assuming that the row contains the necessary information for the text annotation (e.g., x and y coordinates).
- _ = ... The result of the .apply() method is assigned to _, which is a common convention to indicate that the result is not used or stored.
- rotation=20 parameter specifies the rotation angle (in degrees) for the text. In this case, the text will be rotated 20 degrees clockwise from its default horizontal orientation.
- horizontalalignment='left' parameter controls the horizontal alignment of the text relative to the specified coordinates (x, y). Setting it to 'left' means that the text will be aligned to the left of the coordinates.
- verticalalignment='bottom' parameter controls the vertical alignment of the text relative to the specified coordinates (x, y). Setting it to 'bottom' means that the text will be aligned to the bottom of the coordinates.
- lambda x: ... is a lambda function that takes a single argument x, which represents a row of the DataFrame.
- In Python, a lambda function is a small function defined using the lambda keyword. Lambda functions are also known as anonymous functions because they do not have a formal name like regular functions defined with the def keyword. Instead, lambda functions are typically used for simple, one-line operations where a full function definition is not necessary.
- The basic syntax of a lambda function is as follows: lambda arguments(list of one or more parameters that the lambda function takes. These parameters are similar to the parameters of a regular function): expression (a single expression or operation that the lambda function performs. The result of this expression is implicitly returned by the lambda function when it is called)
- axis=1 states that the lambda function is applied row-wise which processes each row in the DataFrame, not column-wise(axis=0).

Other Visualization Alternatives

- Scaling (large datasets)

1. Sampling—drawing a random sample and using it for plotting
2. Reducing marker size
3. Using more transparent marker colors and removing fill
4. Breaking down the data into subsets (e.g., by creating multiple panels)
5. Using aggregation (e.g., bubble plots where size corresponds to number of observations in a certain range)
6. Using jittering (slightly moving each marker by adding a small amount of noise)

- Multivariate plot

1. Parallel coordinate plot (presenting multidimensional information in a two-dimensional plot-multivariate profile)

- Interactive visualization

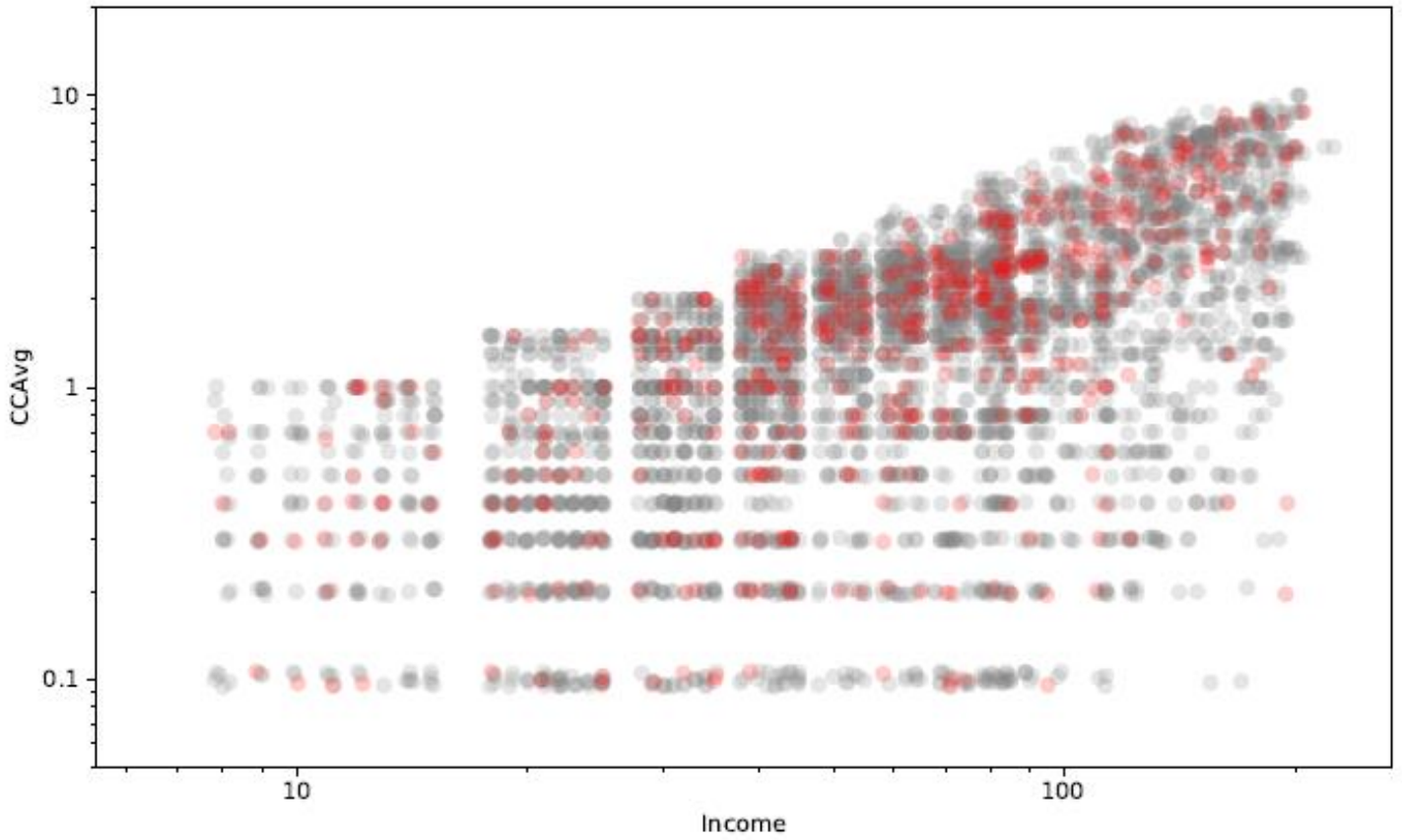
An interface that supports the following principles:

1. Making changes to a chart is easy, rapid, and reversible.
2. Multiple concurrent charts and tables can be easily combined and displayed on a single screen.
3. A set of visualizations can be linked, so that operations in one display are reflected in the other displays.

