Introduction to Python

for Data Science

Alexis Bogroff

July 13, 2022

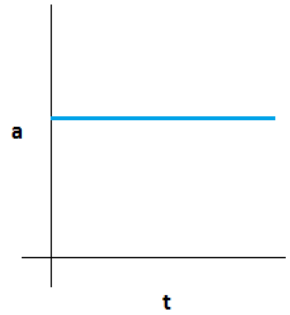
Presenter

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

Data Cleaning

- Select variables (features) drop others
  - Drop constant features



	name	age	time_since_birth	group	country	europe	salary	patrimony	weight	size	comment	voted
0	ju	25.0	25.0	2.0	fr	True	1800.0	3000.0	70.0	175.0	NaN	True
1	ma	36.0	36.0	2.0	gb	True	3000.0	7000.0	NaN	180.0	NaN	NaN
2	lo	18.0	18.0	1.0	gb	True	1500.0	2000.0	NaN	150.0	NaN	True
3	fi	18.0	18.0	1.0	fr	True	1500.0	60000.0	NaN	NaN	NaN	NaN
4	xa	25.0	25.0	2.0	fr	True	1800.0	8000000.0	NaN	NaN	NaN	True
5	pa	38.0	38.0	2.0	es	True	7500.0	6000.0	NaN	170.0	NaN	NaN
6	pe	40.0	40.0	2.0	it	True	8000.0	4000.0	90.0	170.0	NaN	True
7	pe	40.0	40.0	2.0	it	True	8000.0	4000.0	90.0	170.0	NaN	True
8	jb	18.0	18.0	NaN	gb	True	NaN	NaN	60.0	NaN	NaN	NaN
9	mp	40.0	40.0	NaN	fr	True	NaN	NaN	NaN	NaN	NaN	NaN
10	ka	NaN	NaN	NaN	es	True	NaN	NaN	NaN	NaN	NaN	True
11	te	180.0	NaN	1.0	gb	True	4000.0	4000.0	NaN	150.0	NaN	True
12	ko	60.0	60.0	2.0	it	True	20000.0	7000.0	NaN	180.0	NaN	True

Data Cleaning

- Select variables (features) drop others
  - Drop constant features

```
# Detect columns with a constant value
const_col = df.nunique(dropna=False) == 1
const_col
```

✓ 0.5s

```
name                False
age                 False
time_since_birth    False
group               False
country             False
europe               True
salary              False
patrimony           False
weight              False
size                False
comment              True
voted               False
dtype: bool
```

```
# Drop these constant columns
# To drop all:
#df = df.T[~const_col].T
df = df.drop(columns='europe')
df
```

✓ 0.4s

	name	age	time_since_birth	group	country	salary	patrimony	weight	size	comment	voted
0	ju	25.0	25.0	2.0	fr	1800.0	3000.0	70.0	175.0	NaN	True
1	ma	36.0	36.0	2.0	gb	3000.0	7000.0	NaN	180.0	NaN	NaN
2	lo	18.0	18.0	1.0	gb	1500.0	2000.0	NaN	150.0	NaN	True
3	fi	18.0	18.0	1.0	fr	1500.0	60000.0	NaN	NaN	NaN	NaN
4	xa	25.0	25.0	2.0	fr	1800.0	8000000.0	NaN	NaN	NaN	True
5	pa	38.0	38.0	2.0	es	7500.0	6000.0	NaN	170.0	NaN	NaN
6	pe	40.0	40.0	2.0	it	8000.0	4000.0	90.0	170.0	NaN	True
7	pe	40.0	40.0	2.0	it	8000.0	4000.0	90.0	170.0	NaN	True
8	jb	18.0	18.0	NaN	gb	NaN	NaN	60.0	NaN	NaN	NaN
9	mp	40.0	40.0	NaN	fr	NaN	NaN	NaN	NaN	NaN	NaN
10	ka	NaN	NaN	NaN	es	NaN	NaN	NaN	NaN	NaN	True
11	te	180.0	NaN	1.0	gb	4000.0	4000.0	NaN	150.0	NaN	True
12	ko	60.0	60.0	2.0	it	20000.0	7000.0	NaN	180.0	NaN	True

Data Cleaning

- Select variables (features) drop others
  - Duplicated columns

```
# See which column values are duplicated
dup_cols = df.T.duplicated()
dup_cols
```

