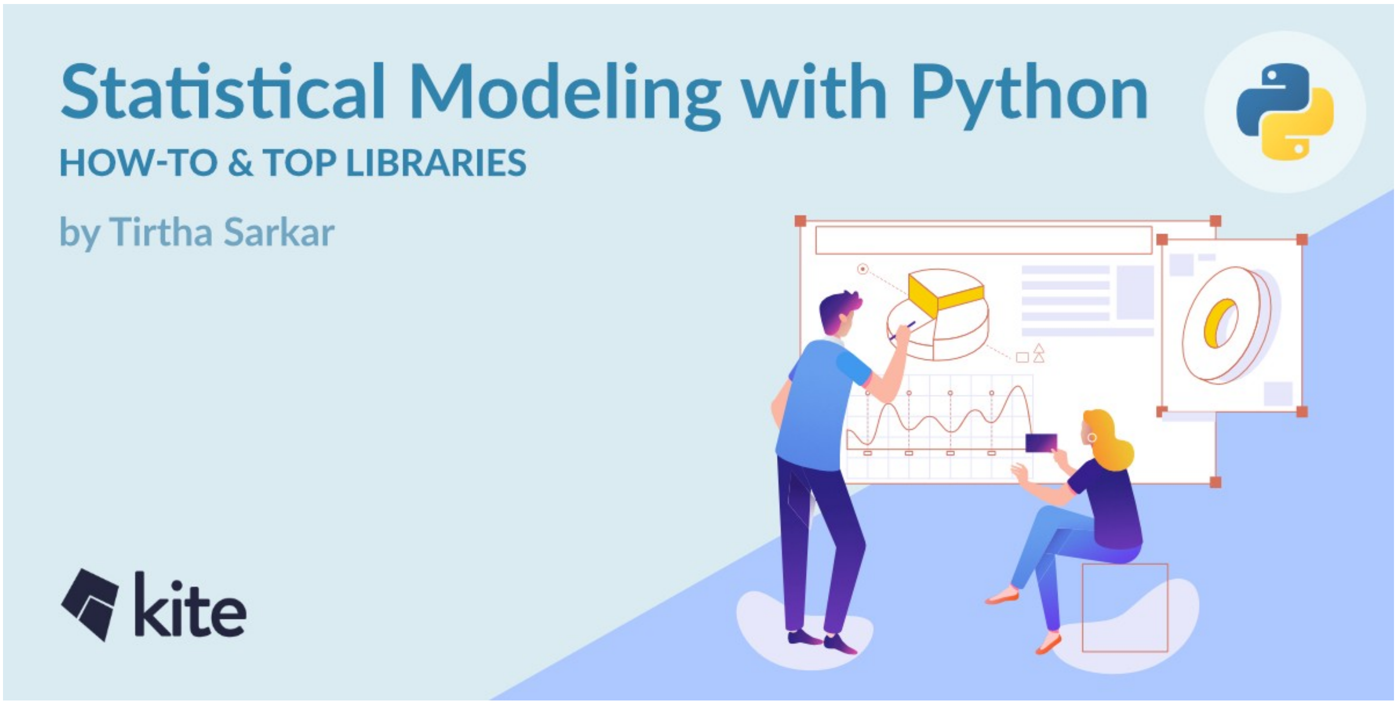
Statistical Modeling with Python: How-to & Top Libraries

Tirthajyoti Sarkar



Introduction: Why Python for data science

One of the most important factors driving Python's popularity as a statistical modeling language is its widespread use as the language of choice in data science and machine learning.

Today, there's a huge demand for data science expertise as more and more businesses apply it within their operations. Python offers the right mix of power, versatility, and support from its community to lead the way.

There are a number of reasons for data scientists to adopt Python as their preferred programming language, including:

- Open-source nature and active community
- Shorter learning curve and intuitive syntax
- Large collection of powerful and standardized libraries
- Powerful integration with fast, compiled languages (e.g. C/C++) for numerical computation primitives (as used in NumPy and pandas)
- Ease of integrating the core modeling process with database access, wrangling post-processing, such as visualization and web-serving
- Availability and continued development of Pythonic interfaces to Big Data frameworks such as Apache Spark or MongoDB
- Support and development of Python libraries by large and influential organizations such as Google or Facebook (e.g. TensorFlow and PyTorch)

It's worth noting, however, that sound statistical modeling occupies a central role in a data science stack, but some statistical modeling fundamentals often get overlooked, leading to poor analysis and bad decisions.

This article covers some of the essential statistical modeling frameworks and methods for Python, which can help us do statistical modeling and probabilistic computation.

Why these frameworks are necessary

While Python is most popular for data wrangling, visualization, general machine learning, deep learning and associated linear algebra (tensor and matrix operations), and web integration, its statistical modeling abilities are far less advertised. A large percentage of data scientists still use other special statistical languages such as R, MATLAB, or SAS over Python for their modeling and analysis.

While each of these alternatives offers their own unique blend of features and power for statistical analyses, it's useful for an up-and-coming data scientist to know more about various Python frameworks and methods that can be used for routine operations of descriptive and inferential statistics.

