Introduction to Python
for Data Science

Alexis Bogroff

July 13, 2022

Presenter

Alexis Bogroff
Lecturer and Mentor on Data Science

- 4 years teaching Data Science, Python, Git, Linux, VBA at ESILV, Sorbonne, Dauphine, UPEC, Openclassrooms
- 1 year Data Scientist/Engineer at Pléiade Asset Management
- Multiple experiences in banks, medium entreprises and startups, in the public and private sector

Measures

- Centrality:
  - Goal: representation of the majority's value
  - Mean (average): average age, mean size
  - Median: median salary, median patrimony
- Dispersion:
  - Goal: majority's spread (variation) around the central value
  - Standard deviation (sqrt variance): financial markets volatility
  - Interquartile Range (IQR)
  - Min-Max: job proposal salary

- Sensible to extreme values
  - Age: good representation
  - Patrimony: biased, not representative

$$mean = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

	name	age	group	country	patrimony
0	ju	25	1	fr	3000
1	ma	36	1	gb	7000
2	lo	40	2	gb	2000
3	fi	18	3	fr	60000
4	na	25	3	es	8000000
5	pe	40	3	it	4000

```python
mean_age = df['age'].mean()
int(mean_age)
```

✓ 0.2s

```
30
```

```python
mean_patrimony = df['patrimony'].mean()
int(mean_patrimony)
```

✓ 0.3s

```
1346000
```

Pandas - Median

- Insensitive to extreme values
    - Age: good representation
    - Patrimony: good representation of the majority

Order values, then:

$$median = \frac{x_{center_2} - x_{center_1}}{2}$$

	name	age	group	country	patrimony
0	ju	25	1	fr	3000
1	ma	36	1	gb	7000
2	lo	40	2	gb	2000
3	fi	18	3	fr	60000
4	na	25	3	es	8000000
5	pe	40	3	it	4000

```python
median_age = df['age'].median()
int(median_age)
```

✓ 0.2s

```
30
```

```python
median_patrimony = df['patrimony'].median()
int(median_patrimony)
```

✓ 0.2s

```
5500
```

Pandas - Standard Deviation (std)

- Sensible to extreme values
- In the unit of the varaible
- Interpretable
  - Age: good representation
  - Patrimony: Patrimony: biased, not representative

$$\overline{x} = mean(values)$$

$$std = \sqrt{\frac{(x_1 - \overline{x}) + \ldots + (x_n - \overline{x})}{n}}$$

	name	age	group	country	patrimony
0	ju	25	1	fr	3000
1	ma	36	1	gb	7000
2	lo	40	2	gb	2000
3	fi	18	3	fr	60000
4	na	25	3	es	8000000
5	pe	40	3	it	4000

```python
median_age = df['age'].median()
int(median_age)
```

✓ 0.2s

```
30
```

```python
median_patrimony = df['patrimony'].median()
int(median_patrimony)
```

✓ 0.2s

```
5500
```

Pandas - Interquartile Range (iqr)

- Insensitive to extreme values
- In the unit of the varaible
- Interpretable
  - Age: good representation
  - Patrimony: quite good representation

$$iqr = 3rd\_quantile - 1st\_quantile$$

	name	age	patrimony
0	ma	36	7000
1	lo	40	2000
2	fi	18	5000
3	na	25	8000000
4	pe	40	4000

```
iqr_age = df_temp['age'].quantile(.75) \
        - df_temp['age'].quantile(.25)
int(iqr_age)
```
✓ 0.2s

```
15
```

```
iqr_patrimony = df_temp['patrimony'].quantile(.75) \
              - df_temp['patrimony'].quantile(.25)
int(iqr_patrimony)
```
✓ 0.5s [1]

```
3000
```

```
# What is a quantile, example:
# These are exact values when 25% and 75% points exist
# Otherwise, linear interpolation
print(int(df_temp['patrimony'].quantile(.25)))
print(int(df_temp['patrimony'].quantile(.75)))
```
✓ 0.4s

```
4000
7000
```