✓ 0.5s

```
name                False
age                 False
time_since_birth    False
group               False
country             False
salary              False
patrimony           False
weight              False
size                False
voted               False
dtype: bool
```

```
# Drop these duplicated columns by transposing the cleaned DataFrame
df = df.T.drop_duplicates().T
df
```

✓ 0.1s

	name	age	time_since_birth	group	country	salary	patrimony	weight	size	voted
0	ju	25.0	25.0	2.0	fr	1800.0	3000.0	70.0	175.0	True
1	ma	36.0	36.0	2.0	gb	3000.0	7000.0	NaN	180.0	NaN
2	lo	18.0	18.0	1.0	gb	1500.0	2000.0	NaN	150.0	True
3	fi	18.0	18.0	1.0	fr	1500.0	60000.0	NaN	NaN	NaN
4	xa	25.0	25.0	2.0	fr	1800.0	8000000.0	NaN	NaN	True
5	pa	38.0	38.0	2.0	es	7500.0	6000.0	NaN	170.0	NaN
6	pe	40.0	40.0	2.0	it	8000.0	4000.0	90.0	170.0	True
7	pe	40.0	40.0	2.0	it	8000.0	4000.0	90.0	170.0	True
8	jb	18.0	18.0	NaN	gb	NaN	NaN	60.0	NaN	NaN
9	mp	40.0	40.0	NaN	fr	NaN	NaN	NaN	NaN	NaN
10	ka	NaN	NaN	NaN	es	NaN	NaN	NaN	NaN	True
11	te	180.0	NaN	1.0	gb	4000.0	4000.0	NaN	150.0	True
12	ko	60.0	60.0	2.0	it	20000.0	7000.0	NaN	180.0	True

Data Cleaning

- Select variables (features) drop others
  - Drop columns full missing values (NA)

```
# See which columns have only NA values
df.isna().all()
```

✓ 0.4s

```
name                False
age                 False
time_since_birth    False
group               False
country             False
salary              False
patrimony           False
weight              False
size                False
comment              True
voted               False
dtype: bool
```

```
# Drop columns with only NA values
df.dropna(axis=1, how='all', inplace=True)
df
```

✓ 0.5s

	name	age	time_since_birth	group	country	salary	patrimony	weight	size	voted
0	ju	25.0	25.0	2.0	fr	1800.0	3000.0	70.0	175.0	True
1	ma	36.0	36.0	2.0	gb	3000.0	7000.0	NaN	180.0	NaN
2	lo	18.0	18.0	1.0	gb	1500.0	2000.0	NaN	150.0	True
3	fi	18.0	18.0	1.0	fr	1500.0	60000.0	NaN	NaN	NaN
4	xa	25.0	25.0	2.0	fr	1800.0	8000000.0	NaN	NaN	True
5	pa	38.0	38.0	2.0	es	7500.0	6000.0	NaN	170.0	NaN
6	pe	40.0	40.0	2.0	it	8000.0	4000.0	90.0	170.0	True
7	pe	40.0	40.0	2.0	it	8000.0	4000.0	90.0	170.0	True
8	jb	18.0	18.0	NaN	gb	NaN	NaN	60.0	NaN	NaN
9	mp	40.0	40.0	NaN	fr	NaN	NaN	NaN	NaN	NaN
10	ka	NaN	NaN	NaN	es	NaN	NaN	NaN	NaN	True
11	te	180.0	NaN	1.0	gb	4000.0	4000.0	NaN	150.0	True
12	ko	60.0	60.0	2.0	it	20000.0	7000.0	NaN	180.0	True

Data Cleaning

- Select variables (features) drop others
  - Excessive NA proportion

```
df.isna().mean()
```

✓ 0.6s

name	0.000000
age	0.076923
time_since_birth	0.153846
group	0.230769
country	0.000000
salary	0.230769
patrimony	0.230769
weight	0.692308
size	0.384615
voted	0.384615

dtype: float64

```
# Create a mask for columns with more than 50% of NAs
excessive_na = df.isna().mean() > .5
excessive_na
```

✓ 0.6s

name	False
age	False
time_since_birth	False
group	False
country	False
salary	Success
patrimony	False
weight	True
size	False
voted	False

dtype: bool

Data Cleaning

- Select variables (features) drop others
  - Excessive NA proportion

```
# Drop these columns
df = df.T[~excessive_na].T
df
```