The biggest motivation for learning about these frameworks is that statistical inference and probabilistic modeling represent the bread and butter of data scientists' daily work. However, only by using such Python-based tools can a powerful end-to-end data science pipeline (a complete flow extending from data acquisition to final business decision generation) be built using a single programming language.

If using different statistical languages for various tasks, you may face some problems. For example:

- Conducting any web scraping and database access using SQL commands and Python libraries such as BeautifulSoup and SQLalchemy
- Cleaning up and preparing your data tables using Pandas, but then switching to R or SPSS for performing statistical tests and computing confidence intervals
- Using ggplot2 for creating a visualization, and then using a standalone LaTeX editor to type up the final analytics report

Switching between multiple programmatic frameworks makes the process cumbersome and error-prone.

What if you could do statistical modeling, analysis, and visualization all inside a core Python platform?

Let's see what frameworks and methods exist for accomplishing such tasks.

Want to Code **Faster**?
Kite is a plugin for PyCharm, Atom, Vim, VSCode, Sublime Text, and IntelliJ that uses machine learning to provide you with code completions in real time sorted by relevance. **Start coding faster today.**

Download Kite Free!

Start with NumPy

NumPy is the de-facto standard for numerical computation in Python, used as the base for building more advanced libraries for data science and machine learning applications such as TensorFlow or Scikit-learn. For numeric processing, NumPy is much faster than native Python code due to the vectorized implementation of its methods and the fact that many of its core routines are written in C (based on the CPython framework).

Although the majority of NumPy related discussions are focused on its linear algebra routines, it offers a decent set of statistical modeling functions for performing basic descriptive statistics and generating random variables based on various discrete and continuous distributions.

For example, let's create a NumPy array from a simple Python list and compute basic descriptive statistics like mean, median, standard deviation, quantiles, etc.

The code for this article may be found at Kite's Github repository.

```
import numpy as np

# Define a python list
a_list = [2, 4, -1, 5.5, 3.5, -2, 5, 4, 6.5, 7.5]

# Convert the list into numpy array
an_array = np.array(a_list)

# Compute and print various statistics
print('Mean:', an_array.mean())
print('Median:', np.median(an_array))
print('Range (Max - min):', np.ptp(an_array))
print('Standard deviation:', an_array.std())
print('80th percentile:', np.percentile(an_array, 80))
print('0.2-quantile:', np.quantile(an_array, 0.2))
```

The results are as follows:

```
Mean: 3.5
Median: 4.0
Range (Max - min): 9.5
Standard deviation: 2.9068883707497264
80th percentile: 5.699999999999999
0.2-quantile: 1.4000000000000001
```

You can also use NumPy to generate various random variables from statistical distributions, such as Binomial, Normal, Chi-square, etc. We'll discuss these in the context of the SciPy package, which is essentially a superset of NumPy.

Check out the NumPy docs for a detailed description of various other functions you can perform with NumPy.

Matplotlib and Seaborn for visualization

Data scientists should be able to quickly visualize various types of data for making observations, detecting outliers, gathering insights, investigation patterns, and most importantly, communicating the results to colleagues and management for business decision-making. We'll briefly mention two powerful Python libraries for the visualization task.

Matplotlib is the most widely used base library in Python for general visualization. There is extensive documentation on how to use this library and there's a bit of a learning curve to understand its core mechanics. Let's illustrate its utility with a simple example (we'll re-use the `an_array` NumPy object from the previous example showing that Matplotlib works natively with NumPy arrays).

```
import matplotlib.pyplot as plt
plt.plot(an_array)
plt.show()
```

These 3 lines of code result in a plot:

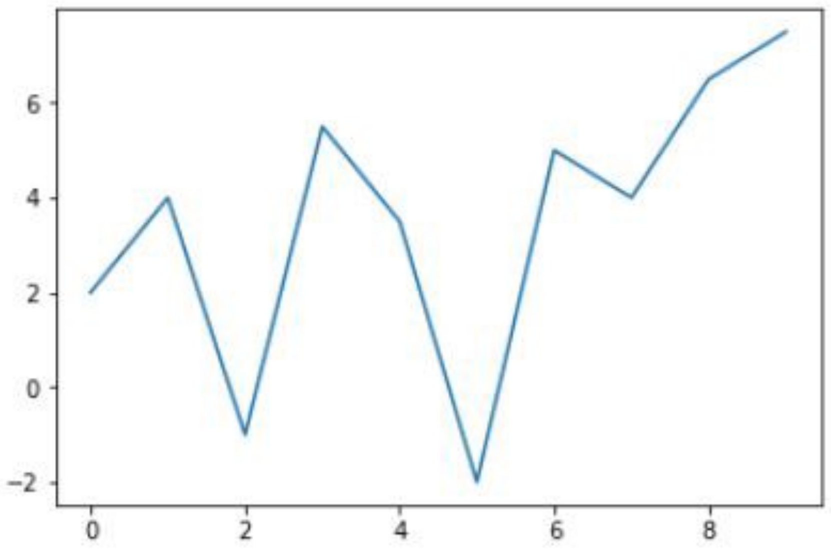


Fig 1: A simple plot with just 3 lines of code using Matplotlib.

