



A Jump Start for Beginners

Content

1	Installation	2
2	Hello World	4
3	First Steps with Data	5
3.1	Code Description	5
3.2	Loading Data	5
3.3	Exercises	6
4	The profK_statistics.py Module	12
4.1	Class describe	12
4.2	Class dataprep	15
4.3	Class plots	18
4.4	Class tests	23
4.5	Class regression	30
4.6	Class outlier	31
5	Working with Data	33
5.1	Data Structures	33
5.2	Data Frames	35
5.3	Data Visualization	38
5.4	Programing Functions	38

1 Installation

Step 1.

Download Anaconda.



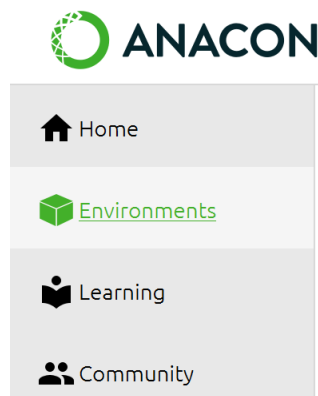
Download link:

- Anaconda:

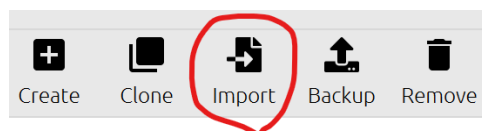
<https://www.anaconda.com/products/individual>

Step 2.

- Go to “environments”

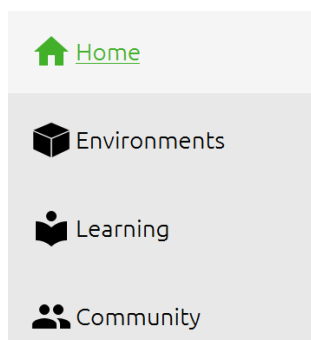


- Now import the file *prof_kauf_environment.yaml*

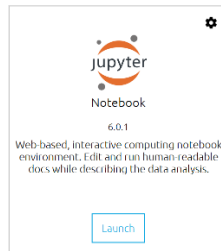


Step 3.

- Go back to “home”



- Open Anaconda and launch Jupyter notebook:



Important remarks:

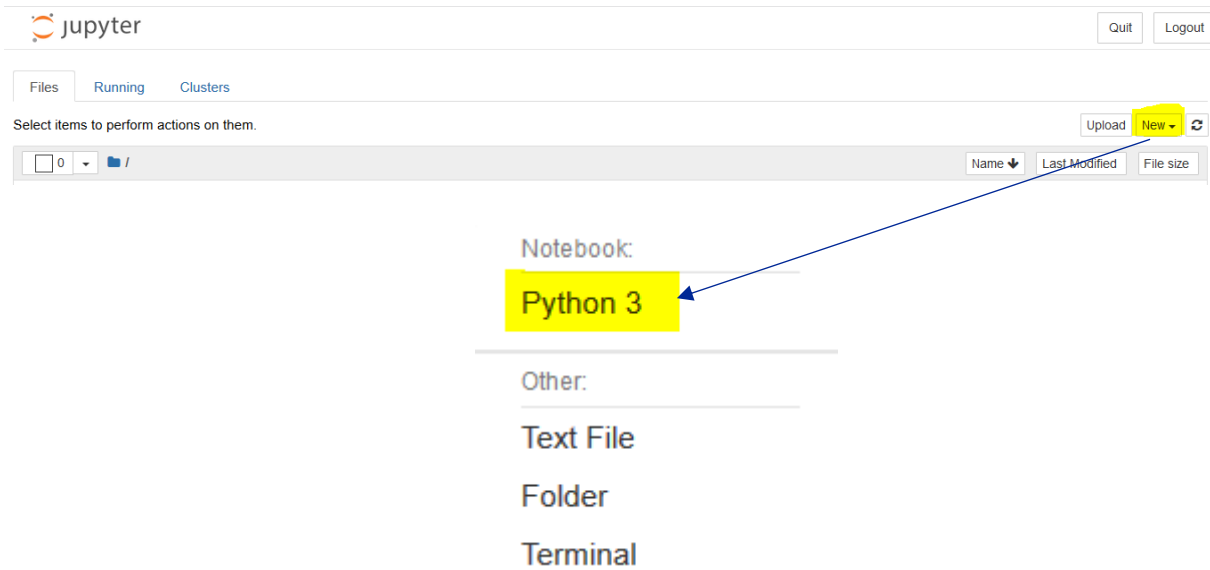
1. Whenever you start a new notebook, you have to have the following first cell:

```
import httpimport
url='https://raw.githubusercontent.com/ProfKauf/Modules/main/'
with httpimport.remote_repo(url):
    import profK_libraries, profK_statistics
from profK_libraries import *
from profK_statistics import *
```

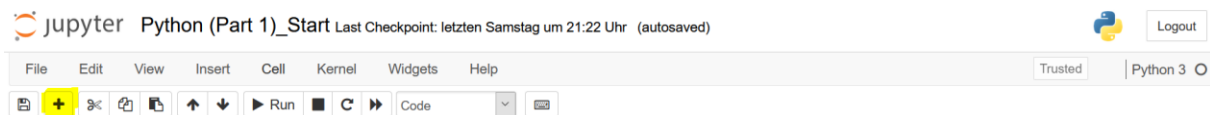
2. Always make sure that you are operating within the prof_kauf_env.

2 Hello World

Step 1. Create a new python notebook:



Step 2. Create a new cell by the "+" button in the toolbar



Step 3. Now, the time has come to say hello to the world:

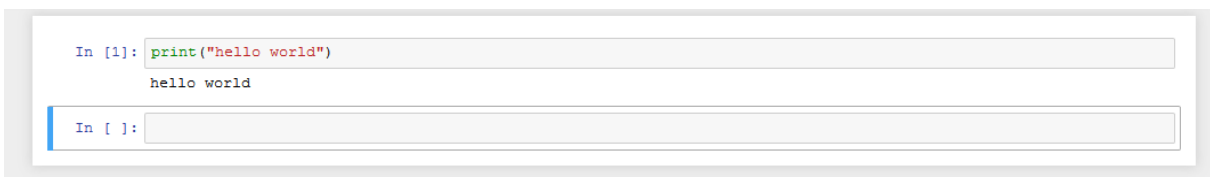
Type

```
print("hello world")
```

and hit



Result:



Important Information!

If you leave ANACONDA and open your notebook sometime later, you need to rerun the entire notebook:



3 First Steps with Data

3.1 Code Description

You are supposed to comment and structure your coding by using markdown cells. In order to switch a coding cell into a comment cell hit “cell”:



Then hit “cell type” and pick “markdown cell”.

There are some styling options, for instance:

- Headlines:
 - Headline 1: `# markdown`
 - Headline 2: `## markdown`
 - ...
- Colored markdown:

```
< div class = "alert alert – block alert – info" >  
< b > markdown colored </b >
```

Try these commands. You should get the following result:

markdown normal

markdown headline 1

markdown colored

Please follow the link below to learn more about the options:

<https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html>

3.2 Loading Data

Data is usually stored in an excel or csv file. In order to work with data in your notebook, you need to load it via the following command:

```
data = pd.read_excel (r'path/to/file/file_name.xlsx')
```

Hint 1. The code above is for files in .xlsx format. If you want to load csv files, replace `_excel` by `_csv`

Hint 2. Before you use a .csv file, check if it is separated by commas. If it is separated by, e.g., semicolons, you need to tell python that, see:

https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

Hint 3. In Anaconda, you may upload your file via



Then, you can omit the path/to/file

3.3 Exercises

Preliminaries

- Open a new notebook
- Copy the preamble in the first cell (see Section 0)
- Load the file YoungPeopleSurvey.xlsx (see previous Section)

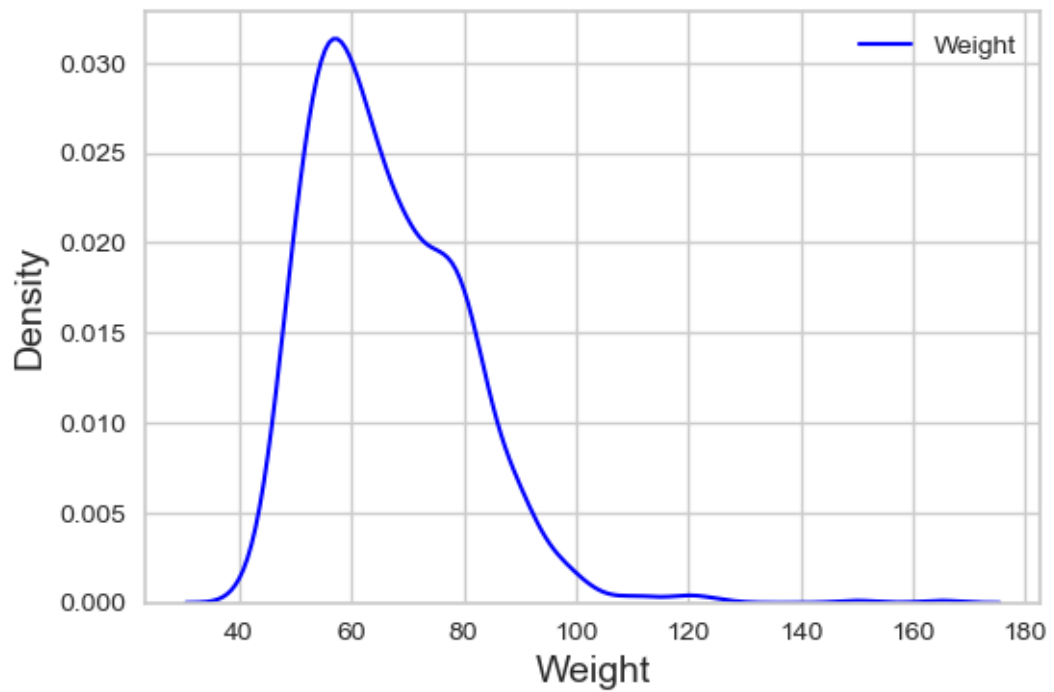
Exercises

The following exercises help you to develop a basic understanding of Python. Notice that the data frame was named data. You may name the data frame differently and adjust the code accordingly.

