# python_stat

May 13, 2020

## 1 Statistical analysis

### 1.1 Introduction

Statistical analysis usually encompasses 3 activities in a data science workflow. These are (a) descriptive analysis, (b) hypothesis testing and (c) statistical modeling. Descriptive analysis refers to a description of the data, which includes computing summary statistics and drawing plots. Hypothesis testing usually refers to statistically seeing if two (or more) groups are different from each other based on some metrics. Modeling refers to fitting a curve to the data to describe the relationship patterns of different variables in a data set

In terms of Python packages that can address these three tasks:

| Task | Packages |
|---|---|
| Descriptive statistics | pandas, numpy |
| Hypothesis testing | scipy, statsmodels |
| Modeling | statsmodels, lifelines |

### 1.2 Descriptive statistics

Descriptive statistics that are often computed are the mean, median, standard deviation, inter-quartile range, pairwise correlations, and the like. Most of these functions are available in `numpy`, and hence are available in `pandas`. We have already seen how we can compute these statistics and have even computed grouped statistics. For example, we will compute these using the diamonds dataset

[2]:
```
import numpy as np
import scipy as sc
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
diamonds = pd.read_csv('data/diamonds.csv.gz')
```

[4]:
```
diamonds.groupby('color')['price'].agg([np.mean, np.median, np.std])
```

[4]:
```
                mean  median          std
color
```

```
D      3169.954096  1838.0  3356.590935
E      3076.752475  1739.0  3344.158685
F      3724.886397  2343.5  3784.992007
G      3999.135671  2242.0  4051.102846
H      4486.669196  3460.0  4215.944171
I      5091.874954  3730.0  4722.387604
J      5323.818020  4234.0  4438.187251
```

There were other examples we saw yesterday along these lines.

## 1.3 Hypothesis testing

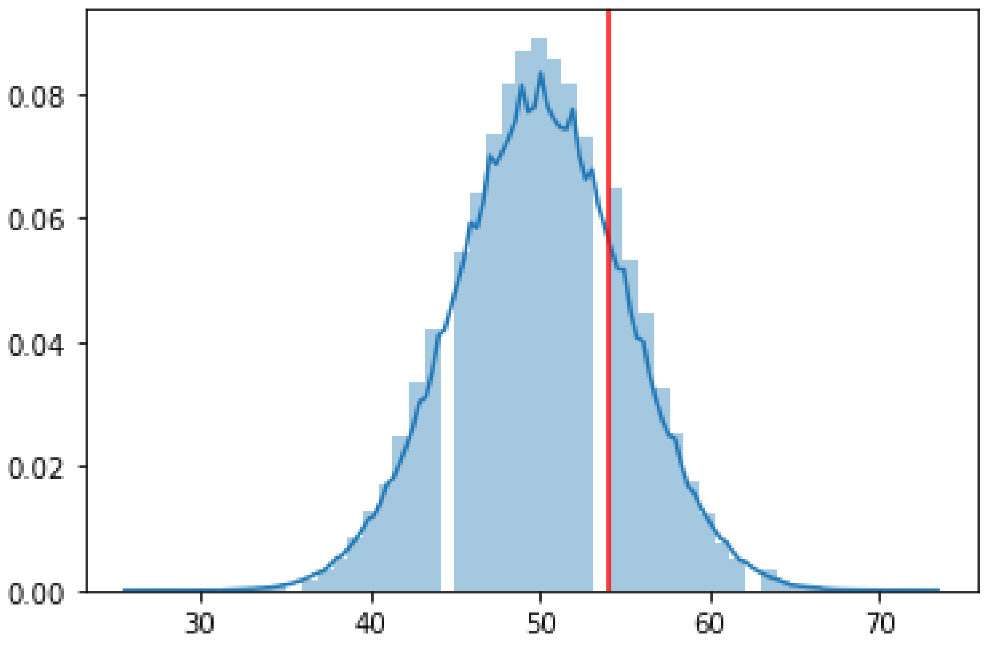
Hypothesis testing is one of the areas where statistics is often used. There are functions for a lot of the standard statistical tests in `scipy` and `statsmodels`. However, I'm going to take a little detour to see if we can get some understanding of hypothesis tests using the powerful simulation capabilities of Python.

You have a coin and you flip it 100 times. You get 54 heads. How likely is it that you have a fair coin?