It looks kind of barren, doesn't it? Let's add some bells and whistles to the plot, such as figure size, title, x- and y-axis labels and ticks (and control their font), line type, color, width, marker color, and size, etc.

```
plt.figure(figsize=(9, 5))
plt.title('A basic plot', fontsize=18)
plt.plot(an_array, color='blue', linestyle='--',
         linewidth=4, marker='o', markersize=20)
plt.xlabel('X-axis points', fontsize=14)
plt.ylabel('Y-axis points', fontsize=14)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(True)
plt.show()
```

The result looks like this:

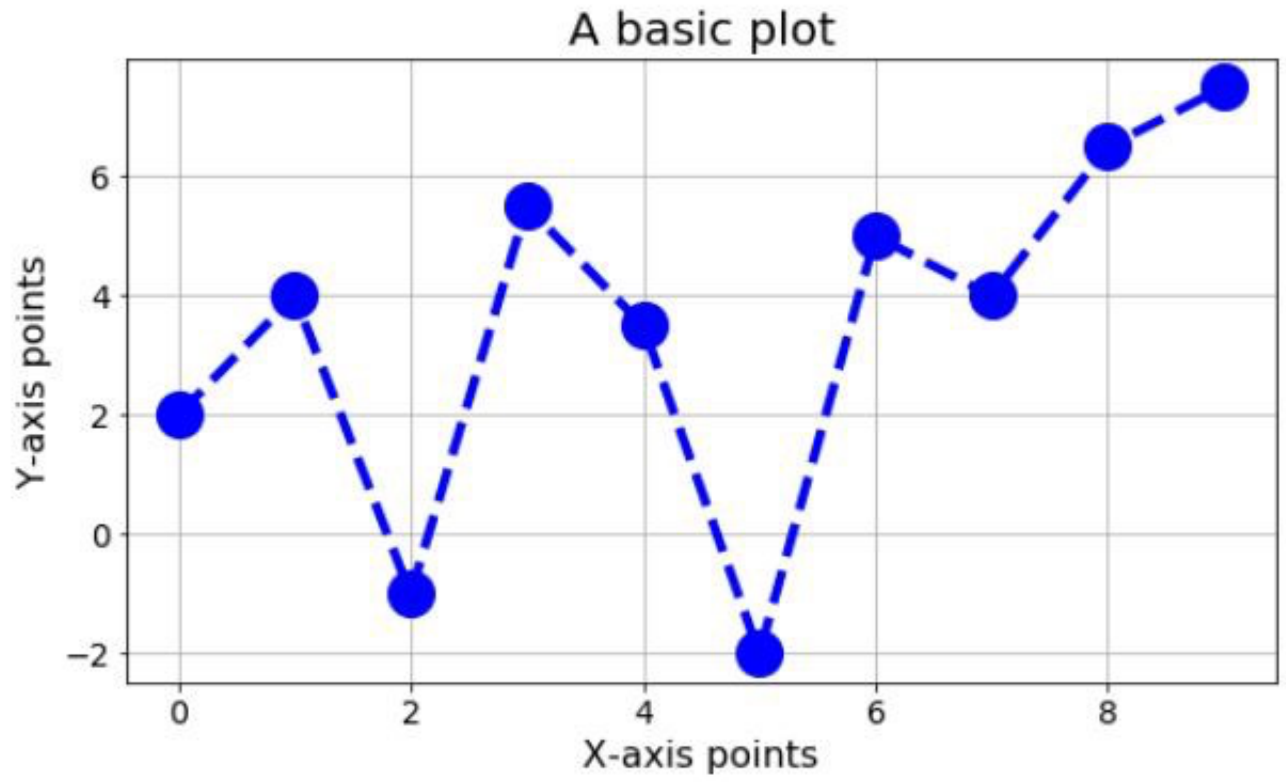


Fig 2: Plot with the same data as *Fig 1*, but with some embellishments added.

These were examples of the line charts. If you fancy other types of charts/plots, Matplotlib can help you there, too.

