📊

Cheat Sheet

Data Manipulation & Visualization with Pandas, Seaborn and matplot

Aa TOPIC	IMPORTANT to remember	Details
TRANSFORMING DATA	- -	- -
DataFrames		- Rectangular data is represented as a DataFrame object. - Every value within a column has the same data type, - different columns can contain different data types.
Exploring a DataFrame	df.describe() = summary statistics for numerical columns, like mean and median	.head() = returns the first few rows of the DataFrame .info() = displays the names of columns, the data types they contain, and whether they have any missing values .shape = contains a tuple that holds the number of rows followed by the number of columns. !!! without parentheses .describe() = summary statistics for numerical columns, like mean and median. // good for a quick overview // "count" is the number of non-missing values in each column .values = contains the data values in a 2-dimensional NumPy array. .columns = contains column names .index = contains row numbers or row names
Sorting	df.sort_values(["column name 1", "column name 1"], ascending = [True, False])	.sort_values("column name") = for sorting rows .sort_values("column name", ascending = False) = for sorting in descending order .sort_values(["column name 1", "column name 1"]) = for sorting by multiple variables .sort_values(["column name 1", "column name 1"], ascending = [True, False]) = for sorting by multiple variables with defined direction of sorting
Subsetting Columns	df[["column name", "column name"]]	df["column name"] = to zoom in on just one column df[["column name", "column name"]] = the outer square brackets = subsetting the DataFrame = the inner square brackets = creating a list of column names to subset.
Subsetting Rows	df[df["column name"] > 50]	df["column name"] > 50 = a logical condition to filter against // results into True or False value for every row df[df["column name"] > 50] = subset the rows that fulfills the logical condition df[df["column name"] == "filter text"] = subset the rows that fulfills the text filter df[df["column name"] > "yyyy-mm-dd"] = subset the rows that fulfills the date condition // date must be in "quotes" and follow the format yyyy-mm-dd
Subsetting based on Multiple Conditions	df[(df["column name"] > Y) & (df["column name"] =="conditionX")]	condition_1 = df["column name"] > Y condition_2 = df["column name"] =="conditionX" df[condition_1 & condition_2] ALTERNATIVELY df[(df["column name"] > Y) & (df["column name"] =="conditionX")] = to combine conditions using logical operators // only rows that meet both of these conditions will be subsetted .isin() condition_1_or_2 = df["column name"].isin(["con_1", "con_2"]) df[condition_1_or_2] = to filter on multiple values of a categorical variable
Adding new columns	df["new_column"] = df["column_calc_basis"] / X	df["new_column"] = df["column_calc_basis"] / X = left-hand side of the equals, we use square brackets with the name of the new column we want to create // on the right-hand side, we have the calculation // IMPORTANT: both the existing column and the new column we just created are in the DataFrame
AGGREGATING DATA	- -	- -
Summarizing numerical data	df["column name"].quantile(q = 0.25)	IN GENERAL: methods are performed by dafault over the index axis DataFrame.methode(axis='columns') gives the method over columns NOT index df["column name"].mean() = shows the "center" of the data of a specific column .median() .mode() = Get the mode(s) of each element along the selected axis // the mode of a set of values is the value that appears most often. It can be multiple values. .min() .max() .var() = shows the variance .std() = return sample standard deviation over requested axis // normalized by N-1 by default. This can be changed using the ddof argument (Delta Degrees of Freedom) .sum() .quantile(q = 0.25) Return values at the given quantile// If q is a float, a Series will be returned where the index is the columns of self and the values are the quantiles. // Value between 0 <= q <= 1, the quantile(s) to compute.