[1] Quantile values 1st: 25%, 2nd: 50%, 3rd: 75% - after ordering

Pandas - Min Max

- Sensible to extreme values
- In the unit of the variable
- Interpretable
- Easy to compute
- Idea of max range

$$min{-}max = max(values){-}min(values)$$

	name	age	patrimony
0	ma	36	7000
1	lo	40	2000
2	fi	18	5000
3	na	25	8000000
4	pe	40	4000

```python
min_max_age = df['age'].max() - df['age'].min()
min_max_age
```

✓ 0.2s

```
22
```

```python
min_max_patrimony = df['patrimony'].max() \
                    - df['patrimony'].min()
min_max_patrimony
```

✓ 0.2s

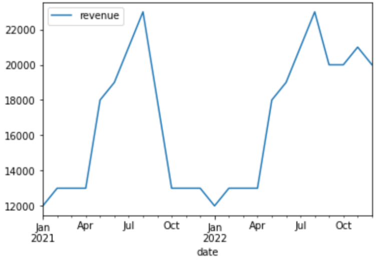
```
7998000
```

Patterns Analysis

- Univariate Analysis
  - Time Series
    - Trend
    - Seasonality
    - Auto-correlation
  - Other quantitative variables
  - Qualitative variables
- Multivariate Analysis (between variables)
  - Quantitative variables
    - Linear
    - Non-Linear
  - Qualitative variables

- Time Series
  - Generate plot using Pandas DataFrame method

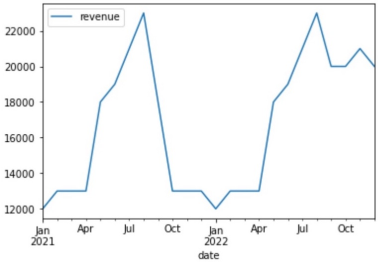
```
df_time.plot()
✓ 0.1s
<AxesSubplot:xlabel='date'>
```



	revenue
date	
2021-01-01	12000
2021-02-01	13000
2021-03-01	13000
2021-04-01	13000
2021-05-01	18000
2021-06-01	19000
2021-07-01	21000
2021-08-01	23000
2021-09-01	18000
2021-10-01	13000
2021-11-01	13000
2021-12-01	13000
2022-01-01	12000
2022-02-01	13000
2022-03-01	13000
2022-04-01	13000
2022-05-01	18000
2022-06-01	19000
2022-07-01	21000
2022-08-01	23000
2022-09-01	20000
2022-10-01	20000
2022-11-01	21000
2022-12-01	20000

Univariate Analysis

- Time Series
  - Compute overall trend




```python
revenue_first = df_time['revenue'].iloc[0]
revenue_last = df_time['revenue'].iloc[-1]
n_dates = len(df_time)
(revenue_last - revenue_first) / n_dates
```

✓ 0.2s

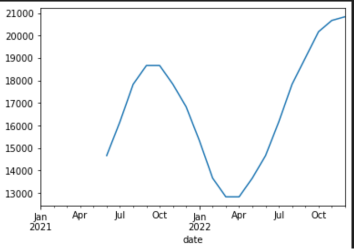
```
333.3333333333333
```

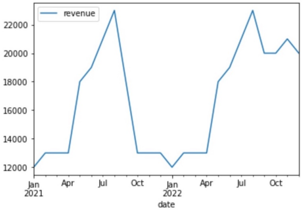
- Time Series
  - Compute rolling mean
    - Get trend along time
    - Smoothen curve, easier to read
    - + info vs overall trend
    - Delay first dates

```python
N_MONTHS = 6
rolling_mean = df_time['revenue'].rolling(window=N_MONTHS).mean()
rolling_mean.plot()
```

✓ 0.1s

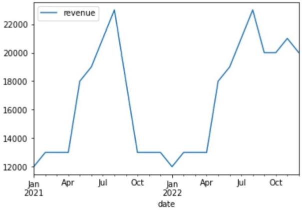
```
<AxesSubplot:xlabel='date'>
```





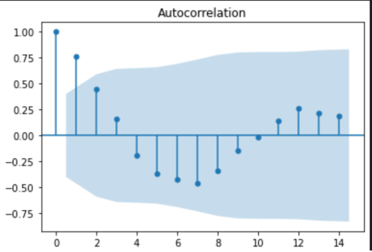
Univariate Analysis

- Time Series
  - Autocorrelogram
    - Correlation between dates (lags)




```python
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```
✓ 0.4s