We can simulate this process, which is random, using Python. The process of heads and tails from coin tosses can be modeled as a **binomial** distribution. So we can repeat this experiment many many times on our computer, assuming we have a fair coin, and then see how likely what we observed is.

[9]:
```python
rng = np.random.RandomState(205)
x = rng.binomial(100, 0.5, 100000)
sns.distplot(x)
plt.axvline(54, color = 'r')
```

[9]:
```
<matplotlib.lines.Line2D at 0x115c80580>
```



```
[10]: np.mean(x > 54)
```

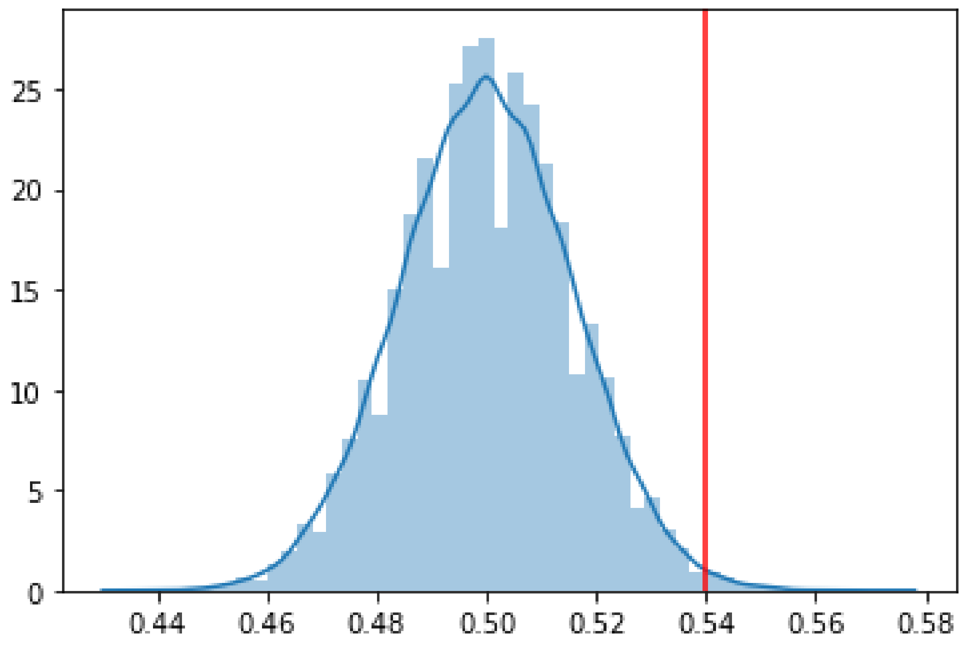
```
[10]: 0.18456
```

This is what would be considered the *p-value* for the test that the coin is fair.

What happens if we increase the number of tosses, and we look at the proportion of heads. We observe 54% heads.

```
[11]: rng = np.random.RandomState(205)
      x = rng.binomial(1000, 0.5, 100000)/1000
      sns.distplot(x)
      plt.axvline(0.54, color = 'r')
```

```
[11]: <matplotlib.lines.Line2D at 0x105b5ffd0>
```



```
[13]: np.mean(x > 0.54)
```

```
[13]: 0.00532
```

This is showing that as we increase the number of tosses, we get more robust signals about the coin. With 1000 tosses we see evidence that the coin is not fair, however small the unfairness is. This is related to the issue of statistical power. As we get more evidence, we can be more confident about what we are seeing.

We can use simulation to make better inference about differences between groups, or to get confidence intervals for a parameter of interest, without making assumptions about distributions or having to test for distributions. Python allows us to do that quickly and efficiently.

### 1.3.1 A permutation test

A permutation test is a 2-group test that asks whether two groups are different with respect to some metric. Let us look at a breast cancer proteomics experiment to illustrate this. The experimental data contains protein expression for over 12 thousand proteins, along with clinical data. We can ask, for example, whether a particular protein expression differs by ER status.

```
[14]: brca = pd.read_csv('data/brca.csv')
```

The idea about a permutation test is that, if there is truly no difference then it shouldn't make a difference if we shuffled the labels of ER status over the study individuals. That's literally what we will do. We will do this several times, and look at the average difference in expression each time. This will form the null distribution under our assumption of no differences by ER status. We'll then see where our observed data falls, and then be able to compute a p-value.

