# Import numpy

import numpy as np

# Create array of DataFrame values: np\_vals

np\_vals = df.values

# Create new array of base 10 logarithm values: np\_vals\_log10

np\_vals\_log10 = np.log10(np\_vals)

# Create array of new DataFrame by passing df to np.log10(): df\_log10

df\_log10 = np.log10(df)

# Print original and new data containers

[print(x, 'has type', type(eval(x))) for x in ['np\_vals', 'np\_vals\_log10', 'df', 'df\_log10']]

==

# Zip the 2 lists together into one list of (key,value) tuples: zipped

zipped = list(zip(list\_keys, list\_values))

# Inspect the list using print()

print(zipped)

# Build a dictionary with the zipped list: data

data = dict(zipped)

# Build and inspect a DataFrame from the dictionary: df

df = pd.DataFrame(data)

print(df)

==

# Build a list of labels: list\_labels

list\_labels = ['year', 'artist', 'song', 'chart weeks']

# Assign the list of labels to the columns attribute: df.columns

df.columns = list\_labels

==

# Make a string with the value 'PA': state

state = 'PA'

# Construct a dictionary: data

data = {'state':state, 'city':cities}

# Construct a DataFrame from dictionary data: df

df = pd.DataFrame(data)

# Print the DataFrame

print(df)

==

<http://www.sidc.be/silso/infossntotdaily>

==

# Read in the file: df1

df1 = pd.read\_csv(data\_file)

# Create a list of the new column labels: new\_labels

new\_labels = ['year', 'population']

# Read in the file, specifying the header and names parameters: df2

df2 = pd.read\_csv(data\_file, header=0, names=new\_labels)

# Print both the DataFrames

print(df1)

print(df2)

==

# Read the raw file as-is: df1

df1 = pd.read\_csv(file\_messy)

# Print the output of df1.head()

print(df1.head())

# Read in the file with the correct parameters: df2

df2 = pd.read\_csv(file\_messy, delimiter=' ', header=3, comment='#')

# Print the output of df2.head()

print(df2.head())

# Save the cleaned up DataFrame to a CSV file without the index

df2.to\_csv(file\_clean, index=False)

# Save the cleaned up DataFrame to an excel file without the index

df2.to\_excel('file\_clean.xlsx', index=False)

==

# Create a plot with color='red'

df.plot(color='red')

# Add a title

plt.title('Temperature in Austin')

# Specify the x-axis label

plt.xlabel('Hours since midnight August 1, 2010')

# Specify the y-axis label

plt.ylabel('Temperature (degrees F)')

# Display the plot

plt.show()

==

# Plot all columns (default)

df.plot()

plt.show()

# Plot all columns as subplots

df.plot(subplots=True)

plt.show()

# Plot just the Dew Point data

column\_list1 = ['Dew Point (deg F)']

df[column\_list1].plot()

plt.show()

# Plot the Dew Point and Temperature data, but not the Pressure data

column\_list2 = ['Temperature (deg F)','Dew Point (deg F)']

df[column\_list2].plot()

plt.show()

==

# Create a list of y-axis column names: y\_columns

y\_columns = ['AAPL', 'IBM']

# Generate a line plot

df.plot(x='Month', y=y\_columns)

# Add the title

plt.title('Monthly stock prices')

# Add the y-axis label

plt.ylabel('Price ($US)')

# Display the plot

plt.show()

==

# Generate a scatter plot

df.plot(kind='scatter', x='hp', y='mpg', s=sizes)

# Add the title

plt.title('Fuel efficiency vs Horse-power')

# Add the x-axis label

plt.xlabel('Horse-power')

# Add the y-axis label

plt.ylabel('Fuel efficiency (mpg)')

# Display the plot

plt.show()

==

# Make a list of the column names to be plotted: cols

cols = ['weight', 'mpg']

# Generate the box plots

df[cols].plot(kind='box', subplots=True)

# Display the plot

plt.show()

==

# This formats the plots such that they appear on separate rows

fig, axes = plt.subplots(nrows=2, ncols=1)

# Plot the PDF

df.fraction.plot(ax=axes[0], kind='hist', bins=30, normed=True, range=(0,.3))

plt.show()

# Plot the CDF

df.fraction.plot(kind='hist', normed=True, cumulative=True, bins=30, ax=axes[1], range=(0,.3))

plt.show()

==

# Print the minimum value of the Engineering column

print(df['Engineering'].min())

# Print the maximum value of the Engineering column

print(df['Engineering'].max())

# Construct the mean percentage per year: mean

mean = df.mean(axis='columns')

# Plot the average percentage per year

mean.plot()

# Display the plot

plt.show()

==

# Print summary statistics of the fare column with .describe()

print(df['fare'].describe())

# Generate a box plot of the fare column

df.fare.plot(kind='box')

# Show the plot

plt.show()

