Basic data analysis with Python

Programa de Doctorat en Economia, Universitat de Barcelona

David Moriña

Random number generation

Basic random number generation

```python
from random import seed, random

seed(1234)
print(random(), random(), random())
```

```
0.9664535356921388 0.4407325991753527 0.007491470058587191
```

Random number generation with *NumPy*

```python
import numpy as np

seed(1234)
s = np.random.normal(size=10000)
```

```python
np.mean(s)
```

```
-0.0020574019048463215
```

```python
np.std(s)
```

```
1.0031003039770017
```

Random number generation with *NumPy*

```
import numpy as np

seed(1234)
s = np.random.binomial(n=10, p=0.5, size=10000)
print(s)
```

```
[6 4 5 ... 5 6 3]
```

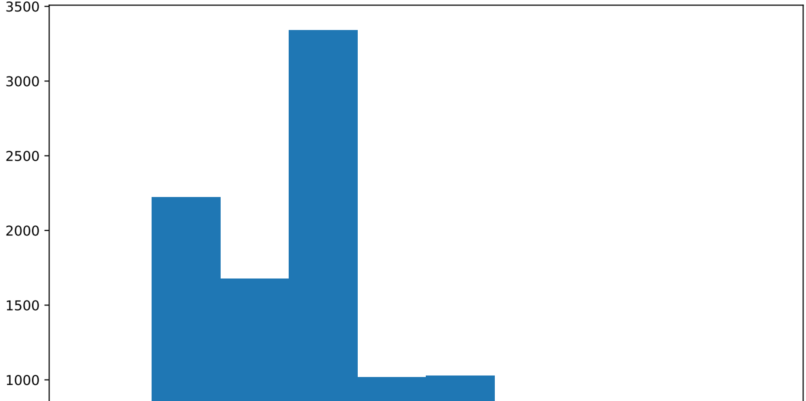
Random number generation with *NumPy*

```
import numpy as np

seed(1234)
s = np.random.poisson(lam=5, size=10000)
np.mean(s)
```

```
5.0326
```

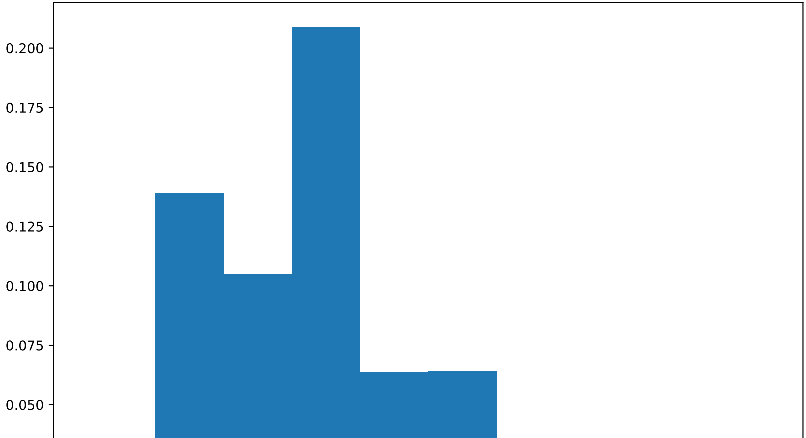
Random number generation with *NumPy*

```
import matplotlib.pyplot as plt

plt.hist(s)

plt.show()
```



Random number generation with *NumPy*

```
import matplotlib.pyplot as plt

plt.hist(s, density=True)
plt.show()
```



Random number generation with *NumPy*

More information on random number generation:
https://numpy.org/doc/stable/reference/random/

Basic inference

Comparing means

One sample:

```
from scipy.stats import ttest_1samp

scipy.stats.ttest_1samp(a, popmean, axis=0,
 ↪  nan_policy='propagate',
                        alternative='two-sided', ...,
 ↪  keepdims=False)
```

Two independent samples:

```
from scipy.stats import ttest_ind

scipy.stats.ttest_ind(a, b, axis=0, equal_var=True,
 ↪  nan_policy='propagate',
                      permutations=None,
 ↪  random_state=None, alternative='two-sided',
                      trim=0, ..., keepdims=False)
```

