PANDAS CHEAT SHEET

ABBREVIATIONS		
nv = new variable	nv_df = new dataframe variable	
v = variable	df = dataframe variable	

READ & WRITE FILES		
function	**command**	**explanation, comments, examples**
load file	nv = "folder_name/file_name"	Can use indirect path as well (os.path.join()).
read csv file	nv_df = pd.read_csv(v)	When file is read it is also converted into a dataframe.
write csv file	df.to_csv("new_file_name.csv", index=False)	index=False, exporting without index.
write xlxs file	df.to_excel("new_file_name.xlxs", index=False)	

ORGANIZING DATA		
function	**command**	**explanation, comments, examples**
rename columns	nv_df = df.rename(columns={"original1":"new1","original2":"new2"})	
change column order	nv_df = df[["column1", "column2", "column3"]]	It can be done with a passing a variable too: nv = ["column1", "column2", "column3"] df = df[nv]
adding a new column	df["new_column"] = nv	nv = contition that we pass in the new variable, for example: membership_weeks = training_data["Membership(Days)"]/7 training_data["Membership(Weeks)"] = membership_weeks training_data.head()
sorting data: descending order	nv_df = df.sort_values(["column"], ascending=False)	
sorting data: ascending order	nv_df = df.sort_values(["column"], ascending=True)	
reset index	nv_df = df.reset_index(drop=True)	After sorting, the indexes are not in order. If we want to have them in order we use reset_index().

INSPECTING DATA		
function	**command**	**explanation, comments, examples**
general information	**nv = df.info()**	nv = optional
count rows in each column	**nv = df.count()**	Get a count of the rows for all and each column containing data ("null", NaNs, are not included).
count rows in a specific column	**nv = df.["column"].count()**	Count all rows in given column.
unique values in a column	**nv = df["column"].value_counts()**	How many times unique rows appear in a column. Note: see **set_index()** method for comparison.
list all columns	**df.columns**	Returns columns in a list.
unique values	**nv = df["column"].unique()**	Rreturns a list of unique values. Use lento count = > nv = len(df["column"].unique()
missing values	**df.isnull()**	Determine empty rows, returns Boolean.
sum of missing values	**df.isnull().sum()**	Determine empty rows and return a sum of a missing values.
not-missing values	**df.notnull().sum()**	Determine not-empty rows and return a sum of a not-missing values.
describe	**df.describe()**	Returns measuers of central tendency.
minimum/maximum	**df.min() / df.max()**	Returns minimum / maximum
columns	**df.columns**	Returns name of columns in a list.
data type	**df.dtypes**	Determine datatype.
data ytpe	**df.column.dtype**	determine datatype for specific column.
loc/filtering data	**nv_df = df.loc[[rows], ["column1", "column2"]]**	nv_df = df.loc[:, ["column1", "column2"]] = selecting all rows
loc with conditional and selected columns after comma	**nv_df = df.loc[(df["column"] > 7), ["column1", "column2"]]**	
loc with conditional operators	**nv_df = df.loc[(df["column"] == "row/value") & (df["column"] == "row/value") | (df["column"] > n)**	

SET NEW INDEX		
function	**command**	**explanation, example**
set new index	**nv= df.set_index(["column_to_be _new_index"])["column"]**	set new index for given column and get a series with a specific column.
Remove the index name.	**df.index.name = None**	

CLEANING DATA		
function	**command**	**explanation, comments, examples**
replace exact given string into new one	**df["column"] = df["column"].str.replace("existing", "replacement")**	for word in prefixes_suffixes: student_data_df["student_name"] = student_data_df["student_name"].str.replace(word, "") (.str = datatype change. We need to convert object to a string before using replace() method!)
replace/rename row names to a new one	**df = df.replace({"old_row_name": "new_row_name"})**	Use when rows has similar names and can be combined into one.
drop	**nv_df = df.dropna(how = 'any')**	Drops the row with NaNs.
fill	**nv_df = df.fillna(value)**	To fill the row with certain value.
to tilst	**nv = df["column"].tolist()**	Add column to a new list.
split	**split()**	split string object on the whitespace, or where there is no text.
	set()	The set() method returns all unique items in a list when that list is added inside the parentheses
	strip()	The strip() method removes any combination of letters and words that are inside the parentheses.
delete a column	**del df['column']**	