```python
plot_acf(df_time['revenue'])
None
```
✓ 0.6s



Univariate Analysis

- Other quantitative variables
  - Descriptive statistics

	transaction	price
0	transac_olp	15
1	transac_ixh	17
2	transac_qkh	20
3	transac_qlz	20
4	transac_mal	19
5	transac_fjh	17
6	transac_rdn	20
7	transac_oaj	20
8	transac_taz	23
9	transac_tgs	17

```
df_quanti['price'].describe()
```

✓ 0.2s

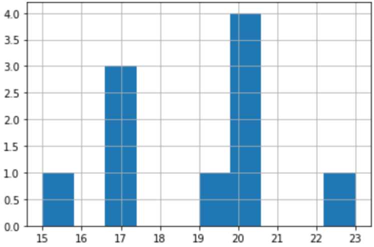
```
count    10.000000
mean     18.800000
std       2.299758
min      15.000000
25%      17.000000
50%      19.500000
75%      20.000000
max      23.000000
Name: price, dtype: float64
```

Univariate Analysis

- Other quantitative variables
  - Histogram
    - Overview
    - Extreme values
    - All information

	transaction	price
0	transac_olp	15
1	transac_ixh	17
2	transac_qkh	20
3	transac_qlz	20
4	transac_mal	19
5	transac_fjh	17
6	transac_rdn	20
7	transac_oaj	20
8	transac_taz	23
9	transac_tgs	17

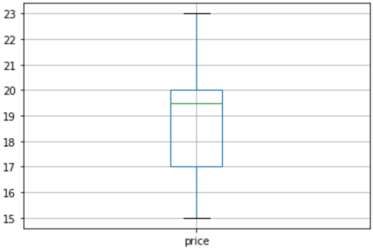
```
df_quanti['price'].hist()
✓ 0.1s
<AxesSubplot:>
```



- Other quantitative variables
  - Boxplot
    - Overview
    - Extreme values
    - Contracted information
    - Quartiles
    - Outliers (seaborn)

```
df_quanti[['price']].boxplot()
✓ 0.7s

<AxesSubplot:>
```



	transaction	price
0	transac_olp	15
1	transac_ixh	17
2	transac_qkh	20
3	transac_qlz	20
4	transac_mal	19
5	transac_fjh	17
6	transac_rdn	20
7	transac_oaj	20
8	transac_taz	23
9	transac_tgs	17

Univariate Analysis

- Qualitative variables
  - Information on categories

	transaction	client_type
0	transac_olp	veggie
1	transac_ixh	veggie
2	transac_qkh	vegan
3	transac_qlz	omnivorous
4	transac_mal	vegan
5	transac_fjh	veggie
6	transac_rdn	veggie
7	transac_oaj	omnivorous
8	transac_taz	vegan
9	transac_tgs	veggie

```
print(df_quali['client_type'].nunique())
print(df_quali['client_type'].unique())
```

✓ 0.3s

```
3
['veggie' 'vegan' 'omnivorous']
```

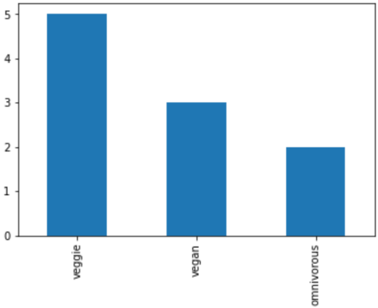
```
counts = df_quali['client_type'].value_counts()
counts
```

✓ 0.3s

```
veggie        5
vegan         3
omnivorous    2
Name: client_type, dtype: int64
```

- Qualitative variables
  - Bar plot
    - Overview
    - Ordinal variables: (small, medium, large companies)