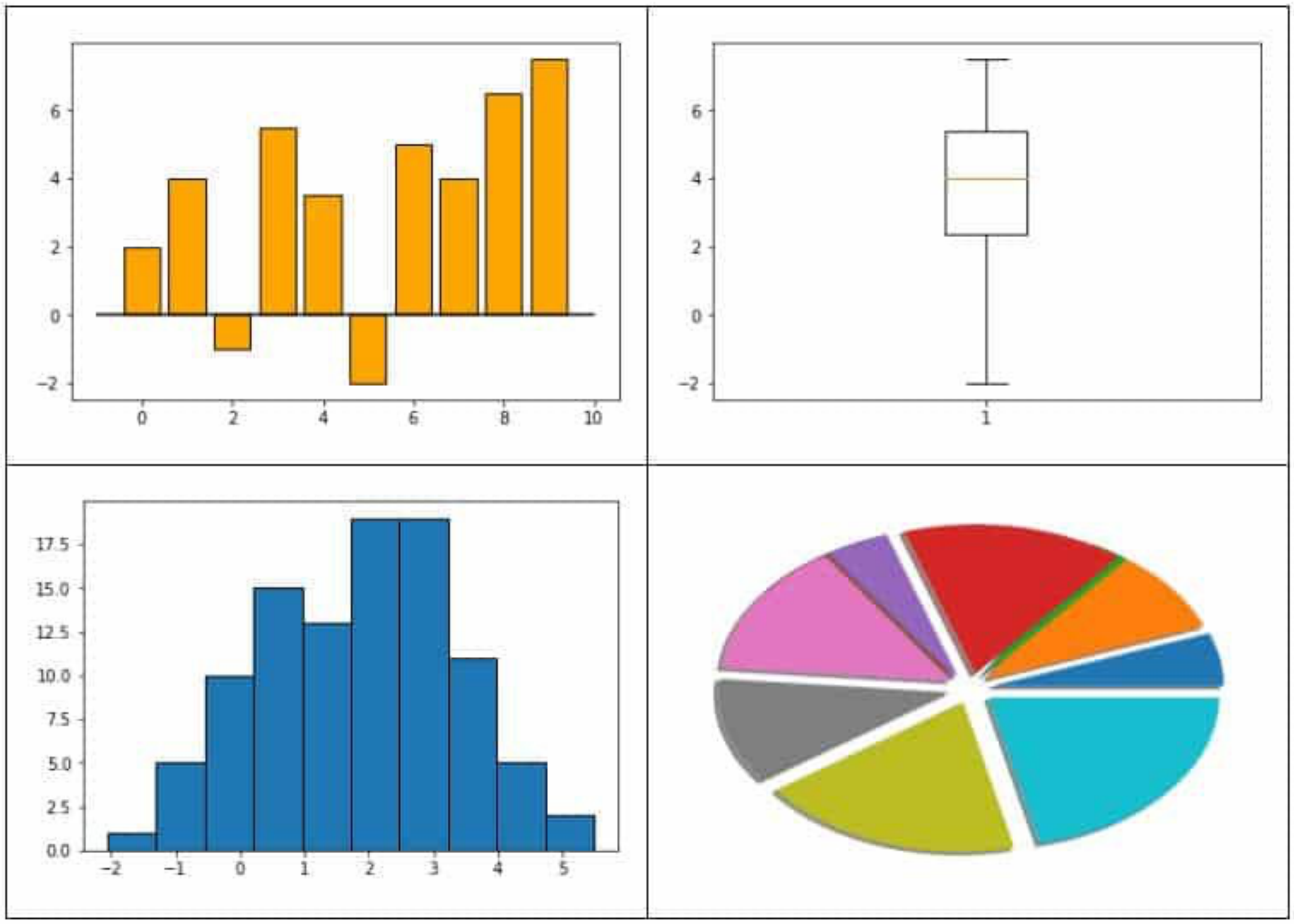


Fig 3: Matplotlib is used for generating a box plot, bar chart, histogram, and pie diagram. Except in the histogram, the same data is used from the `an_array` NumPy object.

Using Seaborn and Matplotlib

Seaborn is another powerful Python library which is built atop Matplotlib, providing direct APIs for dedicated statistical visualizations and is, therefore, a favorite among data scientists. Some of the advanced statistical modeling plots that Seaborn can make are:

- Heatmaps
- Violin plots
- Scatterplots with linear regression, fitting, and confidence intervals

- Pair plots and correlation plots showing mutual dependency among all the variables in a table of data (with multiple rows and columns)
- Plots with facets (i.e. visualizing a relationship between two variables which depend on more than one other variable)

Readers are encouraged to refer to the official Seaborn tutorial for more details.