- Specialized visualizations

1. Networked data (graph)
2. Hierarchical data (treemaps)
3. Geographical data (map charts)

Scaling: Smaller markers, jittering, color contrast (Universal Bank; **red** = accept loan)



Jitter = add noise to “unstack” markers that hide markers underneath

Scaling: Smaller markers, jittering, color contrast (Universal Bank; red = accept loan)

```
def jitter(x, factor=1):
    """ Add random jitter to x values """
```

“” (triple quotes) is called docstring. A docstring is a string literal used to document a module, function, class, or method in Python. It is typically placed immediately after the definition of a function or class and is used to describe the purpose, functionality, and usage of the code. Docstrings are an important part of Python documentation because they can be accessed by other developers and tools like help() or __doc__ attribute.

```
    sx = np.array(sorted(x))
    delta = sx[1:] - sx[:-1]
```

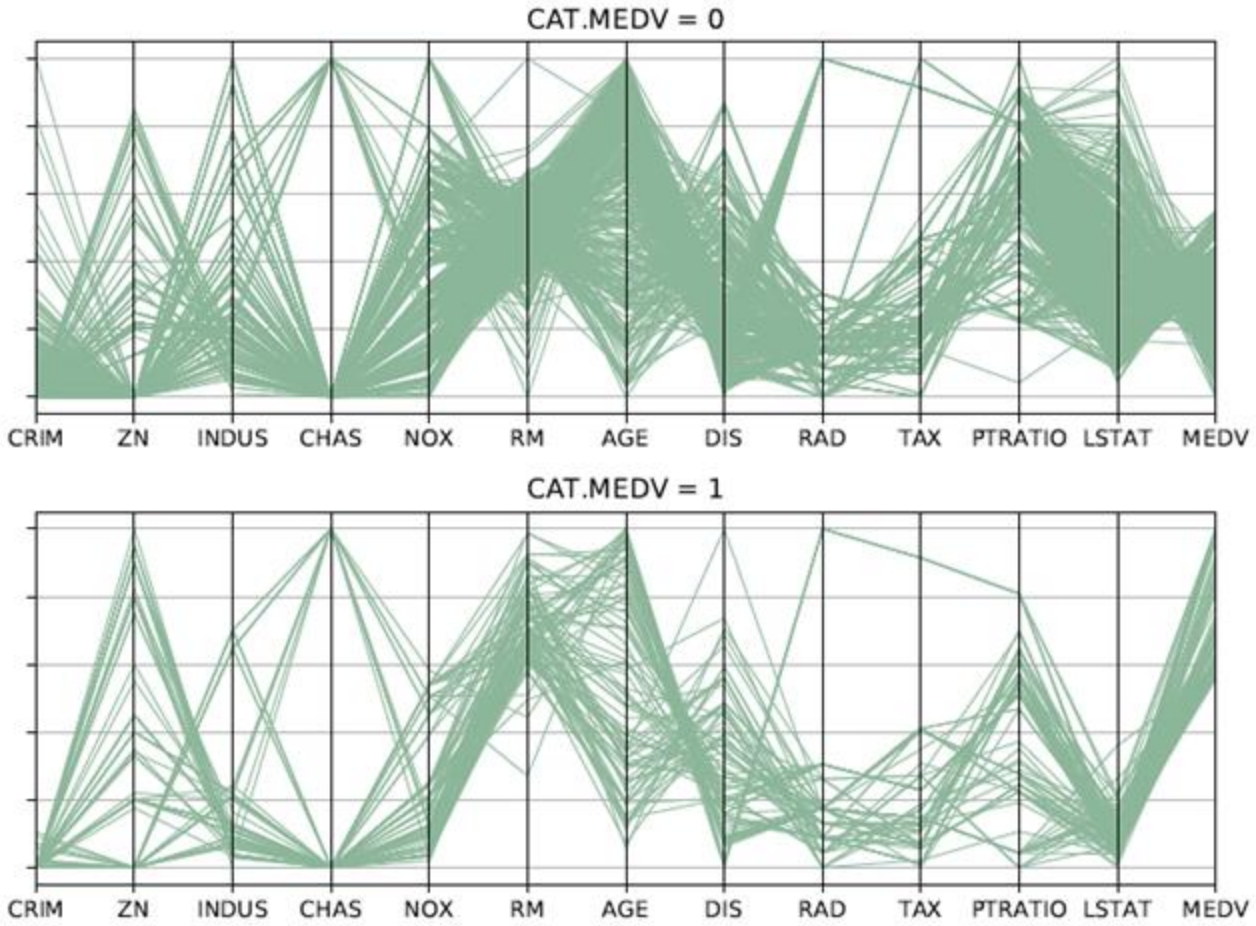
sx[1:] refers to all elements of sx starting from the second element (index 1) to the last element. sx[:-1] refers to all elements of sx except the last element (i.e., from the first element to the second-to-last element).

```
    minDelta = min(d for d in delta if d > 0)
    a = factor * minDelta / 5
    return x + np.random.uniform(-a, a, len(x))

universal_df = pd.read_csv('UniversalBank.csv')

saIdx = universal_df[universal_df['Securities Account'] == 1].index

plt.figure(figsize=(10,6))
plt.scatter(jitter(universal_df.drop(saIdx).Income),
            jitter(universal_df.drop(saIdx).CCAvg),
            marker='o', color='grey', alpha=0.2)
plt.scatter(jitter(universal_df.loc[saIdx].Income),
            jitter(universal_df.loc[saIdx].CCAvg),
            marker='o', color='red', alpha=0.2)
plt.xlabel('Income')
plt.ylabel('CCAvg')
plt.ylim((0.05, 20))
axes = plt.gca()
```