Exercise 1. Distribution plot.

- Create a new cell. Create a distribution plot of weight:

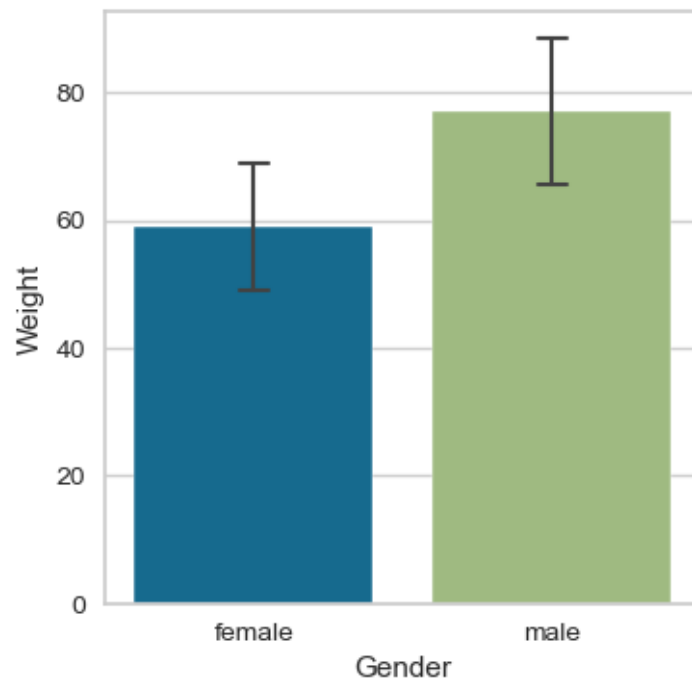
```
plots.dist(data['Weight'],fig=[6,4],labelsize=14,ticksiz=10,legsize=10,linewidth=1.5)
```



Exercise 2. Barplot with error bars that shows the mean weight for male and female.

- Create a new cell. Create the barplot as follows:

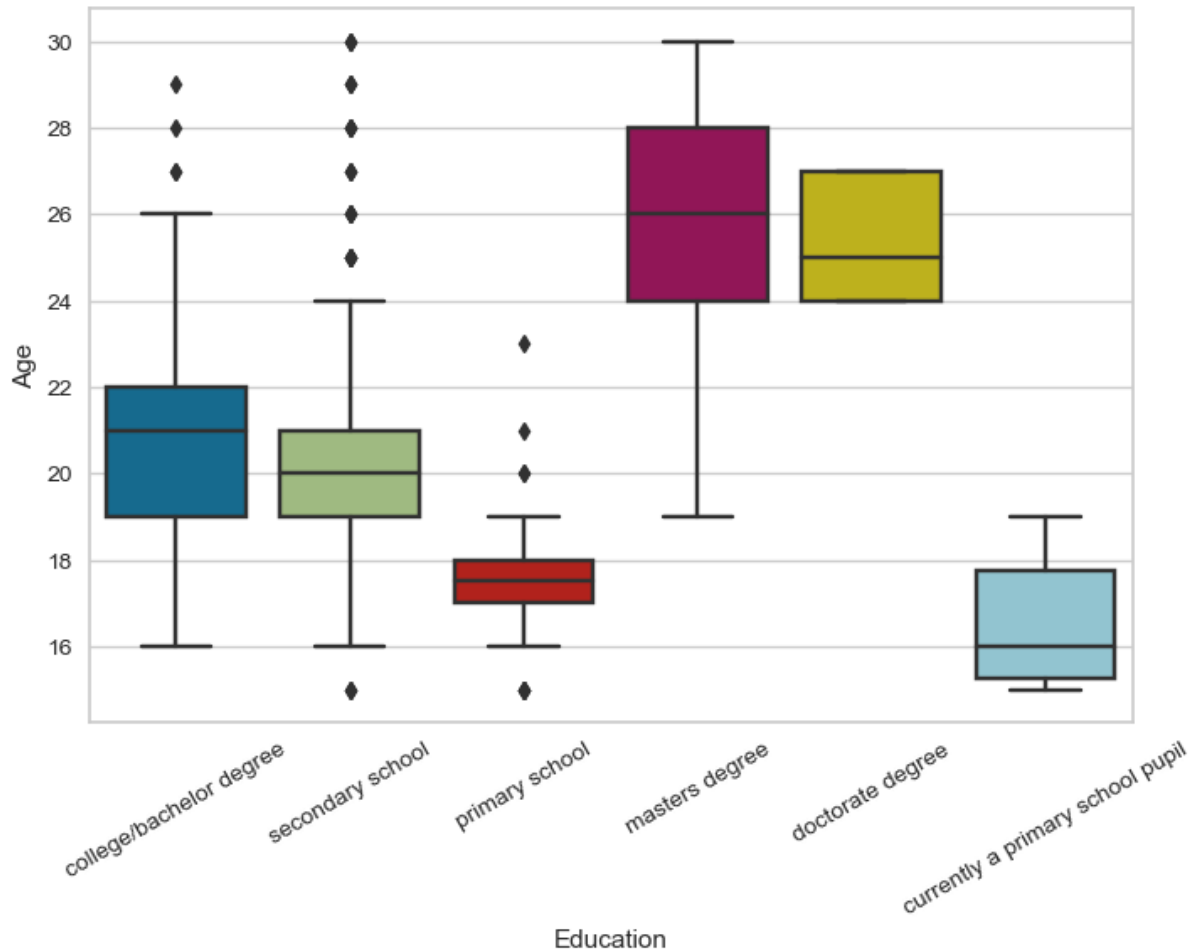
```
plt.figure(figsize=(4,4))
sns.barplot(x='Gender',y='Weight',data=data,ci='sd',capsize=.1,errwidth=1.5
)
```



Exercise 3. Boxplot with quartiles of age for educational levels.

- Create a new cell. Create the boxplot as follows:

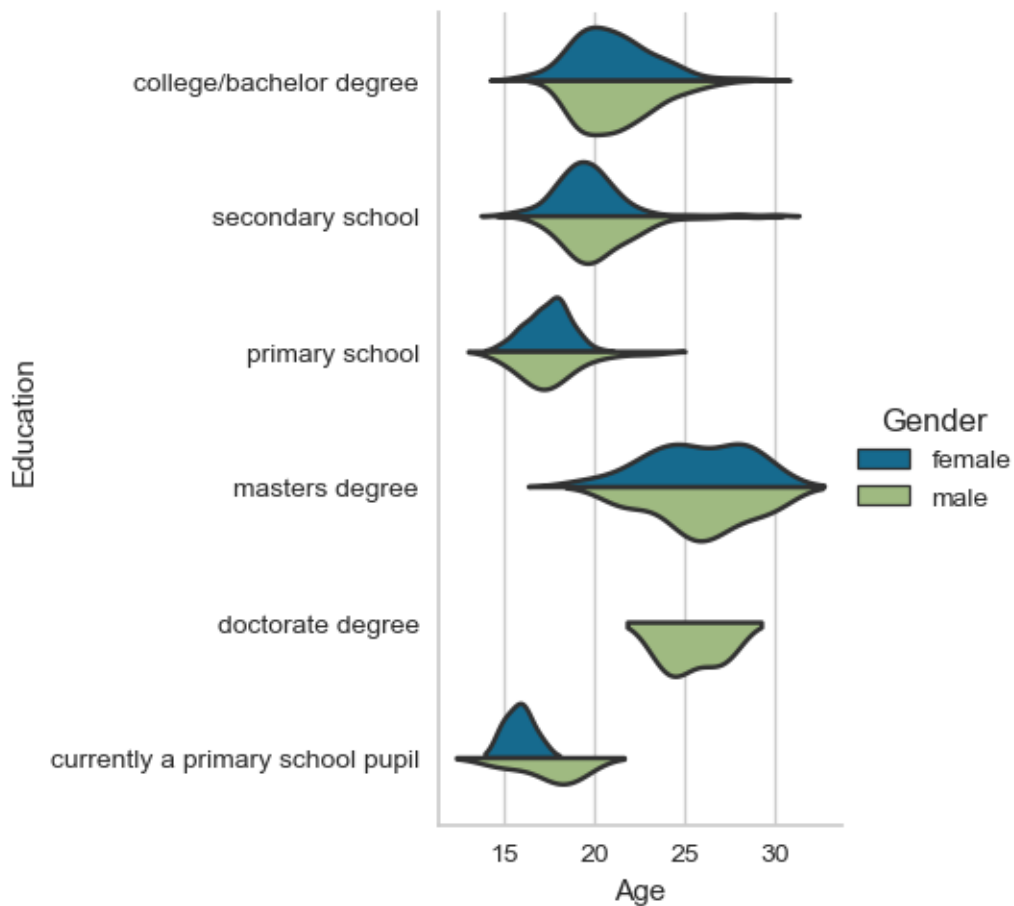
```
sns.boxplot(x='Education',y='Age',data=data)  
plt.xticks(rotation=30)
```



Exercise 4. Violin plot with with age for educational levels and gender.