✓ 0.1s

	name	age	time_since_birth	group	country	salary	patrimony	size	voted
0	ju	25.0	25.0	2.0	fr	1800.0	3000.0	175.0	True
1	ma	36.0	36.0	2.0	gb	3000.0	7000.0	180.0	NaN
2	lo	18.0	18.0	1.0	gb	1500.0	2000.0	150.0	True
3	fi	18.0	18.0	1.0	fr	1500.0	60000.0	NaN	NaN
4	xa	25.0	25.0	2.0	fr	1800.0	8000000.0	NaN	True
5	pa	38.0	38.0	2.0	es	7500.0	6000.0	170.0	NaN
6	pe	40.0	40.0	2.0	it	8000.0	4000.0	170.0	True
7	pe	40.0	40.0	2.0	it	8000.0	4000.0	170.0	True
8	jb	18.0	18.0	NaN	gb	NaN	NaN	NaN	NaN
9	mp	40.0	40.0	NaN	fr	NaN	NaN	NaN	NaN
10	ka	NaN	NaN	NaN	es	NaN	NaN	NaN	True
11	te	180.0	NaN	1.0	gb	4000.0	4000.0	150.0	True
12	ko	60.0	60.0	2.0	it	20000.0	7000.0	180.0	True

Data Cleaning

- Select variables (features) drop others
  - Excessive correlation between features

```
df_no_missing_values = df.fillna(method='bfill')   # Don't replace NA like this
correlation_matrix = df_no_missing_values.corr()
corr_matrix_clean = (correlation_matrix*100).round(2)
corr_matrix_clean
```

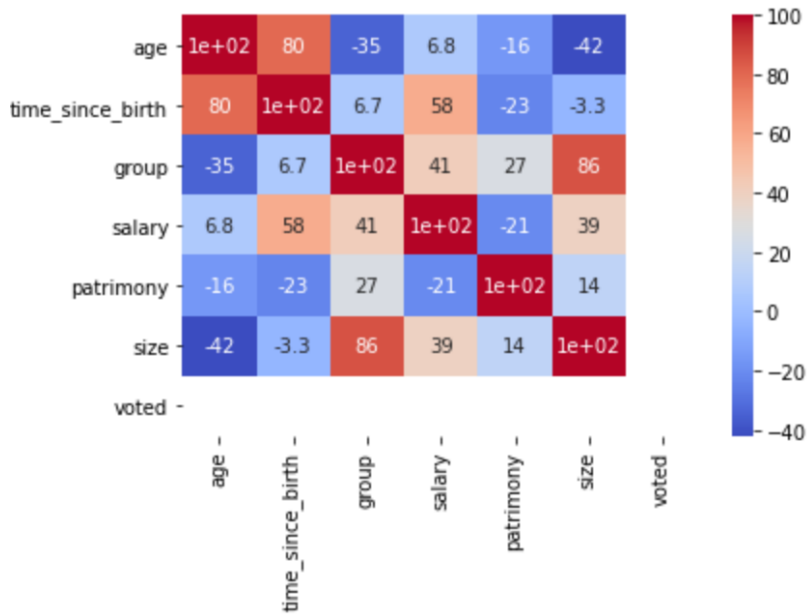
✓ 0.9s

	age	time_since_birth	group	salary	patrimony	size	voted
age	100.00	79.84	-34.76	6.83	-16.18	-42.04	NaN
time_since_birth	79.84	100.00	6.73	57.83	-22.65	-3.34	NaN
group	-34.76	6.73	100.00	41.35	26.56	85.92	NaN
salary	6.83	57.83	41.35	100.00	-21.23	39.17	NaN
patrimony	-16.18	-22.65	26.56	-21.23	100.00	14.33	NaN
size	-42.04	-3.34	85.92	39.17	14.33	100.00	NaN
voted	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Data Cleaning

- Select variables (features) drop others
  - Excessive correlation between features

```
sns.heatmap(corr_matrix_clean, cmap='coolwarm', annot=True)
```



Data Cleaning

- Drop poor rows
  - Drop duplicated rows

```
# See which row values are duplicated
df.duplicated()
```