‘gca’ stands for "Get Current Axes." It is a function provided by Matplotlib that is used to retrieve the current Axes object associated with the current figure. The Axes object represents the region of the figure where you can plot your data. It is for referencing in more advanced/complex plotting.

```
axes.set_xscale("log")
axes.set_yscale("log")
plt.show()
```

Parallel Coordinate Plot (Boston Housing)



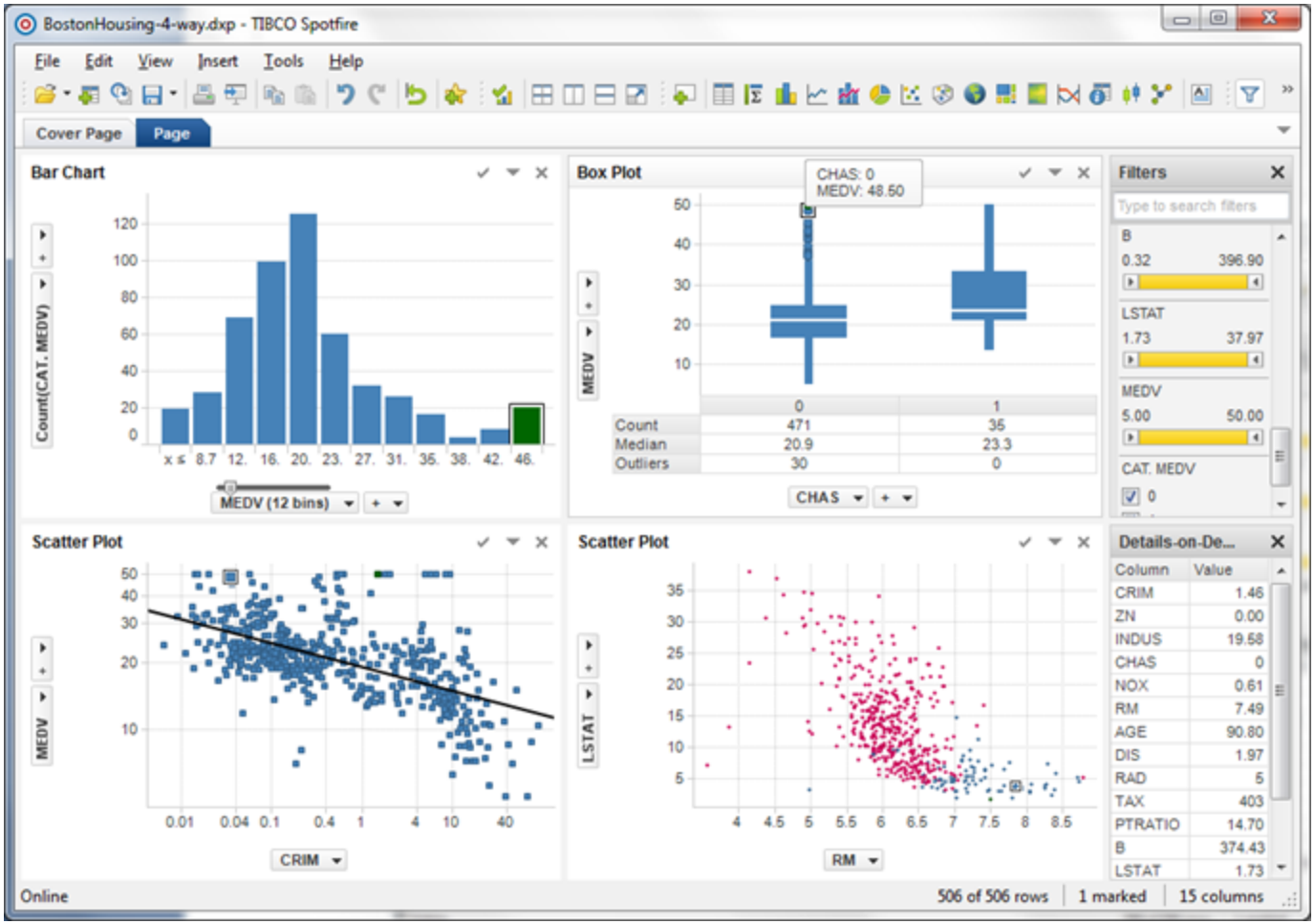
- All variables are rescaled to 0-1 scale.
- Each green line is a single record.
- CAT.MEDV = Is median value of owner-occupied homes in tract above $30,000?