TOPIC	IMPORTANT to remember	Details
Aggregating = Custom Ungrouped Summary statistics	df["column name"].agg(func_name)	.agg(func_name) = aggregate using one operation/function over the specified axis.
Multiple Ungrouped Summaries	df[["column name_x","column name_y"]].agg([func_name_1,func_name_2]) df["column name"].cumsum()	.agg([func_name_1, func_name_2]) = aggregate using more operations/functions over the specified axis. .cumsum() = return cumulatively the sum element by element/ returns an entire column of a DataFrame, rather than a single number .cummax(), .cummin(), .cumprod() = returns the operations cumulatively
Counting	df.drop_duplicates(subset=["column name_1","column name_2"]) df["column name"].value_counts()	df.drop_duplicates(subset="column name") = removes rows that contain an argument in a column selected that was already listed earlier in the dataset df.drop_duplicates(subset=["column name_1","column name_2"]) = removes duplicate pairs .value_counts() = for counting how often a value is in the column selected .value_counts(sort=True) = for sorting in descending order the countings how often a value is in the column selected .value_counts(normalize=True) = for turning the countings how often a value is in the column selected into proportions
Grouped Summary Statistics		df.groupby("column name_grouped over")["column name used as criteria"].statistical_function() = gives the outcome of the statistical function e.g. mean for the grouped-over column for the criteria selected
Multiple Grouped Summaries	df.groupby("column name_grouped over")["column name used as criteria"].agg([statistical_function_1(), statistical_function_2(), statistical_function_3()])	df.groupby("column name_grouped over")["column name used as criteria"].agg([statistical_function_1(), statistical_function_2(), statistical_function_3()]) = gives the outcome of the multiple statistical functions for the grouped-over column for the criteria selected df.groupby("column name_1_grouped over", "column name_2_grouped over")["column name used as criteria"].statistical_function_() = gives the outcome of the statistical functions for the multiple grouped-over columns for the criteria selected
VISUALIZING DATA	- -	- -
Plot with subplots type count	`def plot_with_sub_countplots(): fig, axes = plt.subplots(1,4, figsize=(20, 5)) sns.despine(left=True, bottom=True) n=0 fig.suptitle('title') cols = ['col1', 'col2','col3','col4'] for col in cols: descending_order = df[col].value_counts().sort_values(ascending=False).index sns.countplot(ax=axes[n], data=df, order=descending_order, palette= ['#3B89F3','#3B89F3','#3B89F3','#3B89F3','#3B89F3','#3B89F3','#3B89F3','#3B89F3','#3B89F3','#3B89F3','#3B89F3'], x=col) axes[n].set_title('Count of {}'.format(col)) axes[n].set_xlabel('') axes[n].set_ylabel('') axes[n].tick_params('x',labelrotation=45) n += 1 plt.show()`	- shows four sub plots of the same type - show the count in each plot - useful for overview, e.g. if dataset is in balance
Countplot / bar plot / vertical	`def countplot(variable_name): df[variable_name].value_counts().head(10).sort_values(ascending=False).plot(kind='bar',figsize=(5,5), color= ['#3B89F3']) plt.xticks(rotation=45) sns.despine(left=True, bottom=True)`	- shows one countplot for one variable - shows only the top ten when many categories in variable - vertical orientation
Countplot / bar plot / horizontal	`def h_countplot(variable_name): df[variable_name].value_counts().head(10).sort_values(ascending=True).plot(kind='barh', figsize=(5,5), color= ['#3B89F3']) sns.despine(left=True, bottom=True)`	- shows one countplot for one variable - shows only the top ten when many categories in variable - horizontal orientation
Frequency plot / bar plot / vertical	`def freq_of_something(variable_name): col_list = df_new[variable_name].dropna() col_list=col_list.unique().tolist() df_new['freq'] = df_new[df_new['target']>0].groupby([variable_name]) [variable_name].transform('count') df_new['freq_of_something'] = df_new.groupby([variable_name]) [variable_name].transform('count') df_new['freq_target'] = df_new.groupby([variable_name]) ['freq'].transform('max') f, ax = plt.subplots(figsize=(5,5)) freq_of_something = df_new.loc[:, (variable_name, "freq_target",'freq_of_something')] freq_variable.drop_duplicates(keep="first", inplace=True) freq_variable.sort_values('freq_target', ascending = True, inplace=True) freq_of_something = freq_of_something.dropna().sort_values('freq_of_something', ascending = False).head(10) variable_data = freq_of_something[freq_of_something[variable_name].isin(col_list)].head(10) plt.bar(height="freq_candidate", x=variable_name, data=freq_of_something, label="total candidates", color='#3B89F3') plt.bar(height="freq_target", x=variable_name, data=variable_data, label="something: yes", color="#E93356") ax.legend(loc="upper right") ax.set(xlabel =None, ylabel = 'frequency of "something"') plt.xticks(rotation=45) sns.despine(left=True, bottom=True)`	- shows one frequency plot for one variable - vertical orientation
Proportion plot / bar plot / vertical	`def prop_of_something(variable_name, width): fig_dims = (width, 5) fig, ax = plt.subplots(figsize=fig_dims) df['freq'] = df[df['target']>0].groupby([variable_name])[variable_name].transform('count') df['freq_of_something'] = df.groupby([variable_name])[variable_name].transform('count') prop_variable = df.copy() prop_variable = prop_variable.sort_values('freq_of_something', ascending = False) lp=sns.barplot(x=variable_name, y='target', data=prop_variable, color='#3B89F3', ci=95) lp.axes.set_ylim(0,0.5) lp.axes.set(xlabel =None, ylabel = 'proportion of "something"') lp.set_xticklabels(lp.get_xticklabels(), rotation=45); ax.yaxis.set_major_formatter(PercentFormatter(1.0)) sns.despine(left=True, bottom=True)`	- shows one frequency plot of proportions for one variable - vertical orientation
Distribution plot for one variable	`def displot(variable_name): sns.despine(left=True, bottom=True) ax=sns.displot(x= variable_name, y='target', data=df, kind='kde', color = '#3B89F3', fill=True) ax.set(ylabel='distribution: willing to change job (0=no 1=yes)')`	- good for variables on cardinal scale - shows the distribution of categories of a variable in 'cloud' - good if there are few NaN or Null values and the number of distributions is <10
Pie plot for one variable	`def pie_plot(variable_name): values = df[df['target'] == 1][variable_name].value_counts() labels = values.keys() bar,ax = plt.subplots(figsize=(8,8)) plt.pie(x = values, labels = labels , autopct="%.1f%%",pctdistance=0.9, colors = ('#3B89F3','#f39100','#009bb4', '#94c11c','#BF7300' ,'#ffdd00', '#C6D59F')) plt.title('')`	- good for variables on all scale levels - shows the distribution of categories of a variable - good if there are few NaN or Null values and the number of distributions is <10
Regression plot for one variable	`def regplot(variable_name): sns.despine(left=True, bottom=True) ax=sns.lmplot(x= variable_name, y='target', data=df, scatter_kws={"color": "white"}, ci=None,y_jitter=.02, logistic=True, truncate=False, line_kws={'color' : 'red'}) ax.set(ylabel='distribution: willing to change job')`	- good for variables on cardianal scale - good to show regression curve
Displot with mean and median for one variable	`def displot_median_mean_variable(): sns.displot(df['variable_name'], kde=False, color='#3B89F3') plt.axvline(x=df.variable_name.mean(), linewidth=3, color='#E93356', label="mean") plt.axvline(x=df.variable_name.median(), linewidth=3, color='#ffdd00', label="median") plt.ylabel("Count") plt.legend(["mean", "median"]) plt.xticks(rotation=45) sns.despine(left=True, bottom=True)`	- good for variables on cardinal scale - descriptive statistic for a variable - shows mean and median
Learning Progress		