- Create a new cell. Create the violin plot as follows:

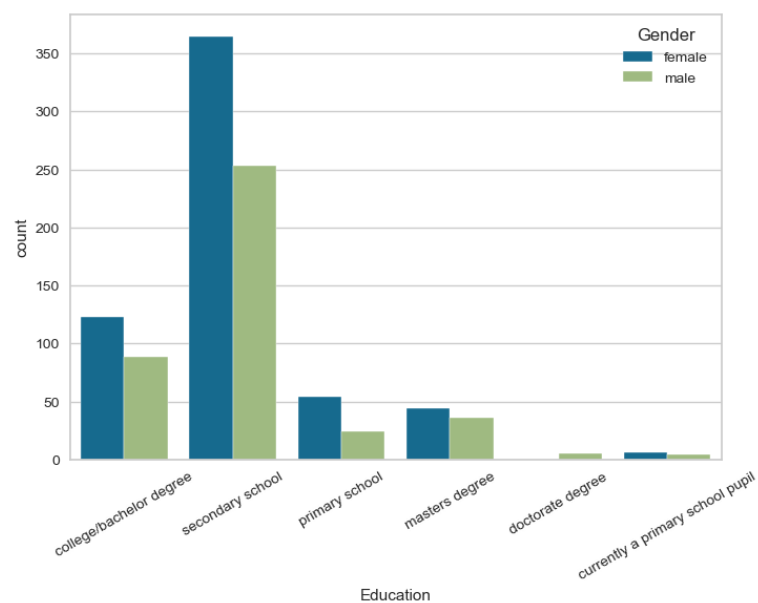
```
sns.catplot(data=data, y="Education", x="Age", hue='Gender',  
kind="violin",inner=None, split=True)
```

Exercise 5. Countplot for educational level and gender.

- Create a new cell. Create the countplot as follows:

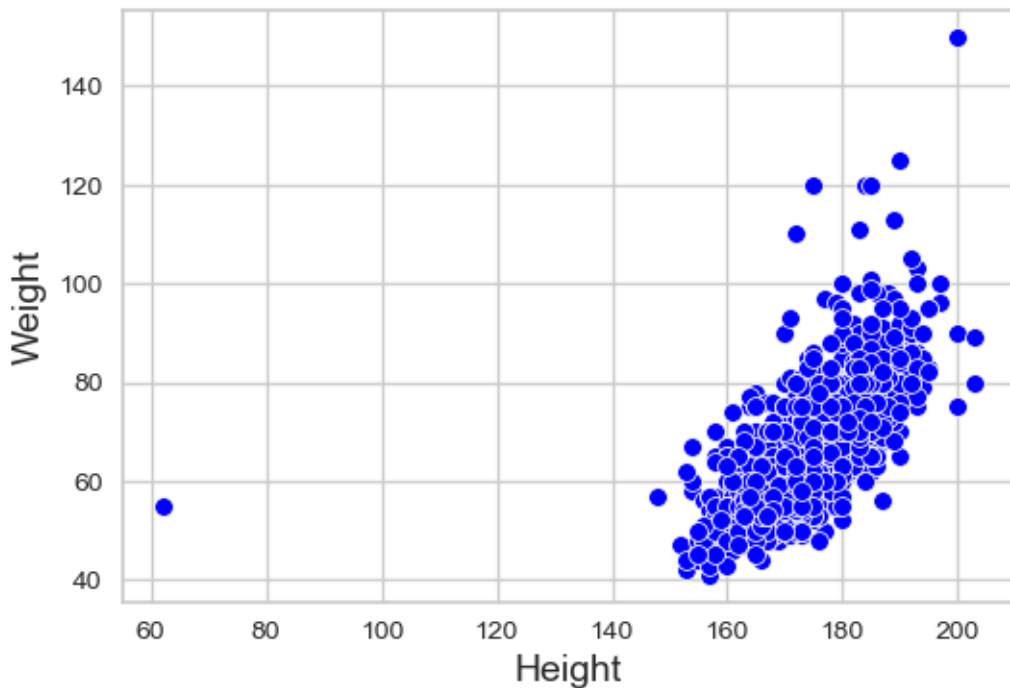
```
sns.countplot(data=data, x='Education', hue='Gender')
plt.xticks(rotation=30)
```



Exercise 6. Scatterplot age vs. height.

- Create a new cell. Create the scatterplot as follows:

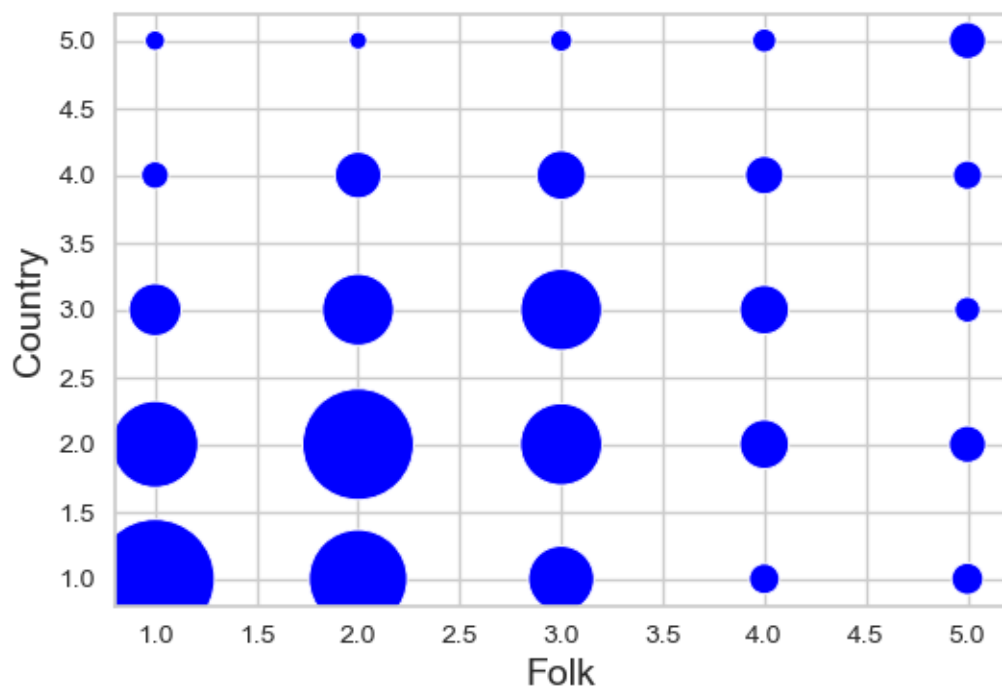
```
plots.scatter(data['Height'],data['Weight'],fig=[6,4],ticksize=10,labelsize=14,dotsize=50)
```



Exercise 7. Bubbleplot Folk vs. Country.

- Create a new cell. Create the bubbleplot as follows:

```
plots.scatter(data['Folk'],data['Country'],fig=[6,4],ticksize=10,labelsize=14,ordinal=True)
```



Exercise 8. Encode Variable Education.

- Have a look at the unique values of the variable:

```
data['Education'].unique()  
array(['college/bachelor degree', 'secondary school', 'primary school',  
      'masters degree', 'doctorate degree',  
      'currently a primary school pupil', nan], dtype=object)
```

- Define encoder where the values are set in ascending order:

```
enc=dataprep.encoder(order={'Education':['currently a primary school  
pupil','primary school','secondary school','college/bachelor  
degree','masters degree','doctorate degree']})
```

- Encode the variable:

```
data_encoded=enc.fit_transform(data)
```

- Have a look at the unique values of the encoded variable:

```
data_encoded['Education'].unique()  
array([ 3.,  2.,  1.,  4.,  5.,  0., nan])
```

Exercise 9. Dropping columns.

- Create a new data set that contains only the variables Education, Age and Gender.

```
data_new=data[['Education','Age','Gender']]
```

Exercise 10. Dropping rows I.

- Create a new data set that contains only people who are over 25:

```
data_over25=data[data['Age']>25]
```

- Have a look at the data shapes:

```
print(data.shape)  
print(data_over25.shape)  
(1010, 150)  
(67, 150)
```

Exercise 11. Dropping rows II.

- Create a new data set that contains only people who hold a masters or doctorate degree.
- Method 1:

```
data_educated=data[(data['Education']=='masters degree') &  
(data['Education']=='doctorate degree')]
```

- Method 2:

```
indices=data[(data['Education'].isin(['masters degree','doctorate degree']))].index
data_educated=data.loc[indices]
```

Exercise 12. Create a new column that contains the body mass index (bmi).

$$bmi = \frac{weight}{height \text{ (in m)}^2}$$

```
data['bmi']=data['Weight']/(data['Height']/100)**2
```

4 The profK_statistics.py Module

In this section, we use the following abbreviations for types of input objects:

Abbreviation	Meaning
var	Variable
str	String
bool	Boolean Expression (True or False)
int	Integer number (1,2,3,4,...)

In the following, a "grouping variable" refers to a variable that groups your data set in categories.