Parallel Coordinate Plot (Boston Housing)

```
# Transform the axes, so that they all have the same range
import sklearn.preprocessing import MinMaxScaler
from pandas.plotting import parallel_coordinates

min_max_scaler = preprocessing.MinMaxScaler()
dataToPlot = pd.DataFrame(min_max_scaler.fit_transform(housing_df),
                          columns=housing_df.columns)

fig, axes = plt.subplots(nrows=2, ncols=1)
for i in (0, 1):
    parallel_coordinates(dataToPlot.loc[dataToPlot.CAT_MEDV == i],
                         'CAT_MEDV', ax=axes[i], linewidth=0.5)
    axes[i].set_title('CAT.MEDV = {}'.format(i))
    # To insert the value of i into the string, you need to use f-strings or .format() to format the string properly.
    axes[i].set_yticklabels([])
    axes[i].legend().set_visible(False)

plt.tight_layout()  # Increase the separation between the plots
```

Linked plots

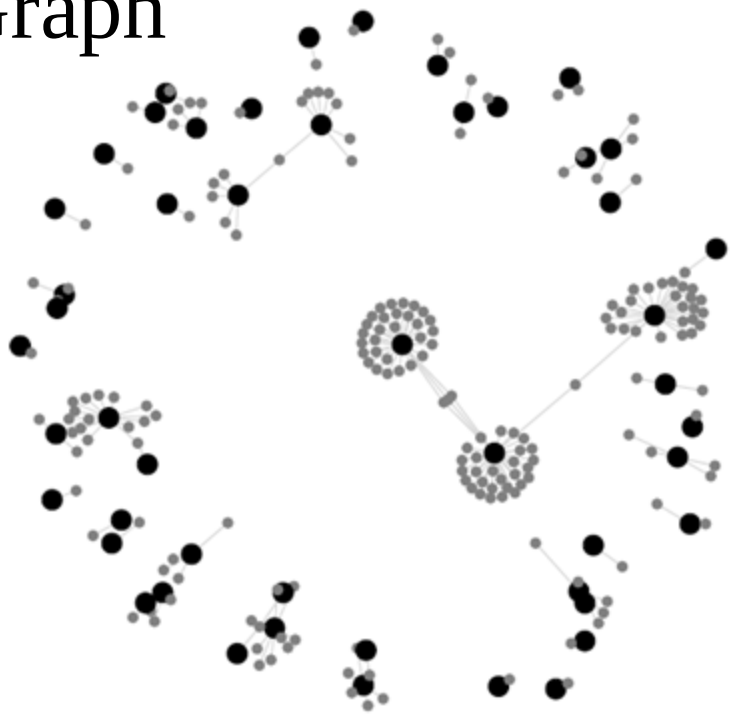
(same record is highlighted in each plot)



Produced in Spotfire

Network Graph

- Consists of actors, called “Nodes”, and relationship between them. They are represented by circles.
- ”Edges” are the relations between nodes and are represented by lines connecting nodes.



eBay Auctions for
Swarovski Beads
Black circles = sellers
Gray circles = buyers

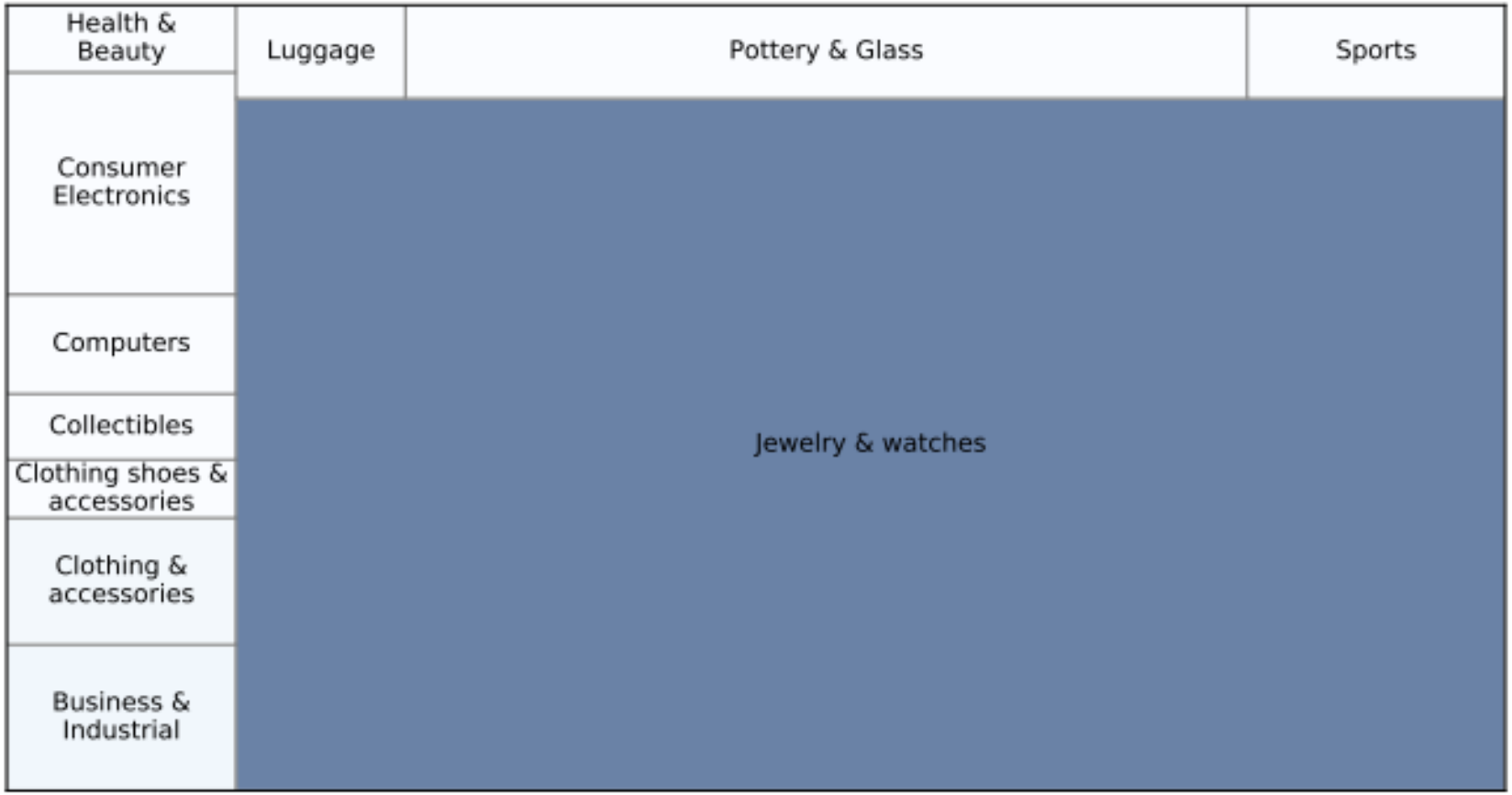
```
import networkx as nx
ebay_df = pd.read_csv('eBayNetwork.csv')

G = nx.from_pandas_edgelist(ebay_df, source='Seller', target='Bidder’) # network graph

isBidder = [n in set(ebay_df.Bidder) for n in G.nodes()] # set() removes duplicates.
isBidder is to create a boolean indicator for each node in the network graph G that
identifies whether the node represents a bidder or not
pos = nx.spring_layout(G, k=0.13, iterations=60, scale=0.5)
plt.figure(figsize=(10,10))
nx.draw_networkx(G, pos=pos, with_labels=False,
                 edge_color='lightgray',
                 node_color=['gray' if bidder else 'black' for bidder in isBidder],
                 node_size=[50 if bidder else 200 for bidder in isBidder])
plt.axis('off')
plt.show()
```