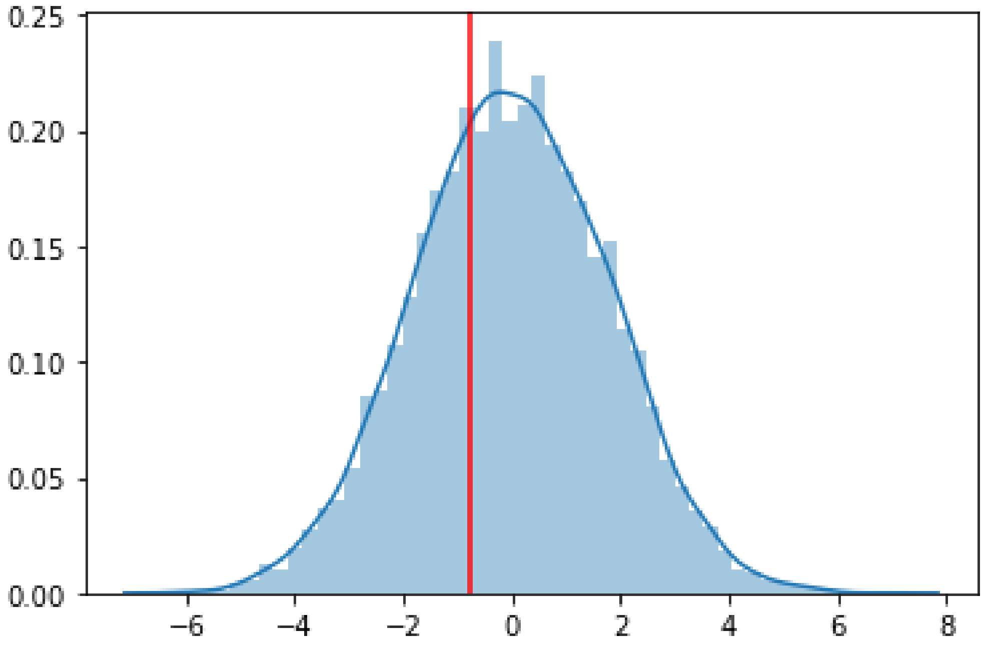
[23]:
```
nsim = 10000
rng = np.random.RandomState(294)
x = np.where(brca['ER Status']=='Positive', -1, 1)
y = brca[protein].to_numpy()

obs_diff = np.nanmean(y[x==1]) - np.nanmean(y[x==-1])

diffs = np.zeros(nsim)
for i in range(nsim):
    x1 = rng.permutation(x)
    diffs[i] = np.nanmean(y[x1==1]) - np.nanmean(y[x1 == -1])
```

[24]:
```
sns.distplot(diffs)
plt.axvline(x = obs_diff, color ='r')
```

[24]:
```
<matplotlib.lines.Line2D at 0x11b04d580>
```



[25]:
```
pval = np.mean(np.abs(diffs) > np.abs(obs_diff))
f"P-value from permutation test is {pval}"
```

[25]:
```
'P-value from permutation test is 0.6709'
```

### 1.3.2 Testing many proteins

We could do the permutation test all the proteins using the array operations in `numpy`

[26]:
```
expr_names = [u for u in list(brca.columns) if u.find('NP') > -1]

exprs = brca[expr_names]
```

[32]:
```
obs_diffs = exprs[x==1].mean(axis=0)-exprs[x==-1].mean(axis=0)
```

[34]:
```
nsim = 1000
diffs = np.zeros((nsim, exprs.shape[1]))
for i in range(nsim):
    x1 = rng.permutation(x)
    diffs[i,:] =exprs[x1==1].mean(axis=0) - exprs[x1==-1].mean(axis=0)
```