==

# Print the number of countries reported in 2015

print(df['2015'].count())

# Print the 5th and 95th percentiles

print(df.quantile([0.05, 0.95]))

# Generate a box plot

years = ['1800','1850','1900','1950','2000']

df[years].plot(kind='box')

plt.show()

==

# Print the mean of the January and March data

print(january.mean(), march.mean())

# Print the standard deviation of the January and March data

print(january.std(), march.std())

==

# Compute the global mean and global standard deviation: global\_mean, global\_std

global\_mean = df.mean()

global\_std = df.std()

# Filter the US population from the origin column: us

us = df[df.origin=='US']

# Compute the US mean and US standard deviation: us\_mean, us\_std

us\_mean = us.mean()

us\_std = us.std()

# Print the differences

print(us\_mean - global\_mean)

print(us\_std - global\_std)

==

# Display the box plots on 3 separate rows and 1 column

fig, axes = plt.subplots(nrows=3, ncols=1)

# Generate a box plot of the fare prices for the First passenger class

titanic.loc[titanic['pclass'] == 1].plot(ax=axes[0], y='fare', kind='box')

# Generate a box plot of the fare prices for the Second passenger class

titanic.loc[titanic['pclass'] == 2].plot(ax=axes[1], y='fare', kind='box')

# Generate a box plot of the fare prices for the Third passenger class

titanic.loc[titanic['pclass'] == 3].plot(ax=axes[2], y='fare', kind='box')

# Display the plot

plt.show()

==

# Prepare a format string: time\_format

time\_format = '%Y-%m-%d %H:%M'

# Convert date\_list into a datetime object: my\_datetimes

my\_datetimes = pd.to\_datetime(date\_list, format = time\_format)

# Construct a pandas Series using temperature\_list and my\_datetimes: time\_series

time\_series = pd.Series(temperature\_list, index = my\_datetimes)

==

# Extract the hour from 9pm to 10pm on '2010-10-11': ts1

ts1 = ts0.loc['2010-10-11 21:00:00':'2010-10-11 22:00:00']

# Extract '2010-07-04' from ts0: ts2

ts2 = ts0.loc['July 4th, 2010']

# Extract data from '2010-12-15' to '2010-12-31': ts3

ts3 = ts0.loc['12/15/2010':'12/31/2010']

==

# Reindex without fill method: ts3

ts3 = ts2.reindex(ts1.index)

# Reindex with fill method, using forward fill: ts4

ts4 = ts2.reindex(ts1.index, method = 'ffill')

# Combine ts1 + ts2: sum12

sum12 = ts1 + ts2

# Combine ts1 + ts3: sum13

sum13 = ts1 + ts3

# Combine ts1 + ts4: sum14

sum14 = ts1 + ts4

==

# Downsample to 6 hour data and aggregate by mean: df1

df1 = df.Temperature.resample('6h').mean()

# Downsample to daily data and count the number of data points: df2

df2 = df.Temperature.resample('D').count()

==

# Extract temperature data for August: august

august = df.Temperature['August, 2010']

# Downsample to obtain only the daily highest temperatures in August: august\_highs

august\_highs = august.resample('D').max()

# Extract temperature data for February: february

february = df.Temperature['February, 2010']

# Downsample to obtain the daily lowest temperatures in February: february\_lows

february\_lows = february.resample('D').min()

==

# Extract data from 2010-Aug-01 to 2010-Aug-15: unsmoothed

unsmoothed = df['Temperature']['2010-8-1':'2010-8-15']

# Apply a rolling mean with a 24 hour window: smoothed

smoothed = unsmoothed.rolling(window=24).mean()

# Create a new DataFrame with columns smoothed and unsmoothed: august

august = pd.DataFrame({'smoothed':smoothed, 'unsmoothed':unsmoothed})

# Plot both smoothed and unsmoothed data using august.plot().

print(august.plot())

plt.show()

==

# Extract the August 2010 data: august

august = df['Temperature']['August, 2010']

# Resample to daily data, aggregating by max: daily\_highs

daily\_highs = august.resample('D').max()

# Use a rolling 7-day window with method chaining to smooth the daily high temperatures in August

daily\_highs\_smoothed = daily\_highs.rolling(window=7).mean()

print(daily\_highs\_smoothed)

==

# Strip extra whitespace from the column names: df.columns

df.columns = df.columns.str.strip()

# Extract data for which the destination airport is Dallas: dallas

dallas = df['Destination Airport'].str.contains('DAL')

# Compute the total number of Dallas departures each day: daily\_departures

daily\_departures = dallas.resample('D').sum()

# Generate the summary statistics for daily Dallas departures: stats

stats = daily\_departures.describe()

==

# Reset the index of ts2 to ts1, and then use linear interpolation to fill in the NaNs: ts2\_interp

ts2\_interp = ts2.reindex(ts1.index).interpolate(how='linear')