Network Graph

- nx.from_pandas_edgelist: This is a function provided by the NetworkX library that creates a graph from an edge list, where each row in the DataFrame represents an edge in the graph.
- The code snippet [n in set(ebay_df.Bidder) for n in G.nodes()] checks whether each node in the NetworkX graph G corresponds to a bidder in the 'Bidder' column of the DataFrame ebay_df.
- set(ebay_df.Bidder): This converts the 'Bidder' column into a set, which eliminates any duplicate names and retains unique bidder names for faster performance.
- The code nx.spring_layout(G, k=0.13, iterations=60, scale=0.5) is using the NetworkX library to compute the layout positions of nodes. The k parameter represents the spring constant, which determines the strength of the attractive forces between nodes. A smaller value of k results in a more spread-out layout, while a larger value makes nodes cluster together more tightly. iterations parameter specifies the number of iterations the spring layout algorithm should run to determine the optimal node positions. More iterations can lead to a more refined layout. scale parameter is used to scale the entire layout. It does not affect the relative positions of nodes but can be used to adjust the overall size of the layout. When you set scale to a value greater than 1, it increases the size of the layout. Nodes will be positioned farther apart, resulting in a more spread-out graph layout. When you set scale to a value less than 1 (e.g., between 0 and 1), it decreases the size of the layout. Nodes will be positioned closer together, leading to a more compact graph layout.
- The spring layout algorithm is a force-directed layout method where nodes are treated as particles connected by springs. Nodes repel each other due to electrostatic forces, and edges (springs) between connected nodes exert attractive forces. Over multiple iterations, the layout algorithm adjusts the positions of nodes based on these forces until an equilibrium is reached.
- The resulting pos variable will contain a dictionary where each node in the graph is mapped to its (x, y) coordinates in the layout. These coordinates can be used for visualization, such as creating a graph plot with nodes and edges in a visually appealing arrangement.

Treemap – eBay Auctions

- Hierarchically ordered by item category, sub-category, and brand.
- Ideally, treemaps should be explored interactively, zooming to different levels of the hierarchy.
- Categorical variables can be included in the display by using hue. Numerical variables can be included via rectangle size and color intensity (ordering of the rectangles is sometimes used to reinforce size).



Treemap – eBay Auctions

(Hierarchical eBay data: Category> sub-category> Brand)

```
import squarify
ebayTreemap = pd.read_csv('EbayTreemap.csv')

grouped = []
for category, df in ebayTreemap.groupby(['Category']):
    negativeFeedback = sum(df['Seller Feedback'] < 0) / len(df)
    grouped.append({
        'category': category,
        'negativeFeedback': negativeFeedback,
        'averageBid': df['High Bid'].mean()
    })
byCategory = pd.DataFrame(grouped)

norm = matplotlib.colors.Normalize(vmin=byCategory.negativeFeedback.min(),
                                   vmax=byCategory.negativeFeedback.max())
colors = [matplotlib.cm.Blues(norm(value)) for value in byCategory.negativeFeedback]

fig, ax = plt.subplots()
fig.set_size_inches(9, 5)

squarify.plot(label=byCategory.category, sizes=byCategory.averageBid, color=colors,
              ax=ax, alpha=0.6, edgecolor='grey')

ax.get_xaxis().set_ticks([])
ax.get_yaxis().set_ticks([])

plt.subplots_adjust(left=0.1)
plt.show()
```