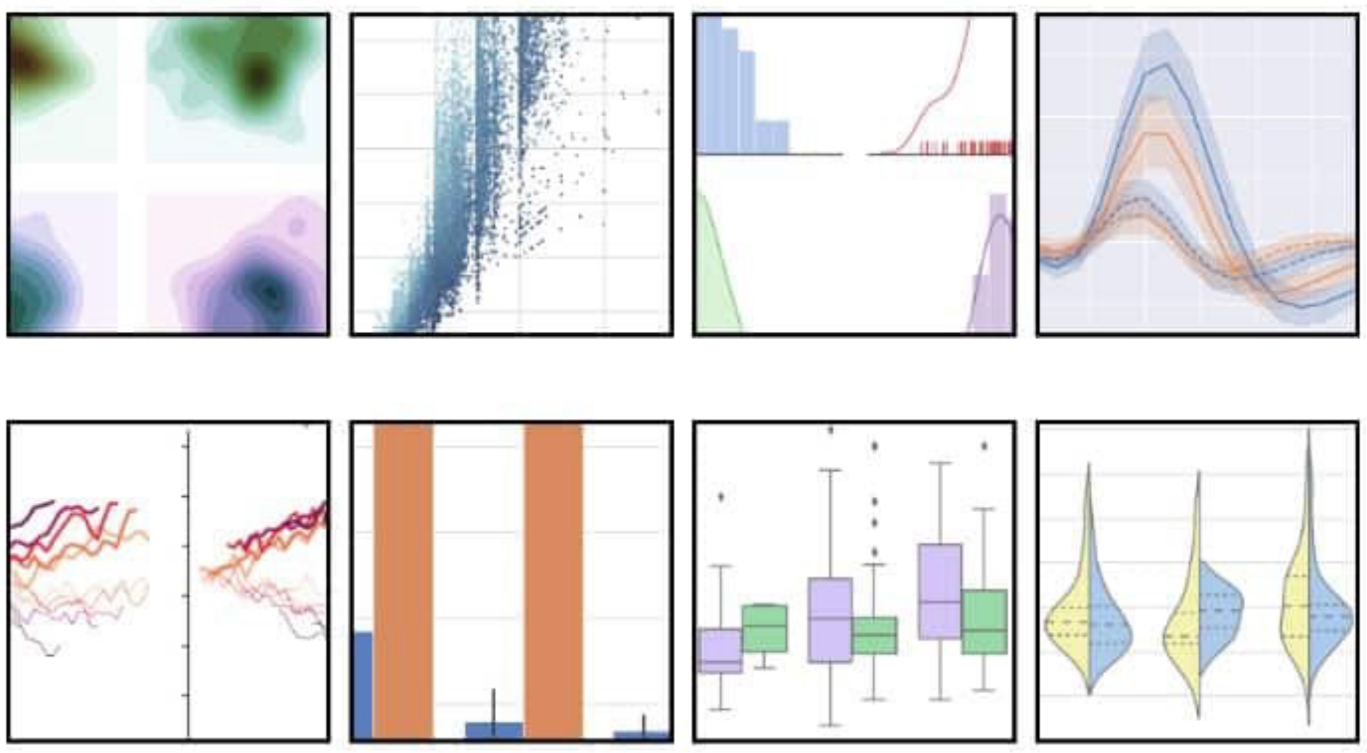


Fig 4: Example of Seaborn visualizations.

SciPy for inferential statistics

According to its website, SciPy (pronounced "Sigh Pie") is a, "Python-based ecosystem of open-source software for mathematics, science, and engineering." In fact, NumPy and Matplotlib are both components of this ecosystem.

NumPy
Base N-dimensional array package
SciPy library
Fundamental library for scientific computing
Matplotlib
Comprehensive 2D Plotting
IP[y]:
IPython
IPython
Enhanced Interactive Console
Sympy
Symbolic mathematics
pandas
Data structures & analysis

Fig 5: Core components of the SciPy ecosystem.

Specifically, in statistical modeling, SciPy boasts of a large collection of fast, powerful, and flexible methods and classes. Due to limited space, we're unable to go through examples of these functions, but here is a snapshot of the [page describing them](https://medium.com/kitepython/statistical-modeling-with-python-how-to-top-libraries-44f6c8cc7ece):