✓ 0.4s

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7      True
8     False
9     False
10    False
11    False
12    False
dtype: bool
```

```
# Drop these duplicated rows
df.drop_duplicates(inplace=True)
df
```

✓ 0.6s

	name	age	time_since_birth	group	country	salary	patrimony	size	voted
0	ju	25.0	25.0	2.0	fr	1800.0	3000.0	175.0	True
1	ma	36.0	36.0	2.0	gb	3000.0	7000.0	180.0	NaN
2	lo	18.0	18.0	1.0	gb	1500.0	2000.0	150.0	True
3	fi	18.0	18.0	1.0	fr	1500.0	60000.0	NaN	NaN
4	xa	25.0	25.0	2.0	fr	1800.0	8000000.0	NaN	True
5	pa	38.0	38.0	2.0	es	7500.0	6000.0	170.0	NaN
6	pe	40.0	40.0	2.0	it	8000.0	4000.0	170.0	True
8	jb	18.0	18.0	NaN	gb	NaN	NaN	NaN	NaN
9	mp	40.0	40.0	NaN	fr	NaN	NaN	NaN	NaN
10	ka	NaN	NaN	NaN	es	NaN	NaN	NaN	True
11	te	180.0	NaN	1.0	gb	4000.0	4000.0	150.0	True
12	ko	60.0	60.0	2.0	it	20000.0	7000.0	180.0	True

Data Cleaning

- Drop poor rows
  - Drop rows with excessing NA proportion

```
df.T.isna().mean()
```

✓ 0.6s

```
0     0.000000
1     0.111111
2     0.000000
3     0.222222
4     0.111111
5     0.111111
6     0.000000
8     0.555556
9     0.555556
10    0.666667
11    0.111111
12    0.000000
dtype: float64
```

```
# Create a mask for rows with more than 50% of NAs
excessive_na = df.T.isna().mean() > .5
excessive_na
```

✓ 0.5s

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
8      True
9      True
10     True
11    False
12    False
dtype: bool
```

Data Cleaning

- Drop poor rows
  - Drop rows with excessing NA proportion

```
# Drop these columns
df = df[~excessive_na]
df
```

✓ 0.6s

	name	age	time_since_birth	group	country	salary	patrimony	size	voted
0	ju	25.0	25.0	2.0	fr	1800.0	3000.0	175.0	True
1	ma	36.0	36.0	2.0	gb	3000.0	7000.0	180.0	NaN
2	lo	18.0	18.0	1.0	gb	1500.0	2000.0	150.0	True
3	fi	18.0	18.0	1.0	fr	1500.0	60000.0	NaN	NaN
4	xa	25.0	25.0	2.0	fr	1800.0	8000000.0	NaN	True
5	pa	38.0	38.0	2.0	es	7500.0	6000.0	170.0	NaN
6	pe	40.0	40.0	2.0	it	8000.0	4000.0	170.0	True
11	te	180.0	NaN	1.0	gb	4000.0	4000.0	150.0	True
12	ko	60.0	60.0	2.0	it	20000.0	7000.0	180.0	True

Data Cleaning

- Impute NAs (NaNs, missing values)
  - Missing can be the information

```
rows_with_na = df.isna().any(axis=1)
df[rows_with_na]
```

✓ 0.6s

	name	age	time_since_birth	group	country	salary	patrimony	size	voted
1	ma	36.0	36.0	2.0	gb	3000.0	7000.0	180.0	NaN
3	fi	18.0	18.0	1.0	fr	1500.0	60000.0	NaN	NaN
4	xa	25.0	25.0	2.0	fr	1800.0	8000000.0	NaN	True
5	pa	38.0	38.0	2.0	es	7500.0	6000.0	170.0	NaN
11	te	180.0	NaN	1.0	gb	4000.0	4000.0	150.0	True

```
df['voted'].value_counts()
```

✓ 0.6s

```
True    6
Name: voted, dtype: int64
```

```
# Replace Nan by False (in 'voted' variable only)
df.loc[df['voted'].isna(), 'voted'] = False
```

```
# Check result of NA imputation
df['voted'].value_counts()
```

✓ 0.6s

```
True     6
False    3
Name: voted, dtype: int64
```

Data Cleaning

- Impute NAs (NaNs, missing values)
  - Standard methods
  - Replace by a constant value (mean, median)

```
remaining_na_rows = df.isna().any(axis=1)
df[remaining_na_rows]
```