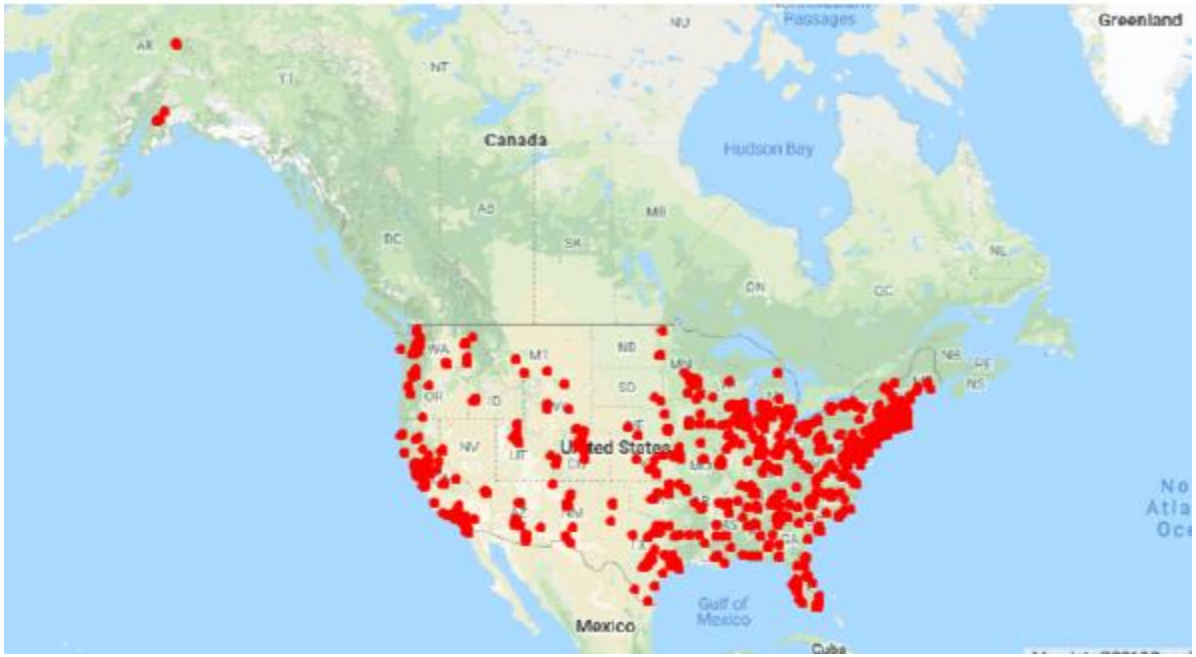
Treemap – eBay Auctions

(Hierarchical eBay data: Category> sub-category> Brand)

- The squarify library is a Python library used for creating treemaps. It allows you to generate treemaps with ease. It provides functions to layout and render hierarchical data as a set of nested rectangles, making it useful for various data visualization tasks, especially when you want to display hierarchical or part-to-whole relationships.
- ‘grouped’ is an empty list to store the calculated statistics for each category. ‘groupby’ creates subsets of the DataFrame, one for each unique category.
- The loop iterates through each category, and for each category, df contains the subset of data for that category.
- It iterates through each group created by groupby, where category represents the category name, and df represents the subset of the DataFrame for that category.
- matplotlib.colors.Normalize is a function from the matplotlib.colors module that is used to normalize data values to the range [0, 1]. This is often used in data visualization to ensure that data values are appropriately mapped to colors in a plot or heatmap. This normalization is useful when you want to visualize data with a color scale, such as in heatmaps, scatter plots, or other types of data visualizations, to represent the variation in the 'negativeFeedback' values across different categories.
- matplotlib.cm.Blues is a colormap from the matplotlib.cm module, specifically the 'Blues' colormap. Colormaps define a range of colors that can be used to represent data values. In this case, 'Blues' is chosen as the colormap.
- squarify.plot is used to create a treemap plot. It takes several parameters, including label for labels, sizes for the sizes of rectangles (in this case, based on 'averageBid'), color for colors (from the 'colors' list), ax for the axes to plot on, alpha for transparency, and edgecolor for the edge color of rectangles.
- ax.get_xaxis().set_ticks([]) is used to hide or remove the x-axis ticks (or tick marks) from a matplotlib plot. This is often done when you want to customize the appearance of a plot and remove the default x-axis tick marks for a cleaner or more customized visualization.
- left=0.1 is an argument passed to subplots_adjust, which specifies the adjustment of the left margin. In this case, it shifts the left margin to the right by 0.1 times the width of the figure.

Using Google Maps

Location of Statistics.com U.S. students and instructors




```
import gmaps
import collections
import pandas as pd

collections.Iterable = collections.abc.Iterable
SCstudents = pd.read_csv('SC-US-students-GPS-data-2016.csv')
GMAPS_API_KEY = 'YOUR_API_KEY'
gmaps.configure(api_key=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
```

1) Go to https://console.cloud.google.com/
2) Watch the "GMAPS Activation" video on Canvas