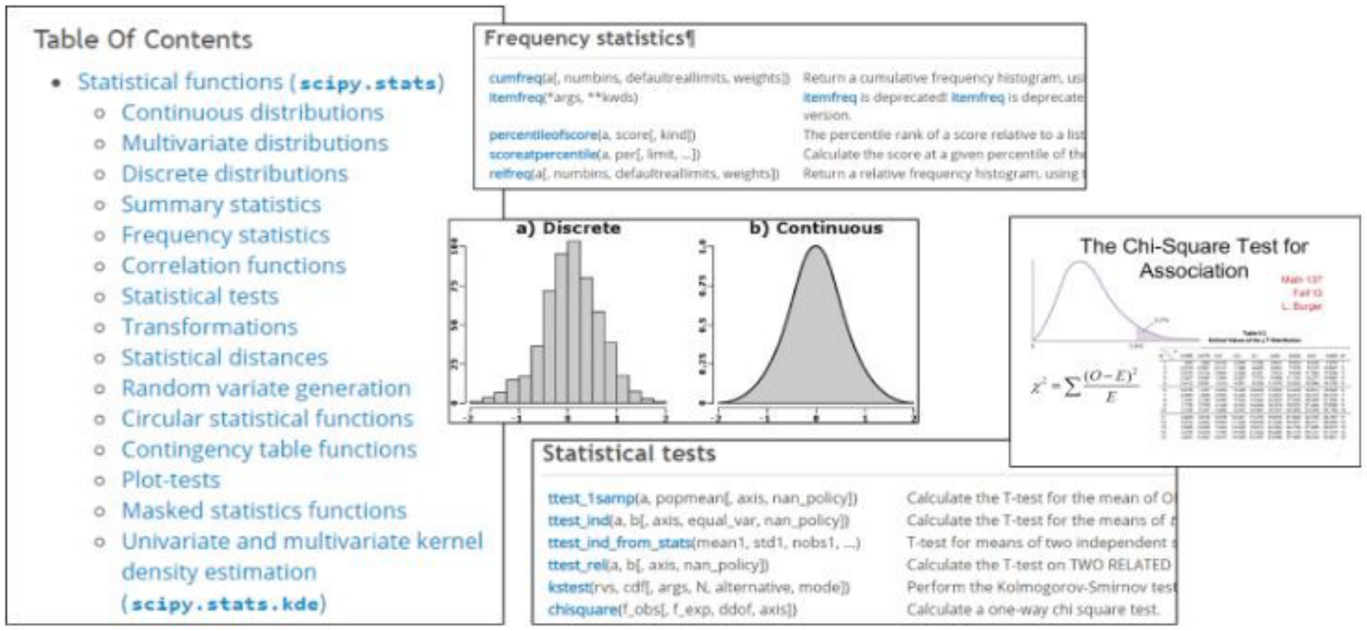


Table Of Contents

- Statistical functions (`scipy.stats`)
  - Continuous distributions
  - Multivariate distributions
  - Discrete distributions
  - Summary statistics
  - Frequency statistics
  - Correlation functions
  - Statistical tests
  - Transformations
  - Statistical distances
  - Random variate generation
  - Circular statistical functions
  - Contingency table functions
  - Plot-tests
  - Masked statistics functions
  - Univariate and multivariate kernel density estimation (`scipy.stats.kde`)

Frequency statistics¶

cumfreq(a[, numbins, defaultreallimits, weights])	Return a cumulative frequency histogram, usi
itemfreq(*args, **kwds)	itemfreq is deprecated! itemfreq is deprecate version.
percentileofscore(a, score[, kind])	The percentile rank of a score relative to a list
scoreatpercentile(a, per[, limit, ...])	Calculate the score at a given percentile of the
relfreq(a[, numbins, defaultreallimits, weights])	Return a relative frequency histogram, using t

$$\chi^2 = \sum \frac{(O-E)^2}{E}$$

Statistical tests

ttest_1samp(a, popmean[, axis, nan_policy])	Calculate the T-test for the mean of O
ttest_ind(a, b[, axis, equal_var, nan_policy])	Calculate the T-test for the means of t
ttest_ind_from_stats(mean1, std1, nobs1, ...)	T-test for means of two independent s
ttest_rel(a, b[, axis, nan_policy])	Calculate the T-test on TWO RELATED
kstest(rvs, cdf[, args, N, alternative, mode])	Perform the Kolmogorov-Smirnov test
chisquare(f_obs[, f_exp, ddof, axis])	Calculate a one-way chi square test.

Fig 6: Snapshot of various methods and routines available with Scipy.stats.

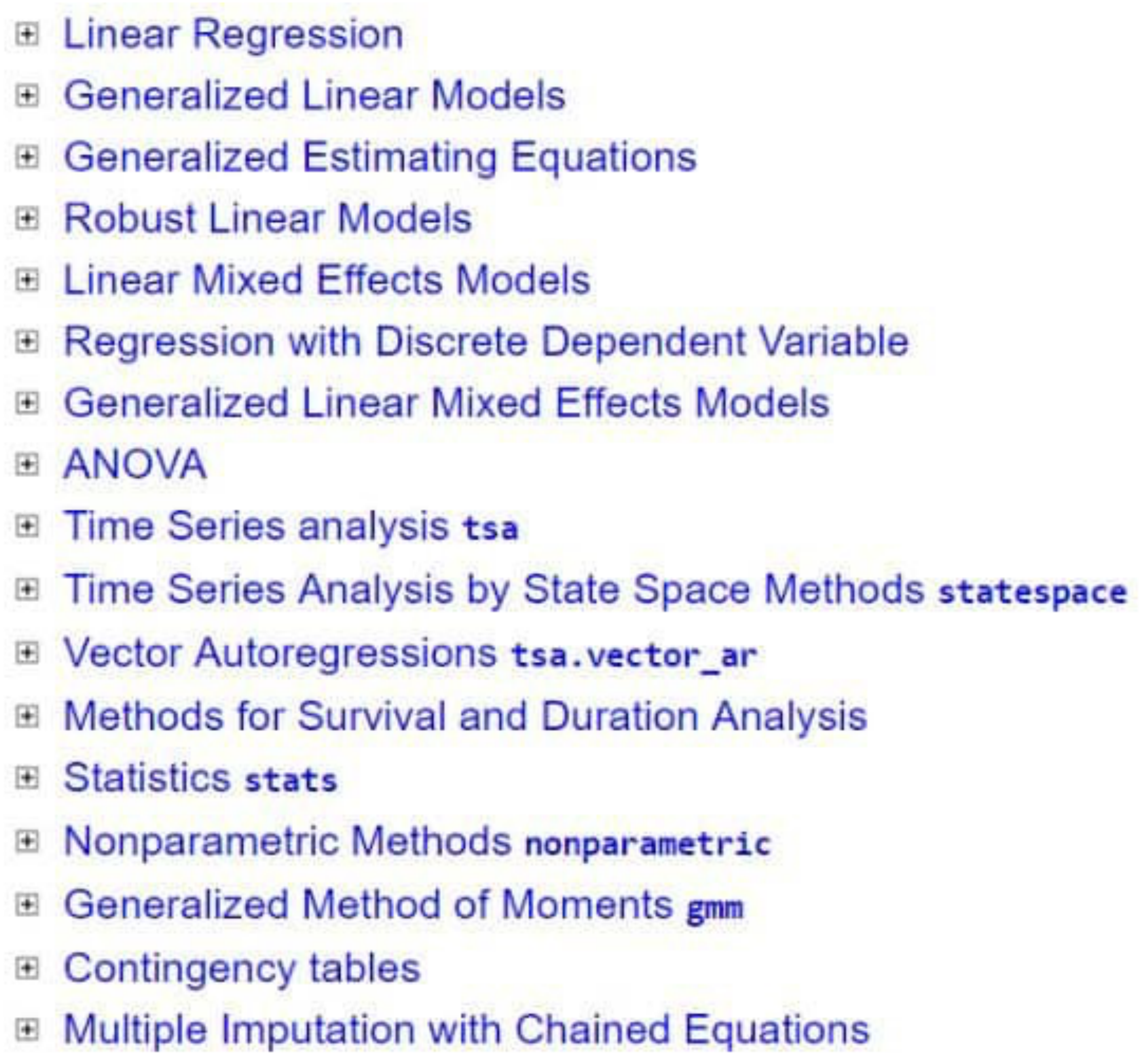
In short, you can do the following with SciPy:

- Generate random variables from a wide choice of discrete and continuous statistical distributions — binomial, normal, beta, gamma,

student's t, etc.
- Compute frequency and summary statistics of multi-dimensional datasets
- Run popular statistical tests such as t-test, chi-square, Kolmogorov-Smirnov, Mann-Whitney rank test, Wilcoxon rank-sum, etc.
- Perform correlation computations such as Pearson's coefficient, ANOVA, Theil-Sen estimation, etc.
- Compute statistical distance measures such as Wasserstein distance and energy distance.

Statsmodels for advanced modeling

Beyond computing basic descriptive and inferential statistics, we enter the realm of advanced modeling, for example, multivariate regression, generalized additive models, nonparametric tests, survivability and durability analysis, time series modeling, data imputation with chained equations, etc. The Statsmodels package allows you to perform all these analyses. Here is a snapshot of their capabilities.


⊞ Linear Regression
⊞ Generalized Linear Models
⊞ Generalized Estimating Equations
⊞ Robust Linear Models
⊞ Linear Mixed Effects Models
⊞ Regression with Discrete Dependent Variable
⊞ Generalized Linear Mixed Effects Models
⊞ ANOVA
⊞ Time Series analysis `tsa`
⊞ Time Series Analysis by State Space Methods `statespace`
⊞ Vector Autoregressions `tsa.vector_ar`
⊞ Methods for Survival and Duration Analysis
⊞ Statistics `stats`
⊞ Nonparametric Methods `nonparametric`
⊞ Generalized Method of Moments `gmm`
⊞ Contingency tables
⊞ Multiple Imputation with Chained Equations

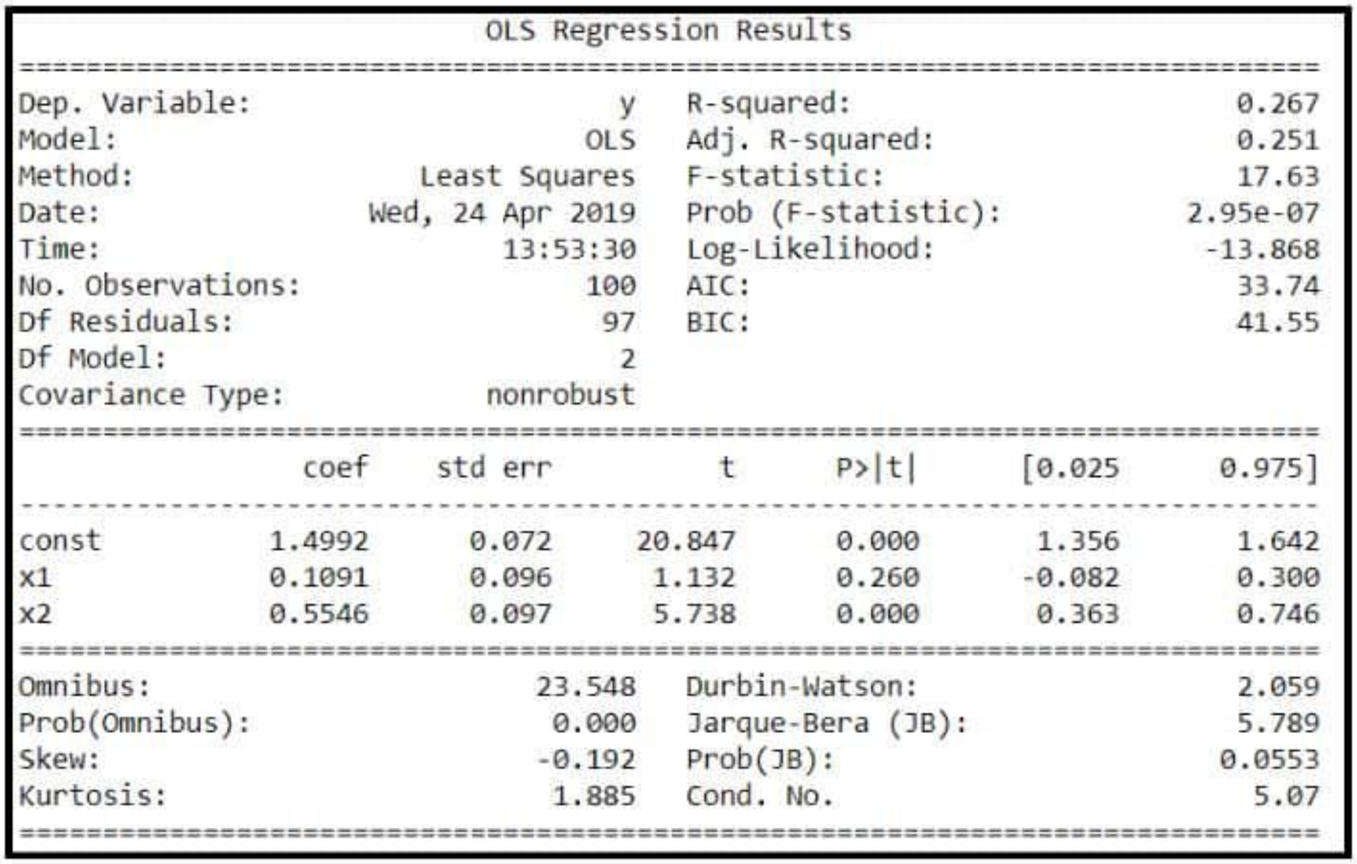
Statsmodels allow R-style formula syntax for many modeling APIs and also produce detailed tables with important values for statistical modeling, like p-values, adjusted R-square, etc. Here is a simple regression example using random numbers generated by NumPy and normally distributed errors.