MERGE FILES		
function	**command**	**explanation, comments, examples**
merge files	**nv_df = pd.merge(first_df, second_df, on=["column", "column"])**	Note: best if combine on a same column with the same name (rename if neccessary).
merge files and matematical operators		Note: when working with merged dataframe, caution how we use sum() When we apply the sum() method on the **merged_df["column"]** we get combined output, since rows are repeting. Instead use sum() **on single_df["column"]**.
setting sufixes with merge files	**nv_df = pd.merge(first_df, second_df, on=["column", "column"], suffixes = ("sufix1","sufix2"))**	

DISPLAYING DATA		
function	**command**	**explanation, comments, examples**
first 5	**df.head()**	
last 5	**df.tail()**	
first n	**df.head(n)**	
last n	**df.head(n)**	
list desired columns	**nv = pd[["column1" "column2", "column3"]]**	note: you can do it with loc too

CALCULATIONS		
function	**command**	**explanation, comments, examples**
sum	**nv = df["column"].sum()**	calculates sum for specific colum
horizontal sum	**df["new_column"] = df.sum(axis = 1)**	horizontally sum up values (same row, different columns) + adding results in a new column with df["new_column"]
other calculations	**nv = df["column"].mean()** **.max(), min(), etc,...**	average_reading_score = school_data_complete_df["reading_score"].mean() calculates average for specific column.
operations between columns + creating a new column	**nv = df["column1"] / df["column2"]** **df["new_column"] = nv**	
matematical operation for each row in selected column	**nv = df["column"] matematical_operator n**	nv = df["column"] /n

CONDITIONALS		
function	**command**	**explanation, comments, examples**
conditionals	**nv = df["column"] >= n**	**returns bolean**
conditionals cont.	**nv = df[df["column"] >= n]**	**returns values (data/dataframe) with this condition**
combine conditionals	**nv = df[(df["column"] >= n) & (df["column"] >= n)]**	

DATA FRAMES		
function	**command**	**explanation, comments, examples**
DATA FRAME : list of dictionaries	**df = pd.DataFrame(** **[{"column1": value1,** **"column2": value2,** **"column3":value3}])**	
DATA FRAME : list of dictionaries	**df = pd.DataFrame([** **{"column1" : "value1", "column2": "value2" }** **{"column1" : "value1.1", "column2": "value2.2" })]**	
DATA FRAME : dictionary of lists	**df = pd.DataFrame({** **"column1": ["value1", "value2", "value3"],** **"column2": ["value1", "value2", "value3"] })**	
DATA SERIES	**nv = pd.Series(v)**	

FORMATTING		
function	**command**	**explanation, comments, examples**
map & formatting & changing data type	**df["column"] = df["column"].astype(type).map("{:,.0f} ".format)**	strongest_df["HP"] = strongest_df["HP"].map("${:.2f} millions".format)
new column order	**df = df[["column1", "column2", "column3"]]**	Note: also with passing a variable: Assign district summary df the new column order. district_summary_df = district_summary_df[new_column_order]

CHANGE DATA TYPE		
function	**command**	**explanation, comments, examples**
change data type	**df["column"] = df["column"].loc[:,"column"].astype(float)**	converted_ufo["duration (seconds)"] = converted_ufo.loc[:, "duration (seconds)"].astype(float)
CREATING BINS & GROUPING		
function	**command**	**explanation, comments, examples**
CREATING BINS **(1)creating bins: range and labels** **(2)cutting** **(3)cutting with creating a new row**	**(1a)bins_variable=[1st_point(i), 2nd_point(ni),3rd_point, etc]** **(1b)labels_variable=["","","","",""] one less than element then in bins** **(2)pd.cut(df["column"], bins_variable, labels=labels_variable)** **(3)df["new_column"] = pd.cut(ted_talks_df["views"], views_bins, labels=views_labels)**	creating bins/categories, based on selected column. (1a)spending_bins = [0, 585, 630, 645, 675] (1b)group_names = ["<$584", "$585-629", "$630-644", "$645-675"]
GROUPING **(1)create a group based on a column**	**nv = df.groupby(["column_to_be_grouped_by"]).matemathical_operator()["column_matematical_operator_apply_to"]**	nv = df.groupby(["column"]).mean()["column1"] OR w/o matemathical operator: nv = df.groupby("column") => use this when establishing bins and you want to check how many elements in each bin. print(ted_group["views"].count())
combinations	**nv_df = df.groupby(["column_to_be_grouped_by"]).matemathical_operator()["column(s)_matematical_operator_apply_to"]**	ted_group = ted_talks_df.groupby(["View Group"]).mean()[["comments", "duration", "languages"]] ted_group *NOTE: if you skip columns the matematical operator will apply to all.