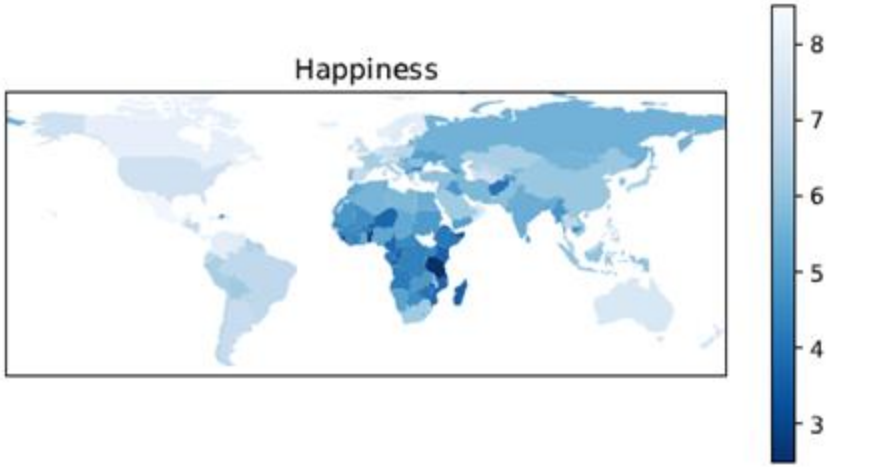
Using Google Maps

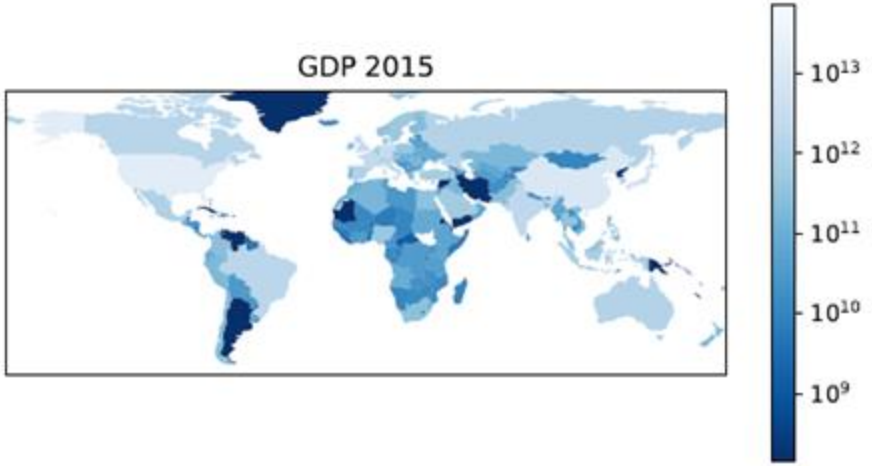
Location of Statistics.com U.S. students and instructors

- gmaps.configure(api_key=GMAPS_API_KEY) is a Python configuration step used when working with the gmaps library to set your Google Maps API key.
- 'GMAPS_API_KEY' is a string representing the name of the environment variable that should contain your Google Maps API key.
- By using gmaps.configure with your API key, you are setting up the gmaps library to use your API key for authentication when making requests to the Google Maps API. This is necessary to access various Google Maps services such as geocoding, mapping, and routing.
- A zoom_level of 0 or 1 would display a very wide view of the world.
- A zoom_level of 10 or higher would zoom in to show more detailed local maps.
- The scale parameter in the gmaps.symbol_layer function determines the size of the symbols (markers) that are drawn on the map.

Map Chart

(Comparing countries' well-being with GDP)





Map Chart Code (1)

```
import matplotlib
import matplotlib.pyplot as plt
import cartopy
import cartopy.io.shapereader as shpreader
import cartopy.crs as ccrs

gdp_df = pd.read_csv('gdp.csv', skiprows=4)
gdp_df.rename(columns={'2015': 'GDP2015'}, inplace=True)
gdp_df.set_index('Country Code', inplace=True)  # use three letter country code to access
   rows

# The file contains a column with two letter combinations, use na_filter to avoid
converting the combination NA into not-a-number
happiness_df = pd.read_csv('Veerhoven.csv', na_filter = False)
happiness_df.set_index('Code', inplace=True)  # use the country name to access rows

fig = plt.figure(figsize=(7, 8))
ax1 = plt.subplot(2, 1, 1, projection=ccrs.PlateCarree())
ax1.set_extent([-150, 60, -25, 60])
ax2 = plt.subplot(2, 1, 2, projection=ccrs.PlateCarree())
ax2.set_extent([-150, 60, -25, 60])

# Create a color mapper
cmap = plt.cm.Blues_r
norm1 = matplotlib.colors.Normalize(vmin=happiness_df.Score.dropna().min(),
                                    vmax=happiness_df.Score.dropna().max())
norm2 = matplotlib.colors.LogNorm(vmin=gdp_df.GDP2015.dropna().min(),
                                  vmax=gdp_df.GDP2015.dropna().max())
```

Map Chart Code (1)

- Cartopy is a Python library for geospatial data visualization and analysis. It is built on top of Matplotlib and provides tools for working with geospatial data, such as maps, projections, and geographic transformations. Cartopy is particularly useful for creating maps and visualizing data that is geographically referenced. ccrs stands for "Cartopy Coordinate Reference System."
- ax1 = plt.subplot(2, 1, 1, projection=ccrs.PlateCarree()): This line creates a subplot with a 2x1 grid (two rows and one column) and selects the first subplot (top subplot). The projection=ccrs.PlateCarree() argument specifies that this subplot should use the Plate Carrée map projection, which is a simple equidistant cylindrical projection often used for global maps. ax1 is a reference to this subplot.
- ax1.set_extent([-150, 60, -25, 60]): This line sets the spatial extent of the subplot. The set_extent() method is used to specify the longitude and latitude limits for the map. In this case, the extent is set to:

  Minimum longitude: -150 degrees

  Maximum longitude: 60 degrees

  Minimum latitude: -25 degrees

  Maximum latitude: 60 degrees
- Blues_r is a reverse (ending with _r) version of the "Blues" colormap, which is a sequential colormap with shades of blue. Reverse colormaps typically start with light colors and progress to darker colors.
- norm2 = matplotlib.colors.LogNorm(vmin=gdp_df.GDP2015.dropna().min(), vmax=gdp_df.GDP2015.dropna().max()): This line creates another normalization object named norm2, but this time it uses a logarithmic scaling (LogNorm). This can be useful when working with data that spans a wide range of values, and you want to emphasize differences in smaller values. Similar to norm1, vmin and vmax are set based on the "GDP2015" column in the gdp_df DataFrame, excluding NaN values.