Note: Be sure to install SciPy before using `statsmodels.api`

```
pip install scipy

import numpy as np
import statsmodels.api as sm
```

```
# Input variables
nobs = 100
X = np.random.random((nobs, 2))
X = sm.add_constant(X)

# Regression coefficients
beta = [1, .1, .5]

# Random errors
e = np.random.random(nobs)

# Output y
y = np.dot(X, beta) + e

# Fit the regression model
reg_model = sm.OLS(y, X).fit()

# Print the summary
print(reg_model.summary())
```

The result looks like the figure below. Your output will be different due to the random data. Note how detailed statistics (p-values, standard errors, and confidence intervals) are printed here along with the estimated regression coefficients.


```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.267
Model:                            OLS   Adj. R-squared:                  0.251
Method:                 Least Squares   F-statistic:                     17.63
Date:                Wed, 24 Apr 2019   Prob (F-statistic):           2.95e-07
Time:                        13:53:30   Log-Likelihood:                -13.868
No. Observations:                 100   AIC:                             33.74
Df Residuals:                      97   BIC:                             41.55
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.4992      0.072     20.847      0.000       1.356       1.642
x1             0.1091      0.096      1.132      0.260      -0.082       0.300
x2             0.5546      0.097      5.738      0.000       0.363       0.746
==============================================================================
Omnibus:                       23.548   Durbin-Watson:                   2.059
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                5.789
Skew:                          -0.192   Prob(JB):                       0.0553
Kurtosis:                       1.885   Cond. No.                         5.07
==============================================================================
```

Scikit-learn for statistical learning

Finally, we come to Scikit-learn, which is the most widely used Python library for classical machine learning.

But why is this included in the discussion of statistical modeling? This is because many classical machine learning (i.e. non-deep learning) algorithms can be classified as statistical learning techniques.

Scikit-learn features various classification, regression, and clustering algorithms, including support vector machines (SVM), random forests, gradient boosting, -means, and DBSCAN. It’s designed to interoperate seamlessly with the Python numerical and scientific libraries NumPy and SciPy, providing a range of supervised and unsupervised learning algorithms via a consistent interface.

The Scikit-learn library is also robust enough for use in production-grade systems because of its support community.

- Pipeline your statistical models in a chain
- Generate randomized regression and classification data for testing algorithms
- Perform various types of encoding of / transformation on the input data
- Hyperparameter search for complex algorithms like SVM

With Scikit-learn you can do advanced statistical learning tasks such as:

- Pipeline your statistical models in a chain
- Generate randomized regression and classification data for testing algorithms
- Perform various types of encoding of / transformation on the input data
- Hyperparameter search for complex algorithms like SVM

Conclusion

In this article, we covered a set of Python open-source libraries that form the foundation of statistical modeling, analysis, and visualization.

On the data side, these libraries work seamlessly with other data analytics and data engineering platforms such as Pandas and Spark (through PySpark).

For advanced machine learning tasks (e.g. deep learning), NumPy knowledge is directly transferable and applicable in popular packages such as TensorFlow and PyTorch.

On the visual side, libraries like Matplotlib integrate nicely with advanced dashboarding libraries like Bokeh and Plotly.

By focusing on these Python libraries and mastering the various methods and functions available, you will be well on your way to acquiring the data analytics, statistical modeling, and machine learning skills needed to excel as a data scientist.

About the author: ***Tirtha Sarkar*** *is a semiconductor technologist, data science author, and author of pydbgen, MLR, and doepy packages. He holds a Ph.D. in Electrical Engineering and M.S. in Data Analytics.*

The code for this article may be found at Kite's Github repository.

Originally published at https://kite.com on August 15, 2019.