Spring 2023 Econ 148 Midterm Reference Sheet

Pandas

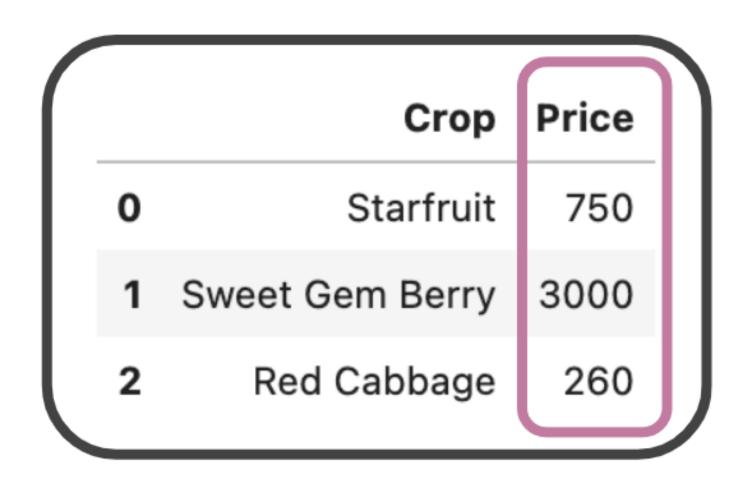
DataFrames & Series

In Pandas, tables are called **DataFrames**. We can think of them as a sequence of columns called **Series**.

This is a DataFrame:

```
farm = pd.DataFrame(
{"Crop":["Starfruit", ...],
"Price":[750,...]})
```

This is a series: farm["Price"]



Crop Price

Starfruit 750

2 Red Cabbage 260

.loc and .iloc accessors

We have two main ways of accessing rows and columns.

```
.loc[ ] lets us grab entries by their label:
df.loc[row_names, col_names]
>> farm.loc[1:2, :]

.iloc[ ] lets us grab entries by their index:
df.iloc[row_indices, col_indices]
>> farm.iloc[1:3, :]
Crop Price

1 Sweet Gem Berry 3000
2 Red Cabbage 260
```

Note that iloc is right-end exclusive!

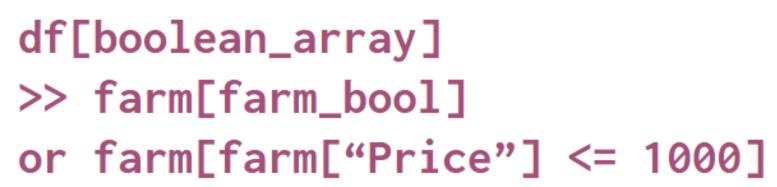
Boolean filtering

We can filter out rows of our DataFrame using a Boolean array of True and False values.

First, apply a Boolean operator to the Series we want to use for filtering:

```
df["column_name"] (<, >, ==, etc.) value
>> farm_bool = farm["Price"] <= 1000</pre>
```

Then, use square brackets to filter out all False values from the DataFrame:



Joining DataFrames using .merge

We can join two DataFrames using the .merge method. The DataFrames will pair up rows that share a common column.

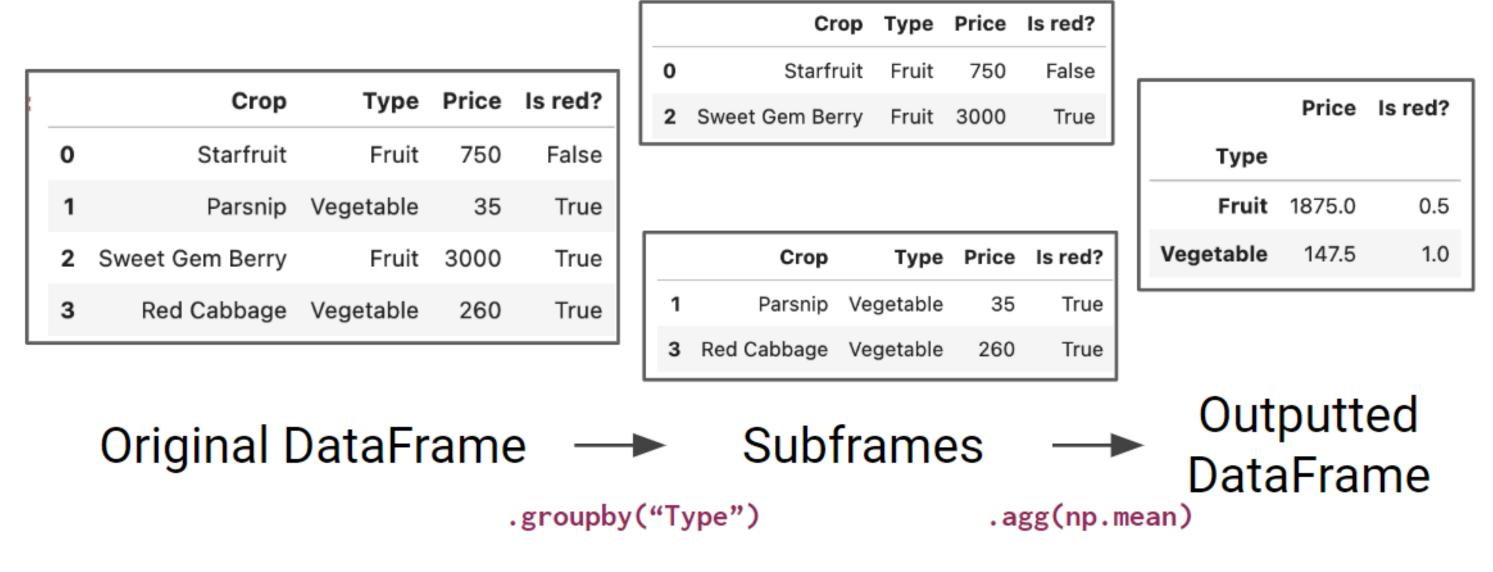
```
pd.merge(df1, df2, left_on="column_name", \
right_on="column_name", how=join_type)
```