✓ 0.6s

	name	age	time_since_birth	group	country	salary	patrimony	size	voted
3	fi	18.0	18.0	1.0	fr	1500.0	60000.0	NaN	False
4	xa	25.0	25.0	2.0	fr	1800.0	8000000.0	NaN	True
11	te	180.0	NaN	1.0	gb	4000.0	4000.0	150.0	True

```
size_mean = df['size'].mean()
edian = df['size'].median()
print(size_mean)
print(size_median)
```

Success

✓ 0.6s

```
167.85714285714286
170.0
```

```
size_mean = df['size'].mean()
size_median = df['size'].median()
print(size_mean)
print(size_median)
```

✓ 0.6s

```
167.85714285714286
170.0
```

```
# Replace Nan by the mean
df['size'].fillna(size_mean)
```

✓ 0.5s

```
0     175.000000
1     180.000000
2     150.000000
3     167.857143
4     167.857143
5     170.000000
6     170.000000
11    150.000000
12    180.000000
Name: size, dtype: float64
```

Data Cleaning

- Impute NAs (NaNs, missing values)
  - Standard methods
  - Replace by a the next or previous row value (ffil, bfill)

```
remaining_na_rows = df.isna().any(axis=1)
df[remaining_na_rows]
```

✓ 0.6s

	name	age	time_since_birth	group	country	salary	patrimony	size	voted
3	fi	18.0	18.0	1.0	fr	1500.0	60000.0	NaN	False
4	xa	25.0	25.0	2.0	fr	1800.0	8000000.0	NaN	True
11	te	180.0	NaN	1.0	gb	4000.0	4000.0	150.0	True

```
df['size'].fillna(method='ffill')
```

✓ 0.5s

```
0     175.0
1     180.0
2     150.0
3     150.0
4     150.0
5     170.0
6     170.0
11    150.0
12    180.0
Name: size, dtype: float64
```

```
df['size'].fillna(method='bfill')
```

✓ 0.5s

```
0     175.0
1     180.0
2     150.0
3     170.0
4     170.0
5     170.0
6     170.0
11    150.0
12    180.0
Name: size, dtype: float64
```

Data Cleaning

- Impute NAs (NaNs, missing values)
  - Advanced methods
  - Group values to get precise mean or median

```
df_mean_by_age = df.groupby('age').mean()
df_mean_by_age
```

✓ 0.6s

age	time_since_birth	group	salary	patrimony	size	voted
18.0	18.0	1.0	1500.0	31000.0	150.0	0.5
25.0	25.0	2.0	1800.0	4001500.0	175.0	1.0
36.0	36.0	2.0	3000.0	7000.0	180.0	0.0
38.0	38.0	2.0	7500.0	6000.0	170.0	0.0
40.0	40.0	2.0	8000.0	4000.0	170.0	1.0
60.0	60.0	2.0	20000.0	7000.0	180.0	1.0
180.0	NaN	1.0	4000.0	4000.0	150.0	1.0

```
nan_age = df[remaining_na_rows]['age']
nan_size_to_impute = df_mean_by_age.loc[nan_age]['size']
nan_size_to_impute
```

✓ 0.6s

```
age
18.0     150.0
25.0     175.0
180.0    150.0
Name: size, dtype: float64
```

Data Cleaning

- Impute NAs (NaNs, missing values)
  - Advanced methods
  - Group values to get precise mean or median

```
df.fillna(df.groupby('age').transform('mean'))
```

✓ 0.6s

	name	age	time_since_birth	group	country	salary	patrimony	size	voted
0	ju	25.0	25.0	2.0	fr	1800.0	3000.0	175.0	True
1	ma	36.0	36.0	2.0	gb	3000.0	7000.0	180.0	False
2	lo	18.0	18.0	1.0	gb	1500.0	2000.0	150.0	True
3	fi	18.0	18.0	1.0	fr	1500.0	60000.0	150.0	False
4	xa	25.0	25.0	2.0	fr	1800.0	8000000.0	175.0	True
5	pa	38.0	38.0	2.0	es	7500.0	6000.0	170.0	False
6	pe	40.0	40.0	2.0	it	8000.0	4000.0	170.0	True
11	te	180.0	NaN	1.0	gb	4000.0	4000.0	150.0	True
12	ko	60.0	60.0	2.0	it	20000.0	7000.0	180.0	True