```
counts.plot.bar()
None
✓ 0.6s
```



	transaction	client_type
0	transac_olp	veggie
1	transac_ixh	veggie
2	transac_qkh	vegan
3	transac_qlz	omnivorous
4	transac_mal	vegan
5	transac_fjh	veggie
6	transac_rdn	veggie
7	transac_oaj	omnivorous
8	transac_taz	vegan
9	transac_tgs	veggie

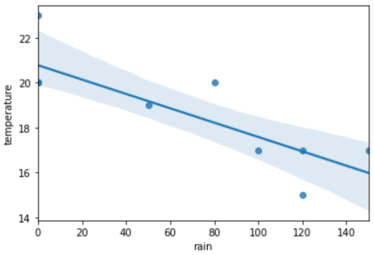
- Quantitative variables with linear relation
  - Scatter plot
  - Linear regression
  - Relation / link

	rain	temperature
0	120	15
1	120	17
2	80	20
3	0	20
4	50	19
5	100	17
6	0	20
7	0	20
8	0	23
9	150	17

```python
import seaborn as sns
sns.regplot(x='rain', y='temperature', data=df_multi_quanti)
```

✓ 0.1s

```
<AxesSubplot:xlabel='rain', ylabel='temperature'>
```

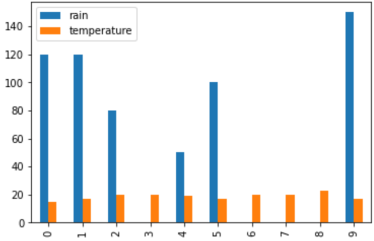


Multivariate Analysis

- Quantitative variables with linear relation
  - Bar plot raw data
  - More interesting when Time Series (not here)

```
df_multi_quanti.plot.bar()
✓ 0.1s
<AxesSubplot:>
```

	rain	temperature
0	120	15
1	120	17
2	80	20
3	0	20
4	50	19
5	100	17
6	0	20
7	0	20
8	0	23
9	150	17



Multivariate Analysis

- Quantitative variables with linear relation
  - Correlation matrix
  - Pearson correlation (linear)
  - Symetric matrix

	rain	temperature
0	120	15
1	120	17
2	80	20
3	0	20
4	50	19
5	100	17
6	0	20
7	0	20
8	0	23
9	150	17

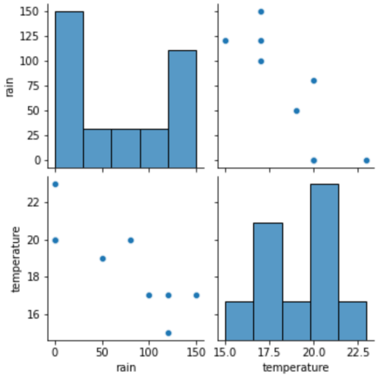
```python
# Pearson correlation for linear relations
df_multi_quanti.corr()
```

	rain	temperature
rain	1.000000	-0.826323
temperature	-0.826323	1.000000

- Quantitative variables with linear relation
  - Pairplot: Histograms and scatter plots
  - Very useful for $+3$ variables

	rain	temperature
0	120	15
1	120	17
2	80	20
3	0	20
4	50	19
5	100	17
6	0	20
7	0	20
8	0	23
9	150	17

```
sns.pairplot(df_multi_quanti)
✓ 0.2s
<seaborn.axisgrid.PairGrid at 0x1445b10d0>
```



- Quantitative variables with linear relation
  - Groupby
  - Aggregation functions: mean, max, std, etc.
  - Groups: type of clients, of investments, etc.
  - Less information, more lisibility

	rain	temperature
0	120	15
1	120	17
2	80	20
3	0	20
4	50	19
5	100	17
6	0	20
7	0	20
8	0	23
9	150	17