You'll learn more about join types and primary/foreign key relationships when we study SQL later in the course.

Grouping with .groupby

If we want to group all entries by their type in a certain column, we can call df.groupby()

```
df.groupby("column_name").aggregator_func(func)
>> produce.groupby("Type").agg(np.mean)
```



We can use many aggregator functions on a GroupBy object:

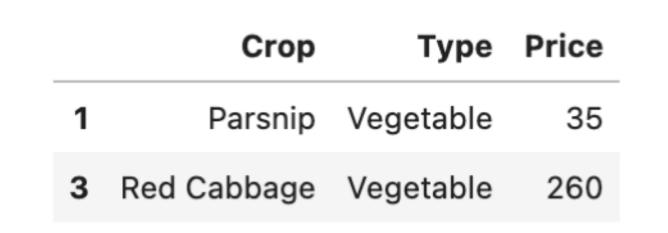
```
gb.agg(func)
gb.mean()
gb.max()/gb.min()
gb.sum()
gb.first()/gb.last()
gb.filter(func)
```

Filtering groups using .filter

Sometimes we only want to keep rows that belong to a group satisfying some condition.

```
produce.groupby("Type").filter(lambda df: df["Price"].mean() < 200)</pre>
```

Here, our filter function takes in a DataFrame (a GroupBy subframe). It outputs one Boolean value. If **True**, all rows belonging to this group are kept in the final DataFrame. If **False**, all rows in this group are omitted.



Mean price of vegetables: 147.5 Mean price of fruit: 1875 So, only the vegetables are kept!

Importing and Exporting Dataframes

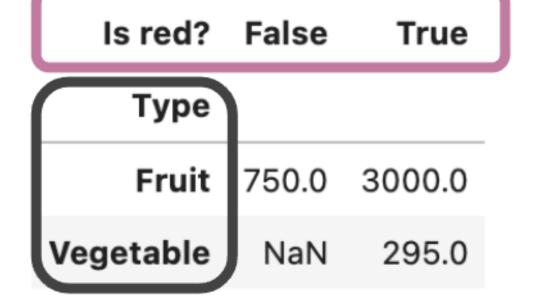
CSV: pd.read_csv reads a comma-separated values (csv) file into DataFrame.

```
pandas.read_csv(filepath, sep, delimiter, encoding,
low_memory=True)
```

Creating pivot tables with .pivot_table

Sometimes we want to group our data by two columns:

pd.pivot_table(data=produce, index="Type", columns="Is
red?", values="Price", aggfunc=sum)



index gives the rows of the table columns gives the columns

To fill out the cells, we apply aggfunc to values

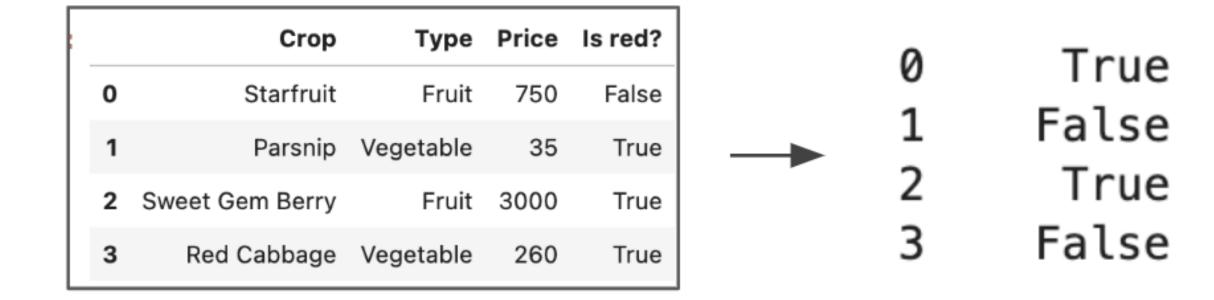
Write object to a comma-separated values (csv) file.

```
DataFrame.to_csv(filepath, sep=',' columns=None,
header=True, index=True, index_label=None, mode='w',
encoding=None)
```