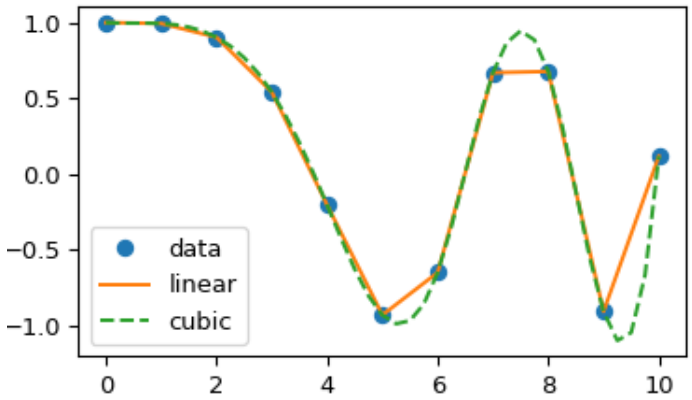
Data Cleaning

- Impute NAs (NaNs, missing values)
  - Advanced methods
  - Interpolate using a model

```
df._get_numeric_data()
```

✓ 0.6s

	age	time_since_birth	group	salary	patrimony	size	voted
0	25.0	25.0	2.0	1800.0	3000.0	175.0	True
1	36.0	36.0	2.0	3000.0	7000.0	180.0	False
2	18.0	18.0	1.0	1500.0	2000.0	150.0	True
3	18.0	18.0	1.0	1500.0	60000.0	NaN	False
4	25.0	25.0	2.0	1800.0	8000000.0	NaN	True
5	38.0	38.0	2.0	7500.0	6000.0	170.0	False
6	40.0	40.0	2.0	8000.0	4000.0	170.0	True
11	180.0	NaN	1.0	4000.0	4000.0	150.0	True
12	60.0	60.0	2.0	20000.0	7000.0	180.0	True



Data Cleaning

- Impute NAs (NaNs, missing values)
  - Advanced methods
  - Interpolate using a model

```
from sklearn.impute import KNNImputer
```

✓ 0.1s

```
imputer = KNNImputer(n_neighbors=2, weights="uniform")
na_prediction = imputer.fit_transform(df._get_numeric_data())
df_filled = pd.DataFrame(na_prediction, columns=df._get_numeric_data().columns)
df_filled
```

✓ 0.6s

	age	time_since_birth	group	salary	patrimony	size	voted
0	25.0	25.0	2.0	1800.0	3000.0	175.0	1.0
1	36.0	36.0	2.0	3000.0	7000.0	180.0	0.0
2	18.0	18.0	1.0	1500.0	2000.0	150.0	1.0
3	18.0	18.0	1.0	1500.0	60000.0	175.0	0.0
4	25.0	25.0	2.0	1800.0	8000000.0	180.0	1.0
5	38.0	38.0	2.0	7500.0	6000.0	170.0	0.0
6	40.0	40.0	2.0	8000.0	4000.0	170.0	1.0
7	180.0	30.5	1.0	4000.0	4000.0	150.0	1.0
8	60.0	60.0	2.0	20000.0	7000.0	180.0	1.0

Data Cleaning

- Outliers
  - Extreme values too keep
  - Abberations to delete
  - Variables to transform

Data Management

- Merge tables
  - Concatenation on rows
  - Merge on unique key column
    - Outer (indicator)
    - Left
    - Right, inner

Feature Engineering

- Quantitative variables (numbers representing quantities): create groups
- Qualitative variables (categories): one-hot encode
- Filter

Why using vizualizations

- Quick understanding simple patterns (trend line plot, groups scatter plot)
- Better intuition on complex patterns (CNN weights maps)
- Reporting

Graphs types

- Univariate Analysis
  - Histograms (distributions)
  - Line plots (Time series)
  - Lorentz Curve (inegalities)
- Multivariate Analysis
  - Scatter plots
  - Heatmaps
    - Correlations
    - Confusion matrices

Libraries

- Matplotlib (.pyplot)
- Seaborn for nice default graphs
- Plotly (Dash) for interactive graphs