```python
groups = df_multi_quanti.groupby('rain').mean()
groups
```

✓ 0.2s

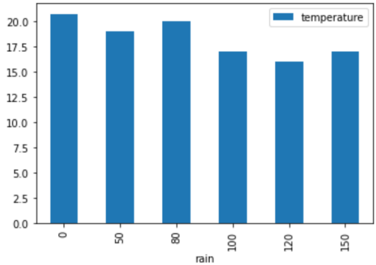
rain	temperature
0	20.75
50	19.00
80	20.00
100	17.00
120	16.00
150	17.00

Multivariate Analysis

- Quantitative variables with linear relation
  - Bar plot
  - Here based on grouped data
  - Complementary to raw data bar plot

	rain	temperature
0	120	15
1	120	17
2	80	20
3	0	20
4	50	19
5	100	17
6	0	20
7	0	20
8	0	23
9	150	17

```
groups.plot.bar()
✓ 0.7s
<AxesSubplot:xlabel='rain'>
```

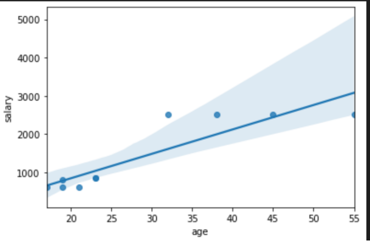


Multivariate Analysis

- Quantitative variables with non-linear relation
  - Scatter plot
  - Linear regression misleading

	age	salary
0	17	600
1	19	600
2	21	600
3	45	2500
4	38	2500
5	55	2500
6	19	800
7	32	2500
8	23	850
9	23	850

```
sns.regplot(x='age', y='salary', data=df_multi_quanti_non_linear)
✓ 0.1s
<AxesSubplot:xlabel='age', ylabel='salary'>
```



- Quantitative variables with non-linear relation
  - Correlation matrix
  - Pearson correlation (linear) also misleading

	age	salary
0	17	600
1	19	600
2	21	600
3	45	2500
4	38	2500
5	55	2500
6	19	800
7	32	2500
8	23	850
9	23	850

```
df_multi_quanti_non_linear.corr()
```

✓ 0.2s

	age	salary
age	1.000000	0.891305
salary	0.891305	1.000000

Multivariate Analysis

- Quantitative variables with non-linear relation
  - Correlation matrix
  - Spearman correlation (non-linear) instead
  - Rank based correlation

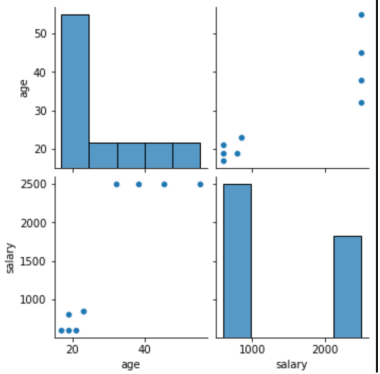
	age	salary
0	17	600
1	19	600
2	21	600
3	45	2500
4	38	2500
5	55	2500
6	19	800
7	32	2500
8	23	850
9	23	850

```
from scipy import stats
stats.spearmanr(df_multi_quanti_non_linear)
✓ 0.2s
SpearmanrResult(correlation=0.9209224503231543, pvalue=0.0001553715621233412)
```

- Quantitative variables with non-linear relation
  - Pairplot
  - Different regims well separated?

	age	salary
0	17	600
1	19	600
2	21	600
3	45	2500
4	38	2500
5	55	2500
6	19	800
7	32	2500
8	23	850
9	23	850

```
sns.pairplot(df_multi_quanti_non_linear)
✓ 0.2s
<seaborn.axisgrid.PairGrid at 0x1452e4f10>
```



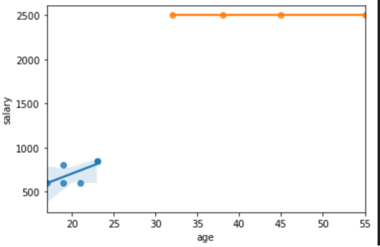
Multivariate Analysis

- Quantitative variables with non-linear relation
  - Analyse regimes relations separatly

	age	salary
0	17	600
1	19	600
2	21	600
3	45	2500
4	38	2500
5	55	2500
6	19	800
7	32	2500
8	23	850
9	23	850