Comparing means

Are average fares different among genders?

```
from scipy.stats import ttest_ind
import pandas as pd
import os

os.chdir("/home/dmorina/Insync/dmorina@ub.edu/OneDrive
↪  Biz/Docència/UB/2023-2024/PyEcon/2. Intermediate
↪  Python/examples/")
titanic = pd.read_csv("titanic.csv")

res = ttest_ind(titanic.Fare[titanic.Sex=="male"],
↪  titanic.Fare[titanic.Sex=="female"])
print(res)
```

```
TtestResult(statistic=-5.529140269385719, pvalue=4.23086787
```

Comparing means

ANalysis Of Variance

```
from scipy.stats import f_oneway

scipy.stats.f_oneway(samples, axis=0)
```

Comparing means

In the Titanic dataset, is the average fare different by class?

```
from scipy.stats import f_oneway

f_oneway(titanic.Fare[titanic.Pclass==1],
↪   titanic.Fare[titanic.Pclass==2],
↪   titanic.Fare[titanic.Pclass==3])
```

```
F_onewayResult(statistic=242.34415651744814, pvalue=1.03137
```

Comparing means

In the Titanic dataset, is the average fare different by class?

```
from scipy.stats import tukey_hsd

res = tukey_hsd(titanic.Fare[titanic.Pclass==1],
↪  titanic.Fare[titanic.Pclass==2],
↪  titanic.Fare[titanic.Pclass==3])
print(res)
```

```
Tukey's HSD Pairwise Group Comparisons (95.0% Confidence In
Comparison  Statistic  p-value  Lower CI  Upper CI
 (0 - 1)     63.493     0.000    54.069    72.916
 (0 - 2)     70.479     0.000    62.809    78.149
 (1 - 0)    -63.493     0.000   -72.916   -54.069
 (1 - 2)      6.987     0.108    -1.133    15.106
 (2 - 0)    -70.479     0.000   -78.149   -62.809
 (2 - 1)     -6.987     0.108   -15.106     1.133
```

Chi-squared test

```
from scipy.stats import chi2_contingency

chi2_contingency(table1, correction=False,
 ↪  lambda_=None)
```

Chi-squared test

Are survival and class independent?

```
table1 = pd.crosstab(titanic['Survived'],
 ↪  titanic['Pclass'])
table1
```

Pclass Survived	1	2	3
0	80	97	372
1	136	87	119

Chi-squared test

Are survival and class independent?

```
from scipy.stats import chi2_contingency

chi2_contingency(table1, correction=False)
```

```
Chi2ContingencyResult(statistic=102.88898875696056, pvalue=
       [ 82.90909091,  70.62626263, 188.46464646]]))
```

Generalized Linear Models

Linear regression model

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

model = sm.OLS(Y, X)
results = model.fit()
results.summary()
```

Linear regression model

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

Y = titanic.Fare
X = titanic[['Age', 'Pclass']]
X = sm.add_constant(X)
model = sm.OLS(Y, X, missing='drop')
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==========================================================
Dep. Variable:                   Fare   R-squared:
Model:                            OLS   Adj. R-squared:
Method:                 Least Squares   F-statistic:
Date:                Tue, 19 Mar 2024   Prob (F-statistic):
Time:                        15:57:28   Log-Likelihood:
No. Observations:                 714   AIC:
```

Logistic regression model

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

Y = titanic.Survived
X = titanic[['Age', 'Pclass']]
X = sm.add_constant(X)
model = sm.GLM(Y, X, family=sm.families.Binomial(),
↪  missing='drop')
results = model.fit()
print(results.summary())
```

```
                 Generalized Linear Model Regression Result
=============================================================
Dep. Variable:                 Survived   No. Observations:
Model:                              GLM   Df Residuals:
Model Family:                  Binomial   Df Model:
Link Function:                    Logit   Scale:
Method:                            IRLS   Log-Likelihood:
```

Discrete dependent variable

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

Y = titanic.Survived
X = titanic[['Age', 'Pclass']]
X = sm.add_constant(X)
model = sm.GLM(Y, X, family=sm.families.Poisson(),
 ↪  missing='drop')
results = model.fit()
print(results.summary())
```

```
                 Generalized Linear Model Regression Result
==================================================================
Dep. Variable:                 Survived   No. Observations:
Model:                              GLM   Df Residuals:
Model Family:                   Poisson   Df Model:
Link Function:                      Log   Scale:
Method:                            IRLS   Log-Likelihood:
```