4.1 Class describe

4.1.1 Summary

```
Class describe
-----
.data
.contingency
.corrmat
```

4.1.2 describe.data

Example:

```
description=describe.data(data=data2[['married','age']],nominal=['married'])
description.table(show='nominal')
```

married	
count	28885
mode	yes
categories	2
least freq	no(37.60%)
most freq	yes(62.40%)

Code Structure:

```
arguments
-----
- data = name of dataframe (var)
- ordinal = list of ordinal variables ([str,str,...])
- nominal = list of nominal variables ([str,str,...])
returns
-----
.table(show) -> descriptive statistics (pd.DataFrame)
  arguments
  -----
  - show = statistics for which variables ('numeric','ordinal','nominal') (str)
[default='numeric']
```

4.1.3 describe.contingency

Example:

```
describe.contingency(data2['married'],data2['house_own'],show='observed')
```

	house_own	no	yes
married			
no	5710	5150	
yes	3540	14485	

Code Structure:

```
arguments
-----
- x,y = vector of the variables (pd.Series)
- show = what to show ('observed','expected','deviations') (str)
[default='observed']
- decimals = decimal places for percentage deviations (int)
returns
-----
observed or expected frequencies or their deviation
```

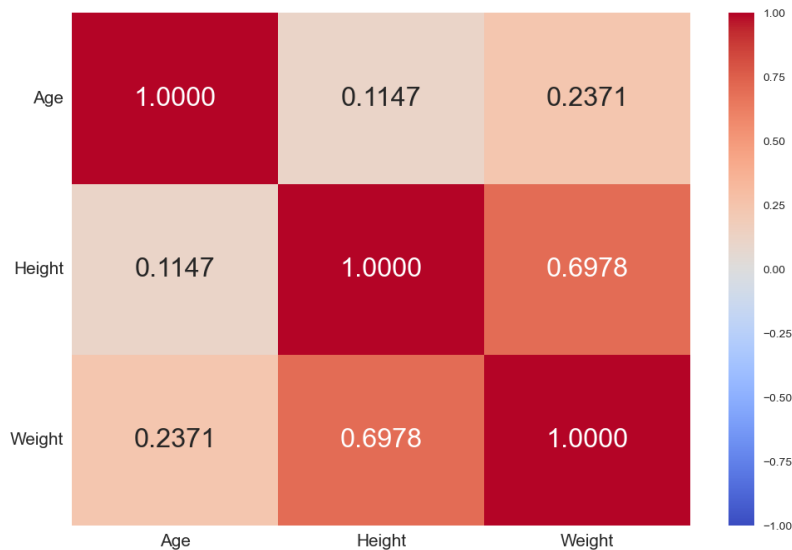
4.1.4 describe.corrmat

Examples:

```
cm=describe.corrmat(data=data_pair2)
cm.table
```

	Age	Height	Weight
Age	1.000000	0.114687	0.237084
Height	0.114687	1.000000	0.697786
Weight	0.237084	0.697786	1.000000

```
cm.heatmap()
```



```
data_pair2=data_pair2.dropna()
cm=describe.corrmat(data=data_pair2,utri=False,stars=True)
```

	Age	Height	Weight
Age		***	****
Height	0.1147***		****
Weight	0.2371****	0.6978****	

Code Structure:

```
arguments
-----
- data = name of dataframe (var)
- nominal,ordinal = list of names of nominal,ordinal variables ([str,str,...])
- ordvsord = correlation coefficient for ordinal vs ordinal/numerical
('spearman','kendall','gk_gamma') (str) [default = 'spearman']
- nomvsnom = correlation coefficient for nominal vs nominal/ordinal ('cramer')
(str)
- numvsnom = correlation coefficient for numerical vs nominal ('eta','pbc') (str)
[default = 'eta']
- stars = do you want to flag significant correlations with stars (bool) [default =
False]
- padjust = method to correct for multiple testing
('none','bonf','sidak','holm','fdr_bh','fdr_by') (str) [default='bonf']
- utri = do you want to show the upper triangle of the matrix (bool) [default =
True]
- ltri = do you want to show the lower triangle of the matrix (bool) [default =
True]
- fill = how to fill the empty spaces if upper/lower triangle is masked (str)
[default = '']
- decimals = how many decimal places to show in case a triangle is masked or stars
is True (int) [default = 4]
- percent = display the correlations as percentages (bool) [default = False]
- force_biserial = always use rank-biserial resp. point-biserial coefficient when
ordinal/numerical vs binary nominal (bool) [default = True]
```

```

returns
-----
.table -> correlation matrix (pd.DataFrame)
.def
heatmap(self, cmap='coolwarm', roty=0, rotx=0, lsize=15, tsize=20, annot=True, fsize=70, fig=[12,8], down=0): -> matrix as heatmap
    arguments
    -----
    - cmap = palette for heatmap (str) [default = 'coolwarm']
    - rotx, roty = rotate x or y labels (int) [default: rotx, roty = 0, 0]
    - lsize = labels size (int) [default = 15]
    - tsize = ticks size (int) [default = 20]
    - annot = show numbers in cells (bool) [default = True]
    - fsize = font size in cells (int) [default = 70]
    - fig = size of figure ([int,int]) [default = [12,8]]
    - down = shift the caption down only relevant when stars = True (dec) [default = 0]

```

4.2 Class dataprep

Examples:

- Separate a data set, e.g., married from not married people:

```

groups=dataprep.group_sep(data=data2[['married','age','income']],groupvar='married')
groups[0].head()

```

	married	age	income
0	no	75	67195.781504
1	no	75	57014.602488
2	no	75	51924.012980
3	no	75	41742.833964
4	no	75	50905.895078

```
groups[1].head()
```

	married	age	income
5	yes	50	38688.480260
6	yes	50	37670.362358
7	yes	50	38688.480260
8	yes	50	38688.480260
9	yes	50	38688.480260

- One-hot-/Dummy-Encoding of data, e.g., ethnicity.

Original data:

	ethnicity	income
0	white	67195.781504
1	white	57014.602488
2	white	51924.012980
3	white	41742.833964
4	white	50905.895078

One-hot-Encoding:

```
enc=dataprep.onehot(cats=['ethnicity'])
data_encoded=enc.fit_transform(data2[['ethnicity','income']])
```

Encoded Data:

	dummy__ethnicity_hispanic	dummy__ethnicity_other	dummy__ethnicity_white	income
0	0.0	0.0	1.0	67195.781504
1	0.0	0.0	1.0	57014.602488
2	0.0	0.0	1.0	51924.012980
3	0.0	0.0	1.0	41742.833964
4	0.0	0.0	1.0	50905.895078

- Standard encoding of an ordinal variable, e.g., happiness:

Original data:

	happy_study_program	income	year	level
0	very unhappy	1100.0	2023	Bachelor
1	happy	650.0	2023	Bachelor
2	unhappy	1000.0	2023	Bachelor
3	so,so	500.0	2023	Bachelor
4	unhappy	1000.0	2023	Bachelor

Standard encoding:

```
enc=dataprep.encoder(order={'happy_study_program':['very
unhappy','unhappy','so,so','happy','very happy']})
data_encoded=enc.fit_transform(data)
```


Encoded data:

	happy_study_program	income	year	level
0	0.0	1100.0	2023	Bachelor
1	3.0	650.0	2023	Bachelor
2	1.0	1000.0	2023	Bachelor
3	2.0	500.0	2023	Bachelor
4	1.0	1000.0	2023	Bachelor

Remark: in case of nominal variables, the order can be omitted.

Code Structure:

```
.group_sep
    arguments
    -----
    - data = name of dataframe (var)
    - groupvar = name of the grouping variable (str)
    returns
    -----
    list of groupwise dataframes (list of pd.DataFrame)
.nan
    arguments
    -----
    - data = name of dataframe (var)
    - cols = list of column names that should be examined ([str,str,...])
    returns
    -----
    .analysis -> summary table for the analysis of nans (pd.DataFrame)
    .drop -> a dataframe where nans are removed (pd.DataFrame)
.onehot
    arguments
    -----
    - drop = string that indicates which dummies should be dropped, can be set to
None (str) [default='first']
    returns
    -----
    .fit(X,y)
    .transform(X,y,sparse) -> a dataframe where nans are removed (pd.DataFrame)
        arguments
        -----
        - X = a dataframe (pd.DataFrame)
        - y = target variable (pd.Series) [optional, default = None]
        - sparse = whether output should be returned in sparse format (bool)
[default=False]
.encoder
    arguments
    -----
    - cols = nominal columns to encoder [default=None]
    - order = a dictionary of the format {colname:order of categories} to encode
ordinal variables [default=None]
    if cols and order are both none the encoder treats all columns with strings as
nominal variables
```

```

returns
-----
.transform(data,sparse) -> an encoded dataframe
arguments
-----
- data = a dataframe (pd.DataFrame) or a vector (pd.Series)
- sparse = whether output should be returned in sparse format (bool)
[default=False]

```

4.3 Class plots

4.3.1 Summary

Class plots

```

-----
.dist
.qq
.scatter
.scatter3d
.outlier

```

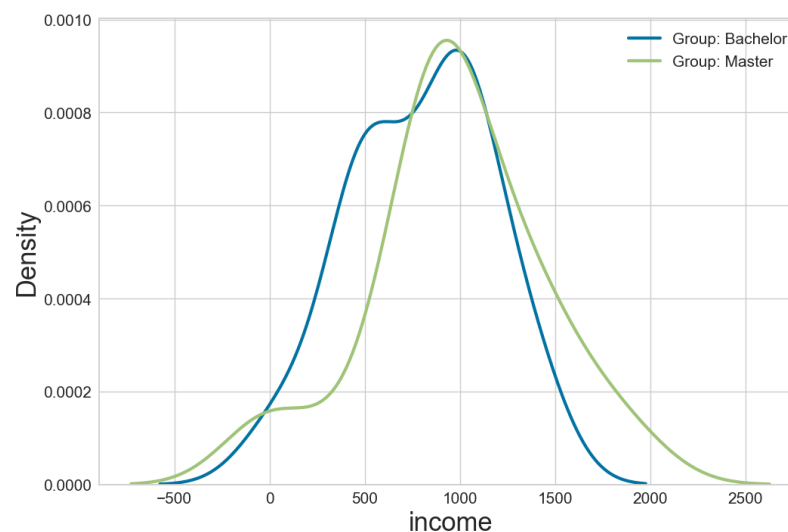
4.3.2 plots.dist

Example:

```

plots.dist(data,var='income',groupvar='level',fig=[12,8],ticksize=15,
labels=25,legsize=15,dark=False,linewidth=3,lineclr='blue',xlabel=None,sh
harediagram=True)