```
sorted = df_multi_quanti_non_linear.sort_values(by='age')
sns.regplot(x='age', y='salary', data=sorted.iloc[:6])
sns.regplot(x='age', y='salary', data=sorted.iloc[6:])
✓ 0.1s

<AxesSubplot:xlabel='age', ylabel='salary'>
```



Multivariate Analysis

- Quantitative variables with non-linear relation
  - Analyse regimes correlations separatly

	age	salary
0	17	600
1	19	600
2	21	600
3	45	2500
4	38	2500
5	55	2500
6	19	800
7	32	2500
8	23	850
9	23	850

```
sorted.iloc[:6].corr()
```

✓ 0.4s

	age	salary
age	1.000000	0.682242
salary	0.682242	1.000000

```
sorted.iloc[6:].corr()
```

✓ 0.3s

	age	salary
age	1.0	NaN
salary	NaN	NaN

- Qualitative variables
  - Counts
  - Proportions on pie chart

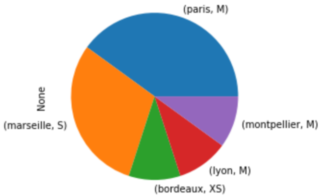
```
# Counts are useful to obtain a
# quantitative variable
df_multi_quali.value_counts()
```
✓ 0.4s

```
city         sweat_size
paris        M             4
marseille    S             3
bordeaux     XS            1
lyon         M             1
montpellier  M             1
dtype: int64
```

	city	sweat_size
0	paris	M
1	marseille	S
2	lyon	M
3	marseille	S
4	paris	M
5	marseille	S
6	bordeaux	XS
7	montpellier	M
8	paris	M
9	paris	M

```
df_multi_quali.value_counts().plot.pie()
```
✓ 0.6s

```
<AxesSubplot:ylabel='None'>
```



Multivariate Analysis

- Qualitative variables
  - Vizualize modalities separately

	city	sweat_size
0	paris	M
1	marseille	S
2	lyon	M
3	marseille	S
4	paris	M
5	marseille	S
6	bordeaux	XS
7	montpellier	M
8	paris	M
9	paris	M

```
df_multi_quali.groupby('city').count()
```

✓ 0.3s

city	sweat_size
bordeaux	1
lyon	1
marseille	3
montpellier	1
paris	4

```
df_multi_quali.groupby('sweat_size').count()
```

✓ 0.2s

sweat_size	city
M	6
S	3
XS	1

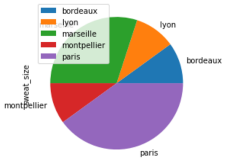
- Qualitative variables
  - Counts on group by

	city	sweat_size
0	paris	M
1	marseille	S
2	lyon	M
3	marseille	S
4	paris	M
5	marseille	S
6	bordeaux	XS
7	montpellier	M
8	paris	M
9	paris	M

```
df_multi_quali.groupby('city').count().plot.pie(y='sweat_size')
```

✓ 0.8s

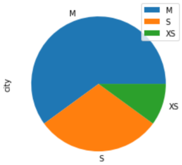
<AxesSubplot:ylabel='sweat_size'>




```
df_multi_quali.groupby('sweat_size').count().plot.pie(y='city')
```

✓ 0.7s

<AxesSubplot:ylabel='city'>



- Qualitative variables
  - Counts on group by

	city	sweat_size
0	paris	M
1	marseille	S
2	lyon	M
3	marseille	S
4	paris	M
5	marseille	S
6	bordeaux	XS
7	montpellier	M
8	paris	M
9	paris	M

```
# test correlation: is there a link between the sweat size and the city?
from scipy.stats import chi2_contingency
crosstab = pd.crosstab(df_multi_quali['city'], df_multi_quali['sweat_size'])
crosstab
```

✓ 0.4s

sweat_size	M	S	XS
city			
bordeaux	0	0	1
lyon	1	0	0
marseille	0	3	0
montpellier	1	0	0
paris	4	0	0