Map Chart Code (2)

```
shpfilename = shpreader.natural_earth(resolution='110m', category='cultural',
name='admin_0_countries’)
# category='cultural': This argument specifies the category of the data you want to retrieve. 'cultural' typically includes information related to political boundaries, cities, and other human-made features.
# name='admin_0_countries': This argument specifies the specific name of the shapefile you want to retrieve. 'admin_0_countries' refers to the shapefile containing the boundaries of countries at the highest level of political administration.
reader = shpreader.Reader(shpfilename)
countries = reader.records()
for country in countries:
    countryCode = country.attributes['ADM0_A3']
    if countryCode in gdp_df.index:
        ax2.add_geometries(country.geometry, ccrs.PlateCarree(),
                           facecolor=cmap(norm2(gdp_df.loc[countryCode].GDP2015)))
    # check various attributes to find the matching two-letter combinations
    nation = country.attributes['POSTAL']
    if nation not in happiness_df.index: nation = country.attributes['ISO_A2']
    if nation not in happiness_df.index: nation = country.attributes['WB_A2']
    if nation not in happiness_df.index and country.attributes['NAME'] == 'Norway': nation = 'NO'
    if nation in happiness_df.index:
        ax1.add_geometries(country.geometry, ccrs.PlateCarree(),
                           facecolor=cmap(norm1(happiness_df.loc[nation].Score)))

ax2.set_title("GDP 2015")
sm = plt.cm.ScalarMappable(norm=norm2, cmap=cmap)
cb = plt.colorbar(sm, ax=ax2)
cb.set_ticks([1e8, 1e9, 1e10, 1e11, 1e12, 1e13])

ax1.set_title("Happiness")
sm = plt.cm.ScalarMappable(norm=norm1, cmap=cmap)
cb = plt.colorbar(sm, ax=ax1)
cb.set_ticks([3, 4, 5, 6, 7, 8])
plt.show()
```

Map Chart Code (2)

- shpreader.natural_earth(): This function is used to obtain shapefile data from the Natural Earth dataset. The Natural Earth dataset is a public domain map dataset that provides geographic information for countries, states, cities, and other geographical features.
- resolution='110m': This argument specifies the resolution of the shapefile data you want to retrieve. In this case, '110m' indicates a medium level of detail, suitable for displaying country boundaries and major lakes and rivers. Other resolution options include '50m' (higher detail) and '10m' (the highest detail).
- category='cultural': This argument specifies the category of the data you want to retrieve. 'cultural' typically includes information related to political boundaries, cities, and other human-made features.
- name='admin_0_countries': This argument specifies the specific name of the shapefile from category you want to retrieve. 'admin_0_countries' refers to the shapefile containing the boundaries of countries at the highest level of political administration, under category=cultural.
- The records() method returns a list of records, where each record contains the geometry of a country (e.g., polygons representing the country's boundaries) along with associated attributes.
- cmap(norm2(...)) combines the normalization step (norm2) and the color mapping step (cmap) to determine the face colors of the country geometries based on GDP values, ensuring that the colors accurately represent the data.
- sm = plt.cm.ScalarMappable(norm=norm2, cmap=cmap): This line creates a ScalarMappable object sm that maps scalar data values to colors. It uses the normalization object norm2 and the colormap cmap to determine how to map data values to colors.
- cb = plt.colorbar(sm, ax=ax2): This line creates a colorbar cb associated with the ScalarMappable object sm. It specifies that the colorbar should be added to the subplot ax2.
- cb.set_ticks([1e8, 1e9, 1e10, 1e11, 1e12, 1e13]): This line sets the tick positions on the colorbar. It specifies where the tick marks should be placed along the colorbar. In this case, it sets the tick positions to powers of 10 (e.g., 1e8 represents 100 million, 1e9 represents 1 billion, and so on). These ticks represent the data values associated with the colors in the colorbar.

Map Chart Code (3)

- If you want to see what is included in the “countries” iterable, the following code may be used:

```
for country in countries:
        # Access country attributes and perform actions
        print(country.attributes) # Print all attributes for the current country
        print(f"Country Name: {country.attributes['NAME']}") # f string. This line prints the name of the current country. It does so by accessing the 'NAME' attribute from the country.attributes dictionary-like structure.
        print(f"ISO Code: {country.attributes['ISO_A3']}") # This line prints the ISO 3-letter country code of the current country. It does so by accessing the 'ISO_A3' attribute from the country.attributes dictionary-like structure.
        print("=" * 40) # This line prints a line of equal signs to separate the information about different countries visually. It's just for formatting and clarity.
```

Partial output:

{'featurecla': 'Admin-0 country', 'scalerank': 1, 'LABELRANK': 6, 'SOVEREIGNT': 'Fiji', 'SOV_A3': 'FJI', 'ADM0_DIF': 0, …
Country Name: Fiji
ISO Code: FJI

.
.
.