```



Code Structure:

```

arguments
-----
- data = either name of dataframe (var) or vector (pd.Series)
- var, groupvar = name of variable and grouping variable (str) -> only relevant
when data is name of a dataframe
- fig = figure size ([int,int]) [default=[12,8]]
- ticksize = size of ticks (int) [default=15]

```

```

- labels = size of labels (int) [default=25]
- legsize = size of legend (int) [default=15]
- dark = dark background (bool) [default=False]
- linewidth = line width (int) [default=3]
- lineclr = line color (str) [default='blue']
- xlabel = custom x-axis label (str) [default=None]
- sharediagram = whether plots should be done in the same diagram (bool) [default = True]

returns
-----
Distribution plot

```

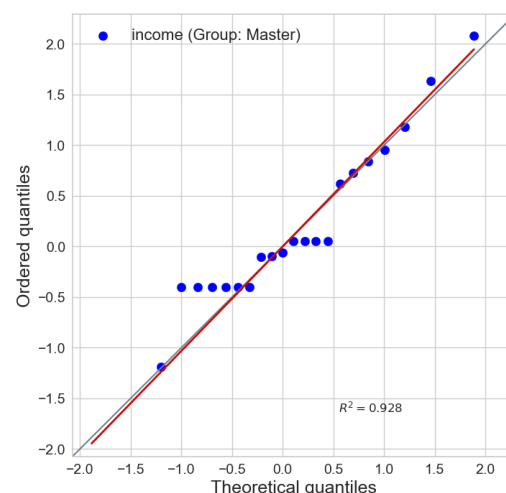
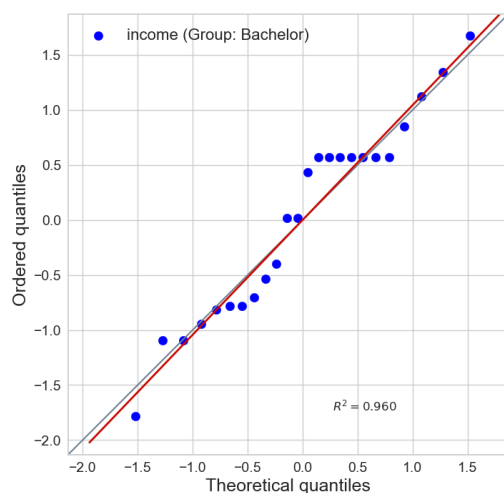
4.3.3 plots.qq

Example:

```

plots.qq(data,var='income',groupvar='level',fig=[12,8],ticksize=14,labels=18,legsize=16,dark=False,rotx=0,roty=90,dotsize=80,confidence=False)

```



Code Structure:

```

arguments
-----
- data = either name of dataframe (var) or vector (pd.Series)
- var, groupvar = name of variable and grouping variable (str) -> only relevant
when data is name of a dataframe
- fig = figure size ([int,int]) [default=[12,8]]
- ticksize = size of ticks (int) [default=15]
- labels = size of labels (int) [default = 18]
- legsize = size of legend [default = 16]
- dark = dark background? (bool) [default = False]
- rotx, roty = rotate x and y (int between 0 and 360) [default: rotx=0, roty=90]
- dotsize = size of dots (int) [default = 80]
- confidence = whether to plot a confidence interval (dec between 0 and 1)
[default = False]

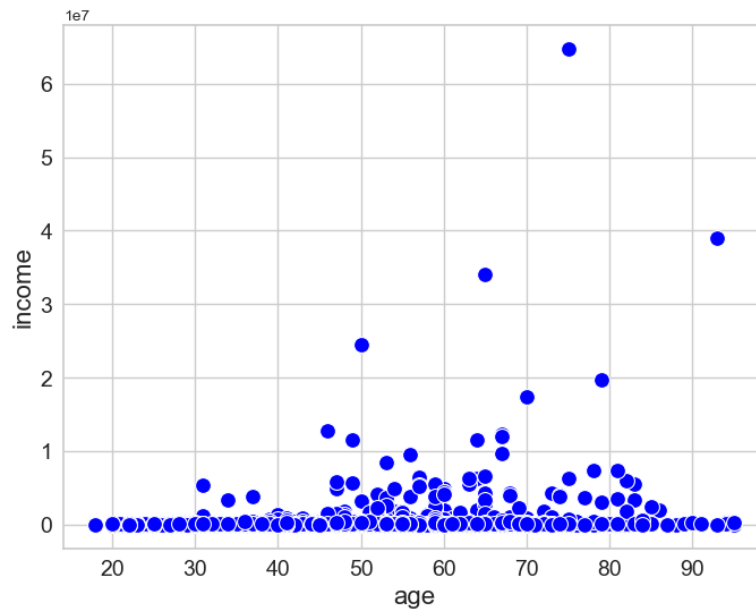
returns
-----
QQ Plot

```

4.3.4 plots.scatter

Example:

```
plots.scatter(data3['age'],data3['income'],data=None,fig=[8,6],ticksize=14,labelsiz  
ze=16,dark=False,dotsize=100,dotclr=['blue'],hue=None,hueclr='tab10',hue_norm=None  
,namexy=[],rotx=0,roty=90,ordinal=False,bubsize=(20,2000),regression=None,  
linewidth=2,lineclr='red',poly_deg=3,legend=False,legendfont=12,legendcol=1,  
legendspaceh=3,legendspacev=2,intext=False,pos=[0,0],txtclr='red',txtsize=12)
```



Code Structure:

```
arguments
-----
- data = name of dataframe (var)
- x,y = names of variables (str)
- fig = figure size ([int,int]) [default = [8,6]]
- ticksize = size of ticks (int) [default = 14]
- labelsiz = size of labels (int) [default = 16]
- dark = dark background? (bool) [default = False]
- dotsize = size of dots (int) [default = 100]
- dotclr = color of dots (str) [default = ['blue']]
- hue = name of third variable whose values are used to color the dots (str)
[default = None]
- hueclr = palette for colors of hue variable (str) [default = 'tab10']
- namexy = names of variables (list) [default = None]
- rotx, roty = rotate x and y (int between 0 and 360) [default: rotx=0, roty=90]
- ordinal = enable bubble plot (bool) [default = False]
- bubsize = size of dots in bubble plot ((int,int)) [default = (20,2000)]
- regression = plot a regression ('linear','logistic','poly') (str) [default =
False]
- linewidth = width of regression line (int) [default = 2]
- lineclr = color of regression line (str) [default = 'red']
- poly_deg = polynomial degrees only relevant if regression = 'poly' (int)
[default = 3]
- legend = enable legend (bool) [default = False]
- legendfont = size of legend (int) [default = 12]
- legendcol = number of legend cols (int) [default = 1]
```

```

- legendspaceh = horizontal space between legend items (int) [default = 3]
- legendspacev = vertical space between legend items (int) [default = 2]
- intext = whether to plot the regression description inside plot (bool) [default
= False]
- pos = position of intext [x,y] ([int,int]) [default = [0,0]]
- txtclr = color of intext (str) [default = 'red']
- txtsize = size of intext [default = 12]
returns
-----
Scatter Plot

```

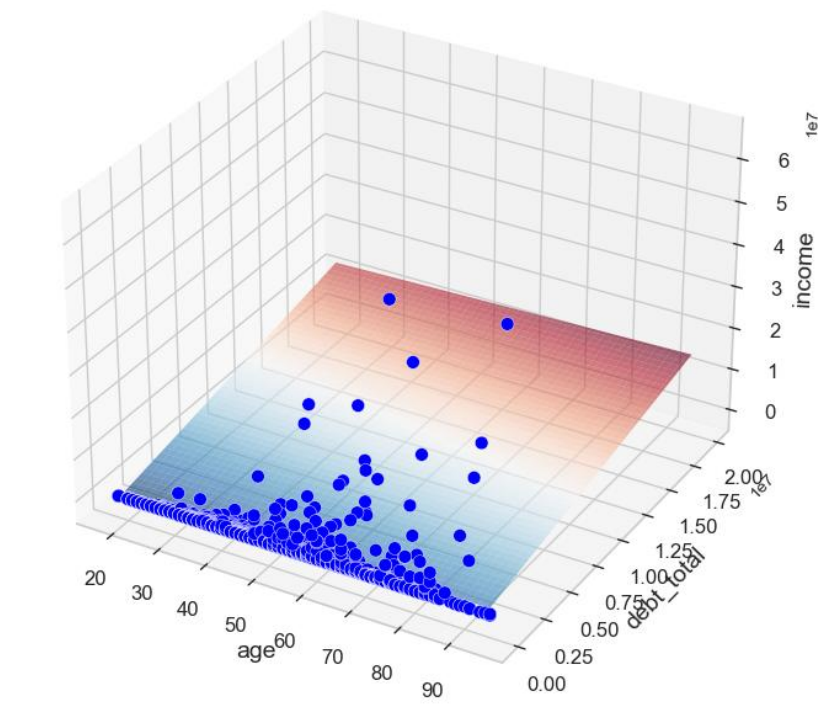
4.3.5 plots.scatter3d

Example:

```

plots.scatter3d(data3[['age','debt_total']],data3['income'],fig=[12,8],ticksize=12
,labelsiz=14,dotsize=60,dotclr='blue',
linreg=True,regpal=plt.cm.RdBu_r,regclr='red',intext=False,pos=[0,0,60],
txtclr='red',txtsize=10)