Manipulating strings with .str

The .str accessory tells Pandas to perform operators on a Series of string data. This lets us manipulate every single string element in the Series, all at once. The process returns a new Series containing the manipulated strings.

df["column_name"].str.str_func()
>> produce["Crop"].str.startswith("S")



We can use many functions with .str:

- .split("delim")
- .contains("val")
- .startswith("val")
- .slice(start, end)

[start:end]

A short(ish) list of important Pandas methods:

df.head(n=5) - gives the first n rows of the DataFrame
df.tail(n=5) - gives the last n rows of the DataFrame

df.shape - gives the dimensions of the DataFrame

df.rename(index, columns, axis) - renames the rows/columns of
the DataFrame

df.set_index() - sets the index to the specified column

df.reset_index() - resets the index to the default 0, 1, 2...

df.drop(labels, axis=0) - removes the specified rows/cols from
the DataFrame

df.sort_values(by, ascending=True) - sorts rows by the specified
column

df.isna() - checks if values in the DataFrame are NaN

df.index - returns the index of the DataFrame

df.columns - returns an array of the column labels

df.copy() - creates a copy of the DataFrame

df.value_counts() - summarizes the count of each column combo

SQL

- SELECT <column list> select columns in <column list> to keep
 a. [DISTINCT] keep only distinct rows (filter out duplicates)
 - b. Aggregation function (when using groupby): COUNT, SUM, AVG FROM <table1> which table are we drawing data **from**
- 2. FROM (Labiel) Which table are we drawing data **nom**
- 4. [GROUP BY <column list>] **group** together rows **by** value of columns in <column list>
- 5. [HAVING <predicate>] only keep groups having <predicate> satisfied
- 6. [ORDER BY <column list> [DESC/ASC]] order the output by value of the columns in <column list>, ASCending by default
- 7. [LIMIT <amount>] limit the output to just the first <amount> rows

Regular Expressions

)	Operator	Description
	•	Matches any character except \n
	\\	Escapes metacharacters
		Matches expression on either side of expression; has lowest priority of any operator
	\d, \w, \s	Predefined character group of digits (0-9), alphanumerics (a-z, A-Z, 0-9, and underscore), or whitespace, respectively
	*	Matches preceding character/group zero or more times
	?	Matches preceding character/group zero or one times
	+	Matches preceding character/group one or more times
	^, \$	Matches the beginning and end of the line, respectively
	()	Capturing group used to create a sub-expression
	[]	Character class used to match any of the specified characters or range (e.g. [abcde] is equivalent to [a-e])
	[^]	Invert character class; e.g. [^a-c] matches all characters except a, b, c

Visualization

Function	Description
plt.plot(x, y)	Creates a line plot of x against y
plt.scatter(x, y)	Creates a scatter plot of x against y
plt.hist(x, bins=None)	Creates a histogram of x; bi
plt.bar(x, height)	Creates a bar plot of c · ·

Function	Description
sns.countplot(data, x)	Create a barplot of value counts of variable x from data
<pre>sns.histplot(data, x, kde=False) sns.displot(x, data, rug = True, kde = True)</pre>	Creates a histogram of x from data; optionally overlay a kernel density estimator. displot is similar but can optionally overlay a rug plot.
<pre>sns.boxplot(data, x=None, y) sns.violinplot(data, x=None, y)</pre>	Create a boxplot of y, optionally factoring by categorical x, from data. violinplot is similar but also draws a kernel density estimator of y.
<pre>sns.scatterplot(data, x, y)</pre>	Create a scatterplot of x versus y from data
<pre>sns.lmplot(x, y, data, fit_reg=True)</pre>	Create a scatterplot of $\mathbf x$ versus $\mathbf y$ from data , and by default overlay a least-squares regression line
<pre>sns.jointplot(x, y, data, kind)</pre>	Combine a bivariate scatterplot of x versus y from data, with univariate density plots of each variable overlaid on the axes; kind determines the visualization type for the distribution plot, can be scatter, kde or hist