Summer 2023 Data C100 Midterm Reference Sheet

Pandas

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

Function	Description
`pd.read_csv(filepath, delimiter)`	Loads the CSV at `filepath` into a DataFrame with records separated by `delimiter` (`','`, `'\t'`, etc.)
`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, sorted in descending order of count.
`pd.merge(left, right, how='inner', on='a')`	Returns a DataFrame joining `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 `left` and `right` on columns labeled `col1` and `col2`.
`df.pivot_table(index, columns, values=None, aggfunc='mean')`	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.
`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 Series containing the size of each group, excluding missing values
`grouped.size()`	Return a Series containing size of each group, including missing values
`grouped.mean()/.min()/.max()`	Return a Series/DataFrame containing mean/min/max of each group for each column, excluding missing values
`grouped.filter(f)` `grouped.agg(f)`	Filters or aggregates using the given function `f`

Text Wrangling and Regular Expressions

Pandas `str` methods

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
`s.str.split(pat)`	Splits the strings in `s` at the delimiter `pat`. Returns a Series of lists, where each list contains strings of the characters before and after the split.

Regex patterns

Operator	Description	Operator	Description
`.`	Matches any character except `\n`	**`*`**	Matches preceding character/group zero or more times
`\`	Escapes metacharacters	**`+`**	Matches preceding character/group one or more times
`|`	Matches expression on either side of expression; has lowest priority of any operator	**`^`**	Matches the beginning of the string
`\d`, **`\w`**, **`\s`**	Predefined character group of digits (0-9), alphanumerics (a-z, A-Z, 0-9, and underscore), or whitespace, respectively	**`$`**	Matches the end of the string
`\D`, **`\W`**, **`\S`**	Inverse sets of **`\d`**, **`\w`**, **`\s`**, respectively	**`( )`**	Capturing group or 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`

Python `re` methods

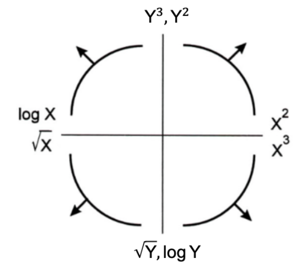
Function	Description
`re.match(pattern, string)`	Returns all matching characters if zero or more characters at beginning of `string` matches `pattern`, else None
`re.search(pattern, string)`	Returns all matching characters if zero or more characters anywhere in `string` matches `pattern`, else None
`re.findall(pattern, string)`	Returns a list of all non-overlapping matches of `pattern` in `string` (if none, returns empty list). If `pattern` includes capturing groups, only return captured characters.
`re.sub(pattern, repl, string)`	Returns `string` after replacing all occurrences of `pattern` with `repl`

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 keyword arguments assigned to string column names in a DataFrame `data`. `import seaborn as sns`

Function	Description
`sns.countplot(data=None, x=None)`	Create a barplot of value counts of variable x from `data`
`sns.histplot(data=None, x=None, stat='count', kde=False)` `sns.displot(data=None, x=None, stat='count', rug=True, kde=True)`	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.
`sns.boxplot(data=None, x=None, y=None)` `sns.violinplot(data=None, x=None, y=None)`	Create a boxplot of a numeric feature (e.g., y), optionally factoring by a category (e.g., x), from `data`. `violinplot` is similar but also draws a kernel density estimator of the numeric feature
`sns.scatterplot(data=None, x=None, y=None)`	Create a scatterplot of x versus y from `data`
`sns.lmplot(data=None, x=None, y=None, fit_reg=True)`	Create a scatterplot of x versus y from `data`, and by default overlay a least-squares regression line
`sns.jointplot(data=None, x=None, y=None, 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`
`sns.kdeplot(data=None, x=None)`	Create a kernel density estimate (KDE) of the distribution of x from `data`

Modeling

Concept	Formula	Concept	Formula
Variance, σ_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	$\hat{\theta}_0 = \bar{y} - \hat{\theta}_1\bar{x} \qquad \hat{\theta}_1 = r\frac{\sigma_y}{\sigma_x}$
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 $\theta = [\theta_0, \theta_1, \dots, \theta_p]^T \in \mathbb{R}^{p+1}$. 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)	$R^2 = \frac{\text{variance of fitted values}}{\text{variance of } y}$

Scikit-Learn

`import sklearn.linear_model`

Function(s)	Description
`LinearRegression(fit_intercept=True)`	Returns an ordinary least squares Linear Regression model.
`model.fit(X, y)`	Fits the scikit-learn `model` to the provided `X` and `y`.
`model.predict(X)`	Returns predictions for the `X` passed in according to the fitted `model`.
`model.coef_`	Estimated coefficients for the linear model, not including the intercept term.
`model.intercept_`	Bias/intercept term of the linear model. Set to 0.0 if `fit_intercept=False`.

Gradient Descent

Let L be an objective function to minimize with respect to θ; assume that some optimal parameter vector $\hat{\theta}$ exists. Suppose $\theta^{(0)}$ is some starting estimate at $t = 0$, and $\theta^{(t)}$ is the estimate at step t. Then for a learning rate α, the gradient update step to compute $\theta^{(t+1)}$ is:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta L$$

where $\nabla_\theta L$ is the partial derivative/gradient of L with respect to θ, evaluated at $\theta^{(t)}$.