```



Code Structure:

```

arguments
-----
- X = feature matrix with 2 variables (pd.DataFrame)
- y = dependent variable (pd.Series)
- fig = figure size ([int,int]) [default = [8,6]]
- ticksize = size of ticks (int) [default = 14]
- labelsiz = size of labels (int) [default = 16]
- dotsize = size of dots (int) [default = 60]
- dotclr = color of dots (str) [default = 'blue']
- linreg = whether to plot regression plane (bool) [default = True]

```

```

- regpal = palette for regression plane, can be None (obj) [default =
plt.cm.RdBu_r]
- regclr = color of regression plan used only if regpal=None (str) [default =
'red']
- intext = whether to plot the regression description inside plot (bool) [default
= False]
- pos = position of intext [x,y] ([int,int]) [default = [0,0]]
- txtclr = color of intext (str) [default = 'red']
- txtsize = size of intext [default = 12]
returns
-----
3D Scatter Plot

```

4.3.6 plots.outlier

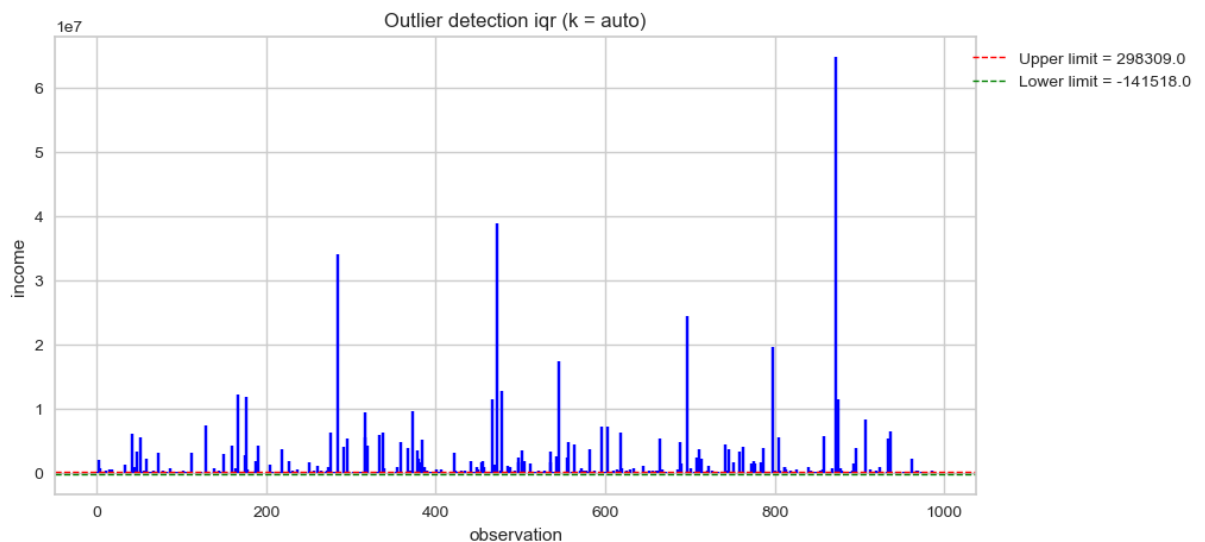
Examples:

- Visualize potential univariate outliers, e.g., for age:

```

plots.outlier(data3['income'],k='auto',method='iqr',limround=0,dtype='univa
riate')

```

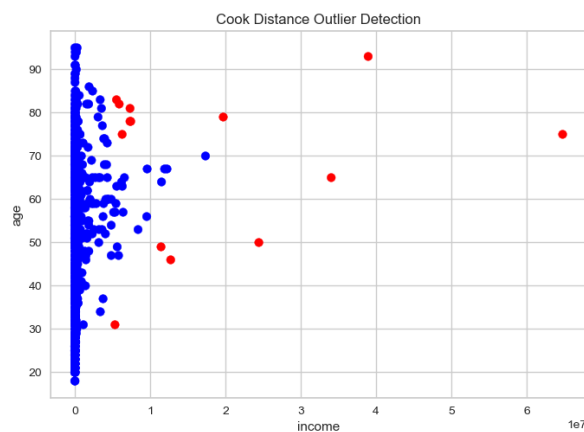


- Visualize potential bivariate outliers, e.g., for age+income:

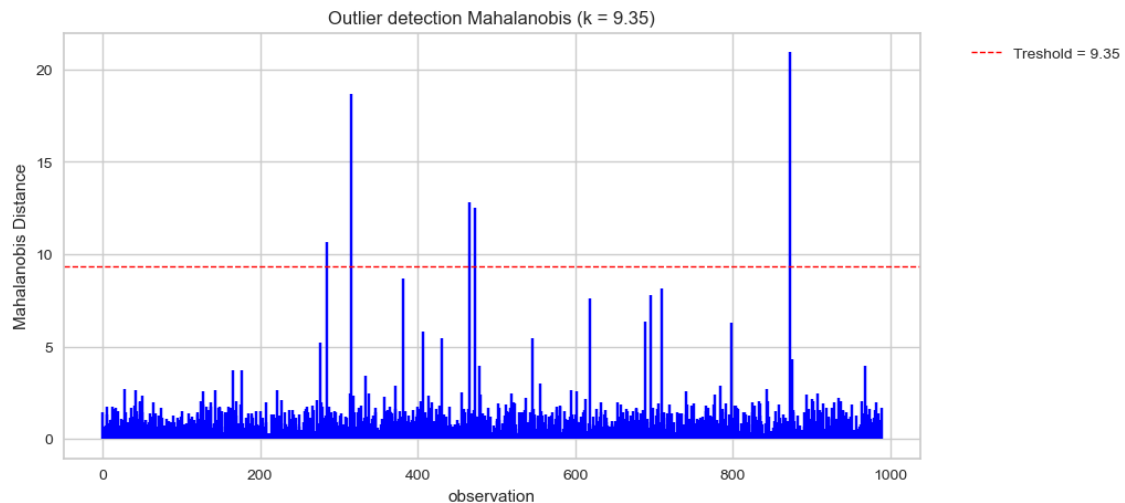
```

plots.outlier(data3[['income']],data3['age'],k='auto',method='Cook',limroun
d=0,dtype='bivariate')

```



- Visualize potential multivariate outliers, e.g., for age+income+debt:



Code Structure:

```
arguments
-----
- x = either feature matrix with variables (pd.DataFrame) or variable (pd.Series)
- y = dependent variable (pd.Series) or None if univariate
- k = outlier detection factor either 'auto' (str) or a dictionary containing
  methods and factors (dic) [default='auto']
- method = outlier detection method ('iqr','zscore','mad','Cook','Mahalanobis')
  (str) [default = iqr]
- limround = decimal places to round thresholds (int) [default = 0]
- dtype = data type ('univariate','bivariate','multivariate') (str) [default =
  'univariate']
returns
-----
uni- and multivariate return a observation vs distance/values plot with thresholds
bivariate returns a scatterplot where outliers are flagged red
```

4.4 Class tests

4.4.1 Summary

```
Class tests
-----
.t
.nonparametric
.independence
.correlation
.association
.equal_var
```

tests.t

Examples:

- One-sample t-test:

```
tests.t.one_sample(data=data3,var='income',nullmean=600).round(4)
```

	var	mean	null mean	t	dof	alternative	p-val	CI95%	cohen-d	BF10	power
One-Sample t-Test	income	674574.7795	600	6.7	988	two-sided	0.0	[477173.42, 871976.14]	0.213	1.146e+08	1.0

- Two-sample t-test:

```
tests.t.two_sample(data=data3,var='income',groupvar='gender').round(4)
```

	var	group	mean	variances	t	dof	alternative	p-val	CI95%	cohen-d	BF10	power
Two-Sample t-Test	income	female	67440.0837	equal	-3.2533	987.0000	two-sided	0.0012	[-1253444.17, -310243.89]	0.2483	14.809	0.9015
		male	849284.1126	unequal	-6.0454	778.8207		0.0000	[-1035718.7, -527969.35]	0.2483	3.821e+06	0.9015

- Paired t-Test

```
tests.t.paired(data_time['income 2019'],data_time['income 2020']).round(4)
```

	var	mean diff	correlation	t	dof	alternative	p-val	CI95%	cohen-d	BF10	power
Paired Sample t-Test	income 2019-income 2020	-227248.599	-0.0482	-0.983	199	two-sided	0.3268	[-683122.89, 228625.7]	0.1003	0.127	0.292

Code Structure:

```
Class t
-----
.one_sample
  arguments
  -----
  - data = name of dataframe (var)
  - var = name of the variable in the dataframe (str)
  - nullmean = mean under null hypothesis (int)
  - alternative = direction of the test ('two-sided', 'left', 'right') (str)
  returns
  -----
  table of one-sample t-test (pd.DataFrame)
.two_sample
  arguments
  -----
  - data = name of dataframe (var)
  - var = name of the variable in the dataframe (str)
  - groupvar = name of the grouping variable (str)
  - alternative = direction of the test ('two-sided', 'left', 'right') (str)
  returns
  -----
  table of two-sample t-test (pd.DataFrame)
.paired
  arguments
  -----
```



```

- data = name of dataframe (var) [optional]
- var1, var2 = names of the variables in the dataframe (str) or vectors
(pd.Series) if data is None
- alternative = direction of the test ('two-sided', 'left', 'right') (str)
returns
-----
table of paired t-test (pd.DataFrame)

Remarks:
-----
- Missing values are automatically removed from the data.