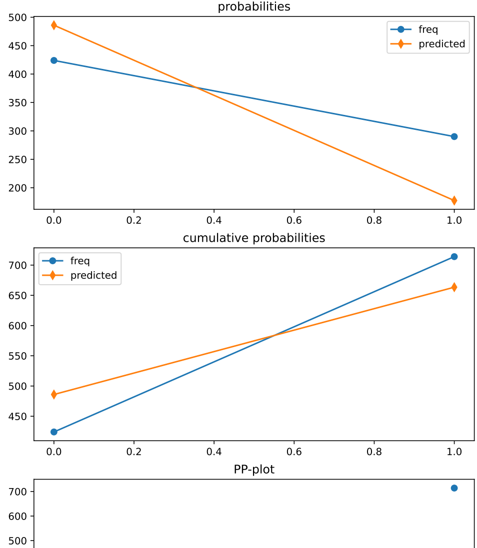
Discrete dependent variable

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

model = sm.poisson("Survived ~ Age+Pclass",
 ↪  data=titanic).fit()
print(model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.720374
         Iterations 5
                          Poisson Regression Results
==================================================================
Dep. Variable:                 Survived   No. Observations:
Model:                          Poisson   Df Residuals:
Method:                             MLE   Df Model:
Date:                  Tue, 19 Mar 2024   Pseudo R-squ.:
Time:                          15:57:28   Log-Likelihood:
converged:                         True   LL-Null:
```

Diagnostics

```
dia_model = model.get_diagnostic()
dia_model.plot_probs()
```



Discrete dependent variable

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

Y = titanic.Survived
X = titanic[['Age', 'Pclass']]
X = sm.add_constant(X)
model = sm.GLM(Y, X,
 ↪  family=sm.families.NegativeBinomial(),
 ↪  missing='drop')
results = model.fit()
print(results.summary())
```

```
                 Generalized Linear Model Regression Result
==========================================================
Dep. Variable:                 Survived   No. Observations:
Model:                              GLM   Df Residuals:
Model Family:          NegativeBinomial   Df Model:
Link Function:                      Log   Scale:
```

Discrete dependent variable

```
import numpy as np
from statsmodels.discrete.truncated_model import
 ↪  HurdleCountModel

Y = titanic.Survived
X = titanic[['Age', 'Pclass']]
X = sm.add_constant(X)
model = HurdleCountModel(Y, X, missing='drop')
results = model.fit()
print(results.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.580579
         Iterations: 12
         Function evaluations: 17
         Gradient evaluations: 17
         Current function value: 0.541325
         Iterations: 0
```

Discrete dependent variable

```
import numpy as np
import statsmodels.api as sm
from statsmodels.discrete.count_model import
 ↪  ZeroInflatedPoisson

titanic2 = titanic.dropna()
Y = titanic2.Survived
X = titanic2[['Age', 'Pclass']]
X = sm.add_constant(X)
model = ZeroInflatedPoisson(Y, X)
results = model.fit()
print(results.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.926245
         Iterations: 34
         Function evaluations: 40
         Gradient evaluations: 40
```

Prediction

```
import numpy as np

data = [[1, 25, 1], [1, 25, 2], [1, 25, 3]]
ex = pd.DataFrame(data, columns=['Intercept', 'Age',
↪ 'Pclass'])
results.predict(ex, which="mean")
```

```
0    0.783927
1    0.660883
2    0.557152
dtype: float64
```

Prediction

```python
import numpy as np
import matplotlib.pyplot as plt

pred = results.predict(ex, which="mean")
plt.scatter(x=["1st class", "2nd class", "3rd
 ↪  class"], y=pred)
```

```
<matplotlib.collections.PathCollection at 0x7eec03caec80>
```