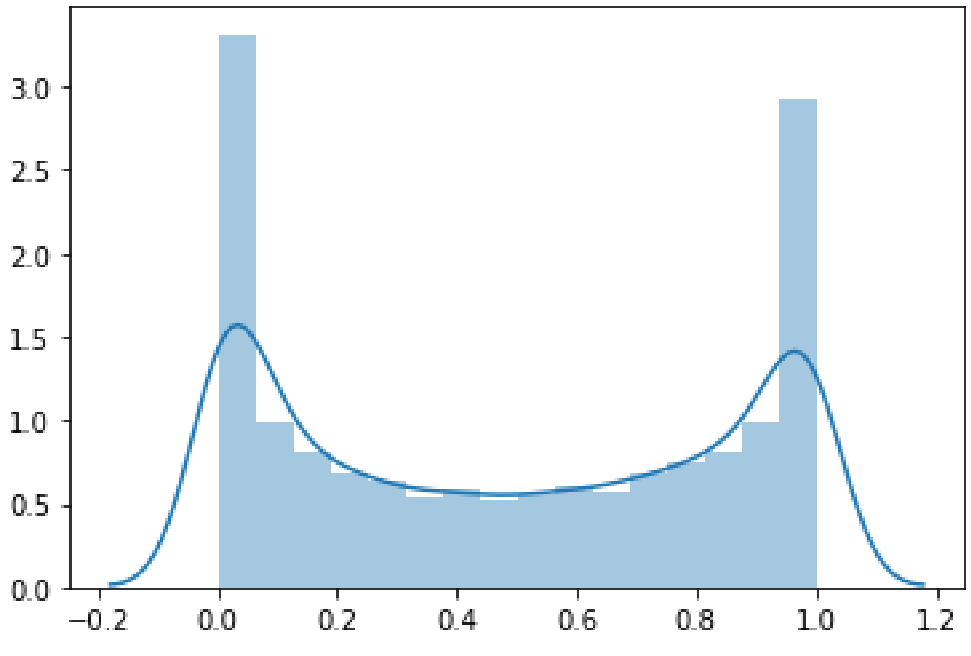
[46]:
```
pvals = np.zeros(exprs.shape[1])
len(pvals)
```

[46]:
```
12395
```

[47]:
```
for i in range(len(pvals)):
    pvals[i] = np.mean(diffs[:,i] > obs_diffs.to_numpy()[i])
```

[50]:
```
sns.distplot(pvals)
```

[50]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11a4866d0>
```



[51]:
```
pd.Series(pvals).describe()
```

```
[51]: count    12395.000000
      mean         0.490345
      std          0.370710
      min          0.000000
      25%          0.105000
      50%          0.486000
      75%          0.868000
      max          1.000000
      dtype: float64
```

```
[54]: exprs_shortlist = [u for i, u in enumerate(list(exprs.columns)) if pvals[i] < 0.
       ↪0001 ]
```

```
[55]: len(exprs_shortlist)
```

```
[55]: 513
```

### 1.4 Regression analysis

The regression modeling frameworks in Python are mainly in `statsmodels`, though some of it can be found in `scikit-learn` which we will see tomorrow. We will use the diamonds dataset for demonstration purposes. We will attempt to model the diamond price against several of the other diamond characteristics

```
[58]: import statsmodels.api as sm
      import statsmodels.formula.api as smf

      mod1 = smf.glm('price ~ carat + clarity + depth + cut + color', data =␣
       ↪diamonds).fit()
```