```

4.4.2 tests.nonparametric

Examples:

- Sign test:

```
tests.nonparametric.sign(data=data3,var='income',nullmedian=60000).round(4)
```

	var	median	null median	n(-)	n(+)	n	M	alternative	p-val
Sign Test	income	78395.0784	60000	402	587	989	92.5	two-sided	0.0

- Mann-Whitney U test:

```
tests.nonparametric.mwu(data=data3,var='income',groupvar='gender').round(4)
```

	var	groups	n	median	U-val	alternative	p-val	RBC	CLES
Mann-Whitney U test	income	female/male	221/768	38688.48026/100793.67226	39015.5	two-sided	0.0	0.5403	0.2299

- Wilcoxon test:

```
tests.nonparametric.wilcoxon(data_time['income 2019'],data_time['income 2020']).round(4)
```

	var	n	median	ranksum (+)	ranksum (-)	W-val	alternative	p-val	RBC	CLES
Wilcoxon signed-rank Test	income 2019-income 2020	200	85012.84478099999/87863.574905	9401.5	10498.5	9401.5	two-sided	0.5005	-0.0551	0.486

Code Structure:

```

Class nonparametric
-----
.sign
arguments
-----
- data = name of dataframe (var or pd.Series)
- var = name of the variable in the dataframe (str) or None if data =
pd.Series
- nullmedian = median under null hypothesis (int) [if nullmedian=None ->
nullmedian=median of variable]

```

```

- alternative = direction of the test ('two-sided', 'left', 'right') (str)
returns
-----
table of one-sample sign test (pd.DataFrame)
.mwu
arguments
-----
- data = name of dataframe (var)
- var = name of the variable in the dataframe (str)
- groupvar = name of the grouping variable (str)
- alternative = direction of the test ('two-sided', 'left', 'right') (str)
returns
-----
table of Mann-Whitney U test (pd.DataFrame)
.wilcoxon
arguments
-----
- data = name of dataframe (var) [optional]
- var1, var2 = names of the variables in the dataframe (str) or vectors
(pd.Series) if data is None
- alternative = direction of the test ('two-sided', 'left', 'right') (str)
returns
-----
table of Wilcoxon signed-rank test (pd.DataFrame)

```

4.4.3 tests.independence

Examples:

- Chi2 tests:

```
tests.independence.chi2(data=data3,var1='gender',var2='expenses_last12')
```

	vars	no. categories	test	chi2	dof	p-val	cramer	power
Chi2 Tests	gender	2	pearson	12.068362	2.0	0.002395	0.110465	0.885033
of Independence	expenses_last12	3	cressie-read	11.737415	2.0	0.002827	0.108940	0.875730
			G(log-likelihood)	11.182538	2.0	0.003730	0.106334	0.858654
			freeman-tukey	10.846742	2.0	0.004412	0.104725	0.847363
			mod-log-likelihood	10.569450	2.0	0.005068	0.103378	0.837466
			neyman	10.161636	2.0	0.006215	0.101364	0.821920

- Exact tests:

```
tests.independence.exact(data['gender'],data['married']).round(4)
```

	vars	test	statistic	p-val
Exact Tests	gender	fisher	123.8947	0.0
of Independence	married	barnard	20.9032	0.0
		boschloo	0.0000	0.0

Code Structure:

```
Class independence
-----
.chi2
  arguments
  -----
  - data = name of dataframe (var) [optional]
  - var1, var2 = names of the variables in the dataframe (str) or vectors
  (pd.Series) if data is None
  - Yates = whether to apply the Yates correction (bool) [default = False]
  returns
  -----
  table of chi2 independence test (pd.DataFrame)
.exact
  arguments
  -----
  - data = name of dataframe (var) [optional]
  - var1, var2 = names of the variables in the dataframe (str) or vectors
  (pd.Series) if data is None
  returns
  -----
  table of containing Fisher, Barnard, and Boschloo Exact Test (pd.DataFrame)
```

4.4.4 tests.correlation

Examples:

- Simple Correlation with Kendall correlation coefficient:

```
tests.correlation.simple(data['income'],data['age'],method='kendall').round(4)
```

	var1	var2	n	r (kendall)	CI95%	alternative	p-val	power
kendall Test of Correlation	income	age	989	0.1277	[0.07, 0.19]	two-sided	0.0	0.981

- Partial Correlation with Pearson correlation coefficient:

```
tests.correlation.partial(data=data,var1='income',var2='age',covar=['debt_total']).round(4)
```

	var1	var2	covar	n	r (pearson)	CI95%	alternative	p-val
pearson Partial Correlation Test	income	age	[debt_total]	989	0.1503	[0.09, 0.21]	two-sided	0.0

Code Structure:

```
Class correlation
-----
.simple
  arguments
  -----
  - data = name of dataframe (var) [optional]
```

```

- var1, var2 = names of the variables in the dataframe (str) or vectors
(pd.Series) if data is None
- alternative = direction of the test ('two-sided', 'left', 'right') (str)
- method = correlation coefficient ('pearson', 'spearman', 'kendall') (str)
[default = 'pearson']
returns
-----
table of simple correlation test (pd.DataFrame)

.partial
arguments
-----
- data = name of dataframe (var)
- var1, var2 = name of the variable in the dataframe (str)
- covar = list of names of the covariates ([str, str, ...])
- groupvar = name of the grouping variable (str)
- method = correlation coefficient ('pearson', 'spearman', 'kendall') (str)
[default = 'pearson']
returns
-----
table of partial correlation test (pd.DataFrame)

.pbc
arguments
-----
- x = vector of the numerical variable (pd.Series)
- nom = vector of the nominal variable (pd.Series)
returns
-----
(point-biserial correlation coefficient, p-value) (tup)

.rbc
- x = vector of the ordinal variable (pd.Series)
- nom = vector of the nominal variable (pd.Series)
returns
-----
(rank-biserial correlation coefficient, p-value) (tup)

.eta
- x = vector of the numerical variable (pd.Series)
- nom = vector of the nominal variable (pd.Series)
returns
-----
(eta, p-value) (tup)

.gk_gamma
arguments
-----
- x, y = vectors of the variables (pd.Series)
- alternative = direction of the test ('two-sided', 'left', 'right') (str)
returns
-----
(Goodman and Kruskal's gamma, asymptotic pvalue under null, asymptotic pvalue
under alternative, pvalue standard) (tup)

.cramer
arguments
-----
- x, y = vectors of the variables (pd.Series)
returns
-----
(Cramer's V, p-value) (tup)

```

4.4.5 tests.equal_var

Examples:

- Levene's Test of equal variances:

```
tests.equal_var.levene(data=data, var='income', groupvar='education').round(4)
```

	var	group		f	dof1	dof2	p-val
Levenes Test of Equal Variances	income	education	Mean	18.0211	3	985	0.0000
			Median	7.4110	3	985	0.0001
			Trimmed	118.0536	3	985	0.0000

- Bartlett's Test of equal variances:

```
tests.equal_var.bartlett(data=data, var='income', groupvar='education').round(4)
```

	var	group	T	dof1	dof2	pval
Bartlett's Test of Equal Variances	income	education	781.5386	3	985	0.0

Code Structure:

```
Class equal_var
-----
.levene
  arguments
  -----
  - data = name of dataframe (var)
  - var = name of the variable in the dataframe (str)
  - groupvar = name of the grouping variable in the dataframe (str)
  - rem = whether or not to show explanatory remarks (bool) [default = False]
  returns
  -----
  table of Levene's test (pd.DataFrame)
.bartlett
  arguments
  -----
  - data = name of dataframe (var)
  - var = name of the variable in the dataframe (str)
  - groupvar = name of the grouping variable in the dataframe (str)
  returns
  -----
  table of Bartlett test (pd.DataFrame)
```

4.5 Class regression

Examples:

- Multiple linear regression:

```
X=data[['age','social_pension_income']]
y=data['income']
reg=regression(X,y)
reg.coef.round(4)
```

		coef	stand. coef	std err	t	P> t	[0.025	0.975]
linear reg.	intercept	-601800.0000	-1.4135	361000.000	-1.665	0.096	-1310000.000	107000.000
coefficients	age	22410.0000	0.7455	7004.086	3.199	0.001	8663.452	36200.000
	social_pension_income	5.8039	0.6680	2.683	2.163	0.031	0.539	11.069

```
reg.datafit.round(4)
```

	dv	dof resid	dof model	R2	adj. R2	omnibus (F)	omnibus (p-val)	LL
linear reg. fit	income	986.0	2.0	0.0277	0.0258	14.0628	0.0	-16191.4723

```
reg.asstest.round(2)
```

	test	statistic	p-val
linear reg.	Jarque-Bera	1816978.14	0.0
assumptions	Breusch-Pagan	7.96	0.0187
	Durbin-Watson	2.06	
	Ramsey RESET	41.56	0.0

- Multiple logistic regression:

```
X=data[['age','income']]
y=data['gender']
reg=regression(X,y,regression='logistic')
reg.coef.round(4)
```

		coef	exp(coef)	std err	z	P> z	[0.025	0.975]
logistic reg.	intercept	-1.1406	0.3196	0.269	-4.247	0.000	-1.667	-0.614
coefficients	age	0.0120	1.0121	0.005	2.508	0.012	0.003	0.021
	income	-0.0000	1.0000	0.000	-5.690	0.000	-0.000	-0.000

Code Structure:

```
Class regression
-----
arguments
-----
- X = matrix of independent variables (pd.DataFrame)
- y = dependent variable (pd.Series)
- method = type of regression ('linear','logistic','multinomial','ordinal') (str)
[default = 'linear']
- var = name of the variable in the dataframe (str)
returns
-----
.resid -> (pseudo) residuals of the regression
.pred -> predictions of the regression
.datafit -> general information and goodness of fit of the regression
.coef -> table with coefficients
.vif -> variance inflation factors
.asstest -> tests to check assumptions
.summary -> summary of regression analysis
```

4.6 Class outlier

Examples:

- Detecting univariate outliers:

```
out=outlier.univariate(data['income'])
out.analysis
```

	method	pot. outlier	proportion
extreme value	zscore	11	1.11%
analysis	iqr	184	18.6%
	mad	240	24.27%
E[ND] (>3 std from mean)		2	0.27%

```
out.show(method='zscore')
```

```
[166, 177, 285, 466, 472, 478, 545, 696, 798, 872, 875]
```

```
data.loc[out.show(method='zscore')][['income']].head()
```

	income
166	1.224796e+07
177	1.192216e+07
285	3.405604e+07
466	1.145383e+07
472	3.896337e+07

- Detecting multivariate outliers:

```
out=outlier.multivariate(data[['income']],data['debt_total'])
out.analysis
```

	method	pot. outlier	proportion
extreme value	Cook	18	1.82%
analysis	Mahalanobis	9	0.91%

```
out.show(method=' Mahalanobis')
```

```
[285, 316, 382, 466, 472, 618, 696, 710, 872]
```

Code Structure:

```
Class outlier
-----
.univariate
  arguments
  -----
  - x = vector (pd.Series)
  - k = detection factor either 'auto' (str) or [(dec,dec,dec)] for
zscore,iqr,mad [default = 'auto']
  returns
  -----
  .analysis -> outlier analysis summary
  .show -> indices of outliers detected by the specified method
  arguments
  -----
  - method = outlier detection method ('zscore','iqr','mad') (str) [default
= 'iqr']
.multivariate
  arguments
  -----
  - x = matrix of independent variables (pd.DataFrame)
  - y = dependent variable (pd.Series)
  - k = detection factor either 'auto' (str) or [(dec,dec)] for Cook,Mahalanobis
[default = 'auto']
  returns
  -----
  .analysis -> outlier analysis summary
  .show -> indices of outliers detected by the specified method
  arguments
  -----
  - method = outlier detection method ('Cook','Mahalanobis') (str) [default
= 'Cook']
```


5 Working with Data

5.1 Data Structures

5.1.1 Types

Python mainly processes data by the following structures

- **Series Object.** A series is a vector-like object (a one-dimensional column array) with index. Link:
<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html>
- **List.** A list object is a one-dimensional row array without index. Link:
https://www.w3schools.com/python/python_lists.asp
- **Array Object.** An array is a matrix-like object. Link:
https://www.w3schools.com/python/python_arrays.asp
- **Pandas Dataframe.** A data frame is like an array embedded within a table. Link:
<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html>
- **Dictionary.** A dictionary is a mapping from keys to values {key:value}. Link:
https://www.w3schools.com/python/python_dictionaries.asp

5.1.2 Converting Data Structures

Convert series to (let ser be the name of the object)	<ul style="list-style-type: none">• list <code>list(ser)</code>• array <code>ser.array</code>• dataframe <code>pd.DataFrame(ser)</code>
Convert list to (let li be the name of the object)	<ul style="list-style-type: none">• series <code>pd.Series(li)</code>• array <code>np.array(li)</code>• dataframe <code>pd.DataFrame(li)</code>
Convert array to (let ar be the name of the object)	<ul style="list-style-type: none">• series <code>pd.Series(ar)</code>• list

	<pre>ar.tolist()</pre> <ul style="list-style-type: none"> • dataframe <pre>pd.DataFrame(ar)</pre>
Convert dataframe to (let df be the name of the object)	<ul style="list-style-type: none"> • series (column with name c) <pre>df['c'] or df.c</pre> <ul style="list-style-type: none"> • list (column with name c) <pre>df['c'].to_list() or df.c.to_list()</pre> <ul style="list-style-type: none"> • array <pre>df.to_numpy()</pre>

5.1.3 Selecting Cells, Columns and Rows

- **Series/Lists.** Select ith item:

```
ser[i]  
li[i]
```

- **Array.** Select cell with row index i and column index j:

```
ar[i,j]
```

- **Dataframe.**

- Select column with name c:

```
df['c']
```

- Select row with index i:

```
df.loc[i]
```

- Select cell with column name c and row index i:

```
df.loc[i,'c']
```

- Select cell with column index j and row index i:

```
df.iloc[i,j]
```

- **Dictionary.** Select value for key k:

```
dic[k]
```

5.2 Data Frames

5.2.1 Dropping Columns

Suppose your data frame is named df.

- The following command drops the column named c. Only one column can be dropped:

```
del df['c']
```

- The following command drops all columns whose names are in [] (c1,c2,...):

```
df=df.drop(['c1','c2',...],axis=1)
```

- The following command keeps all columns whose names are in [] (c1,c2,...):

```
df=df[['c1','c2',...]]
```

5.2.2 Dropping Rows

Suppose your data frame is named df.

- The following command drops all rows whose numbers are in [] (r1,r2,...):

```
df=df.drop([r1,r2,...])
```

- The following command keeps all rows whose numbers are in [] (r1,r2,...):

```
df=df.loc[[r1,r2,...]]
```

- The following command drops all rows where the values in column c are greater or equal x (x = a number):

```
df=df[df['c']<x]
```

- Similarly, you may use a greater (>), a equal (==) or an unequal (!=) statement in this code.

- The following command keeps all rows where the values in column named c1 < x and, at the same time, those in column named c2 are > y:

```
df=df[(df['c1']<x) & (df['c2']>y)]
```

- Instead of an and (&) you may also use an and/or operator (|) in this code.

- The following commands drop all rows where the values in column c take on the values in [] (v1,v2,...):

```
indices = df[df['c'].isin([v1,v2,...])].index  
df=df.drop(indices)
```

5.2.3 Creating new Columns

Suppose your data frame is named df.

- The following code creates a column named new that contains only one value x (which can be a number or a text string):

```
df['new']=x
```

- The following command creates a new column named new whose values are the sum of the values of the columns named c1 and c2:

```
df['new']=df['c1']+df['c2']
```

- The following command creates a new column with name new that assumes value v1 if the column with name c equals value v2 and, otherwise, value v3:

```
df['new']=df['c'].apply(lambda x: v1 if x==v2 else v3)
```

- Create a column conditional on the values of several other columns.
 - Step 1. Define a function. The following function assigns 'yes' if column c1 equals x and c2 is greater than y. It assigns 'no' if c1 equals x and y is less or equal y. Otherwise (if c1 unequal x), it assigns 'maybe':

```
def f(a):
    if a(c1) == x and a(c2)>y:
        return 'yes'
    elif a(c1) == x and a(c2)<=y:
        return 'no'
    else:
        return 'maybe'
```

- Step 2. Apply the function:

```
df.apply(f,axis=1)
```

5.2.4 Rename Columns and and Replace Values

- The following command renames the old columns with new names (old_name: new_name):

```
df=df.rename(columns={'colname1_old':'colname1_new','colname2_old':'colname1_new',...})
```

- The following command replaces value x in column c1 by 1 and the value y in column c2 by 'a':

```
df.replace({'c1': {x: 1}, 'c2':{y:'a'}})
```

5.2.5 Merging Data Frames

Data frames (df1, df2) can be merged as follows:

- Append:

```
df1.append(df2)
```

- Link:

<https://www.geeksforgeeks.org/python-pandas-dataframe-append/>

- Concat:

```
frames=[df1,df2]
df_merged = pd.concat(frames)
```

- Remark: The data frames are stacked vertically and need to have exactly the same number of columns with identical names.
 - However, you may set an inner join (horizontal merge) by adding join='inner':

```
frames=[df1,df2]
df_merged = pd.concat(frames,axis=1,join='inner')
```

- Link:
<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html>

5.2.6 Some Useful Commands

Command	Description
<code>df.describe()</code>	returns the descriptive statistics of the data frame
<code>df.dropna()</code>	eliminates all missing values from your data frame
<code>df._get_numeric_data()</code>	eliminates all columns from <i>df</i> that do not contain numeric data
<code>df.groupby('colname')</code>	groups your data frame w.r.t. the grouping variable named colname
<code>df.head(i)</code>	shows the first <i>i</i> rows of the data frame
<code>df.round(i)</code>	rounds your data frame to <i>i</i> decimal places
<code>df.shape</code>	returns a list tuple: (number of columns, number of rows)
<code>df.sort_values(by=['colname'])</code>	sorts the data frame w.r.t. to the values of the column named colname
<code>df['colname'].count()</code>	shows the number of rows of the column named colname
<code>df['colname'].cumsum()</code>	returns a series object containing the cumulated sum of the entries of the column named colname
<code>df['colname'].mean()</code>	returns the mean of the column named colname
<code>df['colname'].std()</code>	returns the standard deviation of the column named colname

<code>df['colname'].sum()</code>	shows the sum of the entries of the column named colname
<code>df['colname'].unique()</code>	returns a list of the unique values in the column named colname
<code>df['colname'].value_counts()</code>	returns a list of all values of the column named colname and the number of observations per value

5.3 Data Visualization

In Python, it is very easy to create a plot based on two columns of a data frame:

```
df.plot(x='colname1',y='colname2')
```

Example.



5.4 Programing Functions

Python allows to easily program functions. In order to program a function use the command `def f(a1,a2,...): syntax return y`. The arguments of the function are a_1, a_2, \dots and y is the output of the function.

Example.

```
def f(x,a,b):
    return a*x+b
```

```
f(2,3,1)
```

7

WE HOPE THAT THIS BRIEF INTRODUCTION SHOWED YOU SOME ABILITIES OF PYTHON AND HELPED YOU TO DO SOME STEPS ON YOUR OWN.

ENJOY THE PROGRAM!

FLORIAN KAUFFELDT

Please report any typos/errors in this document to florian.kauffeldt@hs-heilbronn.de