# Fall 2023 Data C100/C200 Midterm Reference Sheet

## Pandas

Suppose `df` is a DataFrame; `s` is a Series. `import pandas as pd`

| Function | Description |
|---|---|
| `df[col]` | Returns the column labeled `col` from `df` as a Series. |
| `df[[col1, col2]]` | Returns a DataFrame containing the columns labeled `col1` and `col2`. |
| `s.loc[rows] / df.loc[rows, cols]` | Returns a Series/DataFrame with rows (and columns) selected by their index values. |
| `s.iloc[rows] / df.iloc[rows, cols]` | Returns a Series/DataFrame with rows (and columns) selected by their positions. |
| `s.isnull() / df.isnull()` | Returns boolean Series/DataFrame identifying missing values |
| `s.fillna(value) / df.fillna(value)` | Returns a Series/DataFrame where missing values are replaced by `value` |
| `s.isin(values) / df.isin(values)` | Returns a Series/DataFrame of booleans indicating if each element is in `values`. |
| `df.drop(labels, axis)` | Returns a DataFrame without the rows or columns named `labels` along `axis` (either 0 or 1) |
| `df.rename(index=None, columns=None)` | Returns a DataFrame with renamed columns from a dictionary `index` and/or `columns` |
| `df.sort_values(by, ascending=True)` | Returns a DataFrame where rows are sorted by the values in columns `by` |
| `s.sort_values(ascending=True)` | Returns a sorted Series. |
| `s.unique()` | Returns a NumPy array of the unique values |
| `s.value_counts()` | Returns the number of times each unique value appears in a Series |
| `pd.merge(left, right, how='inner', on='a')` | Returns a DataFrame joining DataFrames `left` and `right` on the column labeled `a`; the join is of type `inner` |
| `left.merge(right, left_on=col1, right_on=col2)` | Returns a DataFrame joining DataFrames `left` and `right` on columns labeled `col1` and `col2`. |
| `df.pivot_table(index, columns, values=None, aggfunc='mean', fill_value=None)` | Returns a DataFrame pivot table where columns are unique values from `columns` (column name or list), and rows are unique values from `index` (column name or list); cells are collected `values` using `aggfunc`. If `values` is not provided, cells are collected for each remaining column with multi-level column indexing. If a `fill_value` is provided, any `NaN` values will be replaced with that `fill_value`. |
| `df.set_index(col)` | Returns a DataFrame that uses the values in the column labeled `col` as the row index. |
| `df.reset_index()` | Returns a DataFrame that has row index 0, 1, etc., and adds the current index as a column. |

Let `grouped = df.groupby(by)` where `by` can be a column label or a list of labels.

| Function | Description |
|---|---|
| `grouped.count()` | Return a DataFrame containing the size of each group, excluding missing values |
| `grouped.size()` | Return a Series containing size of each group, including missing values |
| `grouped.mean()`/`grouped.min()`/`grouped.max()` | Return a Series/DataFrame containing mean/min/max of each group for each column, excluding missing values |
| `grouped.filter(f)`<br>`grouped.agg(f)` | Filters or aggregates using the given function f |

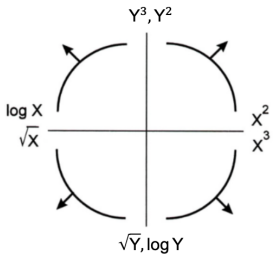
| Function | Description |
|---|---|
| `s.str.len()` | Returns a Series containing length of each string |
| `s.str[a:b]` | Returns a Series where each element is a slice of the corresponding string indexed from `a` (inclusive, optional) to `b` (non-inclusive, optional) |
| `s.str.lower()`/`s.str.upper()` | Returns a Series of lowercase/uppercase versions of each string |
| `s.str.replace(pat, repl)` | Returns a Series that replaces occurences of substrings matching the regex `pat` with string `repl` |
| `s.str.contains(pat)` | Returns a boolean Series indicating if a substring matching the regex `pat` is contained in each string |
| `s.str.extract(pat)` | Returns a Series of the first subsequence of each string that matches the regex `pat`. If `pat` contains one group, then only the substring matching the group is extracted |

## Visualization

Matplotlib: `x` and `y` are sequences of values. `import matplotlib.pyplot as plt`

| 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`; `bins` can be an integer or a sequence |
| `plt.bar(x, height)` | Creates a bar plot of categories `x` and corresponding heights `height` |

Tukey-Mosteller Bulge Diagram.