```
results = chi2_contingency(crosstab)
print('pvalue', results[1])
```

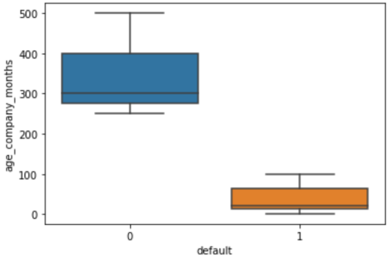
✓ 0.2s

```
pvalue 0.010336050675925726
```

- Quantitative with qualitative variable
  - Boxplot
  - Quality of separation between modalities

	age_company_months	default
0	13	1
1	100	1
2	90	1
3	300	0
4	12	1
5	20	1
6	500	0
7	40	1
8	1	1
9	250	0

```
sns.boxplot(x='default', y='age_company_months',
            data=df_multi_quanti_quali)
✓ 0.6s
<AxesSubplot:xlabel='default', ylabel='age_company_mon
```



- Quantitative with qualitative variable
  - Statistical test ANOVA
  - Correlation

	city	sweat_size
0	paris	M
1	marseille	S
2	lyon	M
3	marseille	S
4	paris	M
5	marseille	S
6	bordeaux	XS
7	montpellier	M
8	paris	M
9	paris	M

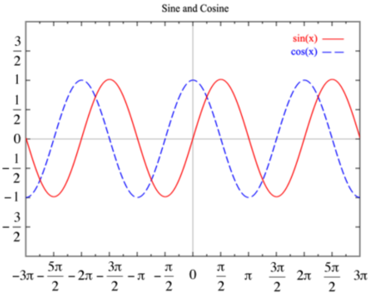
```
# Verify this correlation using ANOVA test
from statsmodels.formula.api import ols
from statsmodels.api import stats
model = ols('default ~ age_company_months', data=df_multi_quanti_quali).fit()
anova_table = stats.anova_lm(model)
anova_table
```

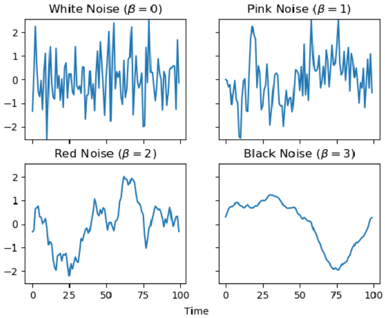
✓ 0.3s

	df	sum_sq	mean_sq	F	PR(>F)
age_company_months	1.0	0.715502	0.715502	4.134365	0.076463
Residual	8.0	1.384498	0.173062	NaN	NaN

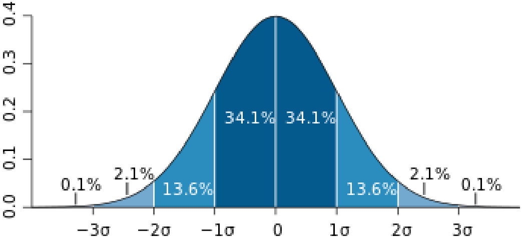
Correlation

- Move repetitively in conjunction
- Methods
  - Pearson
  - Spearman (Rank)
- Spurious correlation (ice cream, Eiffel Tower)

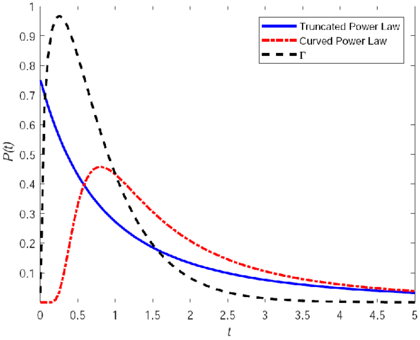




- Normal Law / Gauss Curve
  - Totally resumed by mean and variance
  - Constant mean (0 if centered) and variance (1 if reduced)
  - Uncorrelated individuals
  - Symetric (Skewness=0)
  - Precise bell shape (Kurtosis=3)



- Power Laws: multiplicative growth
- Examples:
  - Normal: human age, size, weight, grades
  - Power: lakes size, wealth



Statistical Tests

- Intention: prevent sampling error
- Hypothesis (Normal Law)
- Examples:
  - Normality test
  - ANOVA
  - Pearson's r
  - Chi square