# Compute the absolute difference of ts1 and ts2\_interp: differences

differences = np.abs(ts1 - ts2\_interp)

# Generate and print summary statistics of the differences

print(differences.describe())

==

# Build a Boolean mask to filter for the 'LAX' departure flights: mask

mask = df['Destination Airport'] == 'LAX'

# Use the mask to subset the data: la

la = df[mask]

# Combine two columns of data to create a datetime series: times\_tz\_none

times\_tz\_none = pd.to\_datetime( la['Date (MM/DD/YYYY)'] + ' ' + la['Wheels-off Time'] )

# Localize the time to US/Central: times\_tz\_central

times\_tz\_central = times\_tz\_none.dt.tz\_localize('US/Central')

# Convert the datetimes from US/Central to US/Pacific

times\_tz\_pacific = times\_tz\_central.dt.tz\_convert('US/Pacific')

==

# Plot the raw data before setting the datetime index

df.plot()

plt.show()

# Convert the 'Date' column into a collection of datetime objects: df.Date

df.Date = pd.to\_datetime(df.Date)

# Set the index to be the converted 'Date' column

df.set\_index('Date', inplace=True)

# Re-plot the DataFrame to see that the axis is now datetime aware!

df.plot()

plt.show()

==

# Plot the summer data

df.Temperature['2010-Jun':'2010-Aug'].plot()

plt.show()

plt.clf()

# Plot the one week data

df.Temperature['2010-06-10':'2010-06-17'].plot()

plt.show()

plt.clf()

==

# Import pandas

import pandas as pd

# Read in the data file: df

df = pd.read\_csv(data\_file)

# Print the output of df.head()

print(df.head())

# Read in the data file with header=None: df\_headers

df\_headers = pd.read\_csv(data\_file, header=None)

# Print the output of df\_headers.head()

print(df\_headers.head())

==

# Split on the comma to create a list: column\_labels\_list

column\_labels\_list = column\_labels.split(',')

# Assign the new column labels to the DataFrame: df.columns

df.columns = column\_labels\_list

# Remove the appropriate columns: df\_dropped

df\_dropped = df.drop(list\_to\_drop, axis='columns')

# Print the output of df\_dropped.head()

print(df\_dropped.head())

==

# Convert the date column to string: df\_dropped['date']

df\_dropped['date'] = df\_dropped['date'].astype(str)

# Pad leading zeros to the Time column: df\_dropped['Time']

df\_dropped['Time'] = df\_dropped['Time'].apply(lambda x:'{:0>4}'.format(x))

# Concatenate the new date and Time columns: date\_string

date\_string = df\_dropped['date'] + df\_dropped['Time']

# Convert the date\_string Series to datetime: date\_times

date\_times = pd.to\_datetime(date\_string, format='%Y%m%d%H%M')

# Set the index to be the new date\_times container: df\_clean

df\_clean = df\_dropped.set\_index(date\_times)

# Print the output of df\_clean.head()

print(df\_clean.head())

==

# Print the dry\_bulb\_faren temperature between 8 AM and 9 AM on June 20, 2011

print(df\_clean.loc['2011-6-20 08:00':'2011-6-20 09:00', 'dry\_bulb\_faren'])

# Convert the dry\_bulb\_faren column to numeric values: df\_clean['dry\_bulb\_faren']

df\_clean['dry\_bulb\_faren'] = pd.to\_numeric(df\_clean['dry\_bulb\_faren'], errors='coerce')

# Print the transformed dry\_bulb\_faren temperature between 8 AM and 9 AM on June 20, 2011

print(df\_clean.loc['2011-6-20 08:00':'2011-6-20 09:00', 'dry\_bulb\_faren'])

# Convert the wind\_speed and dew\_point\_faren columns to numeric values

df\_clean['wind\_speed'] = pd.to\_numeric(df\_clean['wind\_speed'], errors='coerce')

df\_clean['dew\_point\_faren'] = pd.to\_numeric(df\_clean['dew\_point\_faren'], errors='coerce')

==

# Print the median of the dry\_bulb\_faren column

print(df\_clean['dry\_bulb\_faren'].median())

# Print the median of the dry\_bulb\_faren column for the time range '2011-Apr':'2011-Jun'

print(df\_clean.loc['2011-Apr':'2011-Jun', 'dry\_bulb\_faren'].median())

# Print the median of the dry\_bulb\_faren column for the month of January

print(df\_clean.loc['2011-Jan', 'dry\_bulb\_faren'].median())

==

# Downsample df\_clean by day and aggregate by mean: daily\_mean\_2011

daily\_mean\_2011 = df\_clean.resample('D').mean()