Seaborn: `x` and `y` are column names in a DataFrame `data`. `import seaborn as sns`

| Function | Description |
|---|---|
| `sns.countplot(data, x)` | Create a barplot of value counts of variable `x` from `data` |
| `sns.histplot(data, x, stat='count', kde=False)`<br>`sns.displot(data, x, kind='hist', rug=False, kde=False)` | Creates a histogram of `x` from `data`, where bin statistics `stat` is one of `'count'`, `'frequency'`, `'probability'`, `'percent'`, and `'density'`; optionally overlay a kernel density estimator.<br>`displot` is similar but can optionally overlay a rug plot and/or a KDE plot |
| `sns.boxplot(data, x=None, y)`<br>`sns.violinplot(data, x=None, y)` | Create a boxplot of `y`, optionally factoring by categorical `x`, from `data`. `violinplot` is similar but also draws a kernel density estimator of `y` |
| `sns.rugplot(data, x)` | Adds a rug plot on the x-axis of variable `x` from `data` |
| `sns.scatterplot(data, x, y)` | Create a scatterplot of `x` versus `y` from `data` |
| `sns.lmplot(x, y, data, fit_reg=True)` | Create a scatterplot of `x` versus `y` from `data`, and by default overlay a least-squares regression line |
| `sns.jointplot(x, y, data, kind)` | 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` |

## Regular Expressions

| Operator | Description | Operator | Description |
|---|---|---|---|
| `.` | Matches any character except `\n` | `*` | Matches preceding character/group zero or more times |
| `\` | Escapes metacharacters | `?` | Matches preceding character/group zero or one times |
| `\|` | Matches expression on either side of expression; has lowest priority of any operator | `+` | Matches preceding character/group one or more times |
| `\d`, `\w`, `\s` | Predefined character group of digits (0-9), alphanumerics (a-z, A-Z, 0-9, and underscore), or whitespace, respectively | `^`, `$` | Matches the beginning and end of the line, respectively |
| `\D`, `\W`, `\S` | Inverse sets of `\d`, `\w`, `\s`, respectively | `( )` | Capturing group used to create a sub-expression |
| `{m}` | Matches preceding character/group exactly `m` times | `[ ]` | Character class used to match any of the specified characters or range (e.g. `[abcde]` is equivalent to `[a-e]`) |
| `{m, n}` | Matches preceding character/group at least `m` times and at most `n` times. If either `m` or `n` are omitted, set lower/upper bounds to 0 and ∞, respectively | `[^ ]` | Invert character class; e.g. `[^a-c]` matches all characters except `a`, `b`, `c` |

Modified lecture example for capture groups:

```
import re
lines = '169.237.46.168 - - [26/Jan/2014:10:47:58 -0800] "GET ... HTTP/1.1"'
re.findall(r'\[\d+\/(\w+)\/\d+:\d+:\d+:\d+ .+\]', line) # returns ['Jan']
```

| Function | Description |
|---|---|
| `re.match(pattern, string)` | Returns a match if zero or more characters at beginning of `string` matches `pattern`, else None |
| `re.search(pattern, string)` | Returns a match if zero or more characters anywhere in `string` matches `pattern`, else None |
| `re.findall(pattern, string)` | Returns a list of all non-overlapping matches of `pattern` in `string` (if none, returns empty list) |
| `re.sub(pattern, repl, string)` | Returns `string` that replaces all occurrences of `pattern` with `repl` |

## Modeling

| Concept | Formula | Concept | Formula |
|---|---|---|---|
| Variance, $\sigma_x^2$ | $\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2$ | Correlation $r$ | $r=\frac{1}{n}\sum_{i=1}^{n}\frac{x_i-\bar{x}}{\sigma_x}\frac{y_i-\bar{y}}{\sigma_y}$ |
| $L_1$ loss | $L_1(y,\hat{y})=\mid y-\hat{y} \mid$ | Linear regression estimate of $y$ | $\hat{y}=\theta_0+\theta_1 x$ |
| $L_2$ loss | $L_2(y,\hat{y})=(y-\hat{y})^2$ | Least squares linear regression | $\hat{\theta}_0=\bar{y}-\hat{\theta}_1\bar{x}$ $\hat{\theta}_1=r\frac{\sigma_y}{\sigma_x}$ |
| Empirical risk with loss $L$ | $R(\theta)=\frac{1}{n}\sum_{i=1}^{n}L(y_i,\hat{y}_i)$ | | |

## Ordinary Least Squares

Multiple Linear Regression Model: $\hat{\mathbb{Y}}=\mathbb{X}\theta$ with design matrix $\mathbb{X}$, response vector $\mathbb{Y}$, and predicted vector $\hat{\mathbb{Y}}$. If there are $p$ features plus a bias/intercept, then the vector of parameters $\theta=[\theta_0,\theta_1,\ldots,\theta_p]^T\in\mathbb{R}^{p+1}$. The vector of estimates $\hat{\theta}$ is obtained from fitting the model to the sample $(\mathbb{X},\mathbb{Y})$.

| Concept | Formula | Concept | Formula |
|---|---|---|---|
| Mean squared error | $R(\theta)=\frac{1}{n}\|\mathbb{Y}-\mathbb{X}\theta\|_2^2$ | Normal equation | $\mathbb{X}^T\mathbb{X}\hat{\theta}=\mathbb{X}^T\mathbb{Y}$ |
| | | Least squares estimate, if $\mathbb{X}$ is full rank | $\hat{\theta}=(\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y}$ |
| Residual vector, $e$ | $e=\mathbb{Y}-\hat{\mathbb{Y}}$ | Multiple $R^2$ (coefficient of determination) | $R^2=\frac{\text{variance of fitted values}}{\text{variance of } y}$ |