```
[59]: mod1.summary()
```

```
[59]: <class 'statsmodels.iolib.summary.Summary'>
      """
                       Generalized Linear Model Regression Results
      ==============================================================================
      Dep. Variable:                  price   No. Observations:                53940
      Model:                            GLM   Df Residuals:                    53920
      Model Family:              Gaussian   Df Model:                           19
      Link Function:             identity   Scale:                      1.3382e+06
      Method:                        IRLS   Log-Likelihood:            -4.5699e+05
      Date:              Wed, 13 May 2020   Deviance:                   7.2158e+10
      Time:                      13:21:00   Pearson chi2:                 7.22e+10
      No. Iterations:                   3
      Covariance Type:          nonrobust
      ================================================================================
      ====
                             coef    std err          z      P>|z|      [0.025
      0.975]
      --------------------------------------------------------------------------------
      ----
      Intercept         -6902.0434    245.309    -28.136      0.000   -7382.839
      -6421.247
      clarity[T.IF]      5415.0087     52.191    103.754      0.000    5312.717
      5517.301
      clarity[T.SI1]     3571.3831     44.613     80.053      0.000    3483.944
      3658.822
      clarity[T.SI2]     2623.0139     44.813     58.532      0.000    2535.181
      2710.846
      clarity[T.VS1]     4531.3874     45.570     99.437      0.000    4442.071
      4620.704
      clarity[T.VS2]     4214.9672     44.865     93.948      0.000    4127.033
      4302.901
      clarity[T.VVS1]    5068.3553     48.248    105.049      0.000    4973.792
      5162.919
      clarity[T.VVS2]    4963.7218     46.924    105.781      0.000    4871.752
      5055.692
      cut[T.Good]         644.1406     34.173     18.849      0.000     577.162
      711.119
      cut[T.Ideal]        982.1534     31.780     30.905      0.000     919.866
      1044.441
      cut[T.Premium]      849.9079     32.551     26.110      0.000     786.110
      913.706
      cut[T.Very Good]    833.2870     32.291     25.806      0.000     769.998
      896.576
      color[T.E]         -211.8364     18.316    -11.566      0.000    -247.734
      -175.939
      color[T.F]         -303.2741     18.509    -16.385      0.000    -339.551
      -266.997
      color[T.G]         -505.3602     18.127    -27.879      0.000    -540.888
      -469.832
      color[T.H]         -977.5332     19.281    -50.699      0.000   -1015.323
      -939.743
      color[T.I]        -1439.0796     21.655    -66.455      0.000   -1481.523
      -1396.636
      color[T.J]        -2323.8709     26.731    -86.935      0.000   -2376.263
      -2271.479
      carat              8885.8162     12.034    738.362      0.000    8862.229
      8909.403
      depth                -7.1602      3.727     -1.921      0.055     -14.464
      0.144
      ================================================================================
      ====
      """
```

This is the basic syntax for modeling in statsmodels. Let's go through and parse it.

One thing you notice is that we've written a formula inside the model

`mod1 = smf.glm('price ~ carat + clarity + depth + cut + color', data = diamonds).fit()`

This is based on another Python package, `patsy`, which allows us to write the model like this. This will read as "price depends on carat, clarity, depth, cut and color". Underneath a lot is going on.

1. color, clarity, and cut are all categorical variables. They actually need to be expanded into dummy variables, so we will have one column for each category level, which is 1 when the diamond is of that category and 0 otherwise.
2. An intercept terms is added
3. The dummy variables are concatenated to the continuous variables
4. The model is run

We can see the dummy variables using `pandas`

```
[60]: pd.get_dummies(diamonds)
```

```
[60]:        carat  depth  table  price     x     y     z  cut_Fair  cut_Good  \
0       0.23   61.5   55.0    326  3.95  3.98  2.43         0         0
1       0.21   59.8   61.0    326  3.89  3.84  2.31         0         0
2       0.23   56.9   65.0    327  4.05  4.07  2.31         0         1
3       0.29   62.4   58.0    334  4.20  4.23  2.63         0         0
4       0.31   63.3   58.0    335  4.34  4.35  2.75         0         1
…        …     …      …      …    …     …    …        …         …
53935   0.72   60.8   57.0   2757  5.75  5.76  3.50         0         0
53936   0.72   63.1   55.0   2757  5.69  5.75  3.61         0         1
53937   0.70   62.8   60.0   2757  5.66  5.68  3.56         0         0
53938   0.86   61.0   58.0   2757  6.15  6.12  3.74         0         0
53939   0.75   62.2   55.0   2757  5.83  5.87  3.64         0         0

       cut_Ideal  …  color_I  color_J  clarity_I1  clarity_IF  clarity_SI1  \
0              1  …        0        0           0           0            0
1              0  …        0        0           0           0            1
2              0  …        0        0           0           0            0
3              0  …        1        0           0           0            0
4              0  …        0        1           0           0            0
…            …  …       …       …           …           …          …
53935          1  …        0        0           0           0            1
53936          0  …        0        0           0           0            1
53937          0  …        0        0           0           0            1
53938          0  …        0        0           0           0            0
53939          1  …        0        0           0           0            0

       clarity_SI2  clarity_VS1  clarity_VS2  clarity_VVS1  clarity_VVS2
0                1            0            0             0             0
1                0            0            0             0             0
2                0            1            0             0             0
```

```
3                0            0            1             0             0
4                1            0            0             0             0
…           …          …          …            …            …
53935            0            0            0             0             0
53936            0            0            0             0             0
53937            0            0            0             0             0
53938            1            0            0             0             0
53939            1            0            0             0             0

[53940 rows x 27 columns]
```