# Extract the dry\_bulb\_faren column from daily\_mean\_2011 using .values: daily\_temp\_2011

daily\_temp\_2011 = daily\_mean\_2011['dry\_bulb\_faren'].values

# Downsample df\_climate by day and aggregate by mean: daily\_climate

daily\_climate = df\_climate.resample('D').mean()

# Extract the Temperature column from daily\_climate using .reset\_index(): daily\_temp\_climate

daily\_temp\_climate = daily\_climate.reset\_index()['Temperature']

# Compute the difference between the two arrays and print the mean difference

difference = daily\_temp\_2011 - daily\_temp\_climate

print(difference.mean())

==

# Using df\_clean, when is sky\_condition 'CLR'?

is\_sky\_clear = df\_clean['sky\_condition']=='CLR'

# Filter df\_clean using is\_sky\_clear

sunny = df\_clean.loc[is\_sky\_clear]

# Resample sunny by day then calculate the max

sunny\_daily\_max = sunny.resample('D').max()

# See the result

sunny\_daily\_max.head()

==

# Using df\_clean, when does sky\_condition contain 'OVC'?

is\_sky\_overcast = df\_clean['sky\_condition'].str.contains('OVC')

# Filter df\_clean using is\_sky\_overcast

overcast = df\_clean.loc[is\_sky\_overcast]

# Resample overcast by day then calculate the max

overcast\_daily\_max = overcast.resample('D').max()

# See the result

overcast\_daily\_max.head()

==

# From previous steps

is\_sky\_clear = df\_clean['sky\_condition']=='CLR'

sunny = df\_clean.loc[is\_sky\_clear]

sunny\_daily\_max = sunny.resample('D').max()

is\_sky\_overcast = df\_clean['sky\_condition'].str.contains('OVC')

overcast = df\_clean.loc[is\_sky\_overcast]

overcast\_daily\_max = overcast.resample('D').max()

# Calculate the mean of sunny\_daily\_max

sunny\_daily\_max\_mean = sunny\_daily\_max.mean()

# Calculate the mean of overcast\_daily\_max

overcast\_daily\_max\_mean = overcast\_daily\_max.mean()

# Print the difference (sunny minus overcast)

print(sunny\_daily\_max\_mean - overcast\_daily\_max\_mean)

==

# Import matplotlib.pyplot as plt

import matplotlib.pyplot as plt

# Select the visibility and dry\_bulb\_faren columns and resample them: weekly\_mean

weekly\_mean = df\_clean[['visibility', 'dry\_bulb\_faren']].resample('W').mean()

# Print the output of weekly\_mean.corr()

print(weekly\_mean.corr())

# Plot weekly\_mean with subplots=True

weekly\_mean.plot(subplots=True)

plt.show()

==

# Using df\_clean, when is sky\_condition 'CLR'?

is\_sky\_clear = df\_clean['sky\_condition'] =='CLR'

# Resample is\_sky\_clear by day

resampled = is\_sky\_clear.resample('D')

# See the result

resampled

==

# From previous step

is\_sky\_clear = df\_clean['sky\_condition'] == 'CLR'

resampled = is\_sky\_clear.resample('D')

# Calculate the number of sunny hours per day

sunny\_hours = resampled.sum()

# Calculate the number of measured hours per day

total\_hours = resampled.count()

# Calculate the fraction of hours per day that were sunny

sunny\_fraction = sunny\_hours / total\_hours

==

# From previous steps

is\_sky\_clear = df\_clean['sky\_condition'] == 'CLR'

resampled = is\_sky\_clear.resample('D')

sunny\_hours = resampled.sum()

total\_hours = resampled.count()

sunny\_fraction = sunny\_hours / total\_hours

# Make a box plot of sunny\_fraction

sunny\_fraction.plot(kind='box')

plt.show()

==

# Resample dew\_point\_faren and dry\_bulb\_faren by Month, aggregating the maximum values: monthly\_max

monthly\_max = df\_clean[['dew\_point\_faren', 'dry\_bulb\_faren']].resample('M').max()

# Generate a histogram with bins=8, alpha=0.5, subplots=True

monthly\_max.plot(kind='hist', bins=8, alpha=0.5, subplots=True)

# Show the plot

plt.show()

==

# Extract the maximum temperature in August 2010 from df\_climate: august\_max

august\_max = df\_climate.loc['2010-Aug', 'Temperature'].max()

print(august\_max)

# Resample August 2011 temps in df\_clean by day & aggregate the max value: august\_2011

august\_2011 = df\_clean.loc['2011-Aug', 'dry\_bulb\_faren'].resample('D').max()

# Filter for days in august\_2011 where the value exceeds august\_max: august\_2011\_high

august\_2011\_high = august\_2011[august\_2011>august\_max]

# Construct a CDF of august\_2011\_high

august\_2011\_high.plot(kind='hist',normed=True,cumulative=True, bins=25)

# Display the plot

plt.show()

==