Fall 2025 Data C100/C200 Final Reference Sheet

Pandas

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

Function	Description
`df.shape`	Returns a tuple containing the number of rows and columns, in that order
`df.index`	Returns the index (row labels) of `df` as an Index object
`df[col]`	Returns the column labeled `col` from `df` as a Series
`df.index[i]`	Returns the row label at position `i` from `df`'s index
`df[[col1, col2]]`	Returns a DataFrame containing the columns labeled `col1` and `col2`
`s.idxmax()`	Returns the index label of the first occurrence of the maximum value in Series `s`
`s.astype(dtype)`	Returns a Series casted to the specified type `dtype`
`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 of `s` in the order that they appear
`s.value_counts()`	Returns the number of times each unique value appears in a Series
`pd.merge(left, right, how='inner', left_on=col1, right_on=col2)`	Returns a DataFrame joining left and right on columns labeled col1 and col2; 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(values=None, index=None, columns=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.
`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()`/`.min()`/`.max()`	Return a Series/DataFrame containing mean/min/max of each group for each column, excluding missing values
`grouped.head(n)`/`.tail(n)`	Return a Series/DataFrame containing first/last `n` entries of each group for each column, excluding missing values
`grouped.filter(f)` `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, regex=False)`	Returns a Series that replaces occurences of substrings matching `pat` with string `repl`. When `regex=False`, `pat` is treated as a literal string; when `regex=True`, `pat` is treated as a RegEx pattern.
`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 DataFrame 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` (defaults to a whitespace). Returns a Series of lists, where each list contains strings of the characters before and after the split.

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`

Seaborn: `x` and `y` are 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, kind='hist', rug=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. `displot` is similar but can optionally overlay a rug plot and/or a KDE plot
`sns.rugplot(data=None, x=None)`	Adds a rug plot on the x-axis of variable `x` from `data`
`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='scatter')`	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'\[\d+\/(\w+)\/\d+:\d+:\d+:\d+ .+\]', lines) # 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` after replacing all occurrences of `pattern` with `repl`

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	$R(\theta)=\frac{1}{n}\sum_{i=1}^{n}L(y_i,\hat{y}_i)$		

Multiple Linear Regression Formulas

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	$\hat{\theta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y}$	Multiple R^2 (coefficient of determination)	$R^2 = \frac{\text{variance of fitted values}}{\text{variance of } y}$
Ridge Regression L2 Regularization	$\frac{1}{n}\|\|\mathbb{Y} - \mathbb{X}\theta\|\|_2^2 + \lambda\|\|\theta\|\|_2^2$	Squared L2 Norm of $\theta \in \mathbb{R}^d$	$\|\|\theta\|\|_2^2 = \sum_{j=1}^d \theta_j^2$
Ridge regression estimate (closed form)	$\hat{\theta}_{\text{ridge}} = (\mathbb{X}^T\mathbb{X} + n\lambda I)^{-1}\mathbb{X}^T\mathbb{Y}$	L1 Norm of $\theta \in \mathbb{R}^d$	$\|\|\theta\|\|_1 = \sum_{j=1}^d \|\theta_j\|$
LASSO Regression L1 Regularization	$\frac{1}{n}\|\|\mathbb{Y} - \mathbb{X}\theta\|\|_2^2 + \lambda\|\|\theta\|\|_1$		

Scikit-Learn

Package: `sklearn.linear_model`

Linear Regression	Logistic Regression	Function(s)	Description
✓	-	`LinearRegression(fit_intercept=True)`	Returns an ordinary least squares Linear Regression model.
-	✓	`LogisticRegression( fit_intercept=True, penalty='l2', C=1.0)`	Returns an ordinary least squares Linear Regression model. Hyperparameter C is inverse of regularization parameter, C = 1/λ.
✓	-	`LassoCV()`, `RidgeCV()`	Returns a Lasso (L1 Regularization) or Ridge (L2 regularization) linear model, respectively, and picks the best model by cross validation.
✓	✓	`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.predict_proba(X)`	Returns predicted probabilities for `X` according to the fitted `model`. If binary classes, will return probabilities for both class 0 and 1.
✓	✓	`model.coef_`	Estimated coefficients for the linear model, excluding the intercept.
✓	✓	`model.intercept_`	Bias/intercept term of the linear model. Set to 0.0 if `fit_intercept=False`.

Package: `sklearn.model_selection`

		Function	Description
		`train_test_split(*arrays, test_size=0.2)`	Returns two random subsets of each array passed in, with 0.8 of the array in the first subset and 0.2 in the second subset.

Probability

Let X have a discrete probability distribution $P(X = x)$. X has expectation $\mathbb{E}[X] = \sum_x xP(X = x)$ over all possible values x, variance $\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$, and standard deviation $\mathrm{SD}(X) = \sqrt{\mathrm{Var}(X)}$.

Notes	Property of Expectation	Property of Variance
X is a random variable.	$\mathbb{E}[X] = \sum_x xP(X = x)$	$\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = E[X^2] - (E[X])^2$
X is a random variable, $a, b \in \mathbb{R}$ are scalars.	$\mathbb{E}[aX + b] = a\mathbb{E}[X] + b$	$\mathrm{Var}(aX + b) = a^2\mathrm{Var}(X)$
X, Y are random variables.	$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$	$\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\mathrm{Cov}(X, Y)$
X is a Bernoulli random variable that takes the value 1 with probability p, and 0 otherwise.	$\mathbb{E}[X] = p$	$\mathrm{Var}(X) = p(1 - p)$

Parameter Estimation and Gradient Descent Update Rule

Parameter Estimation

Suppose for each individual with fixed input x, we observe a random response $Y = g(x) + \epsilon$, where g is the true relationship and ϵ is random noise with zero mean and variance σ^2.

For a new individual with fixed input x, define our random prediction $\hat{Y}(x)$ based on a model fit to our observed sample $(\mathbb{X}, \mathbb{Y})$. The model risk is the mean squared prediction error between Y and $\hat{Y}(x)$: $\mathbb{E}[(Y - \hat{Y}(x))^2] = \sigma^2 + \left(\mathbb{E}[\hat{Y}(x)] - g(x)\right)^2 + \mathrm{Var}(\hat{Y}(x))$.

Suppose that input x has p features and the true relationship g is linear with parameter $\theta \in \mathbb{R}^{p+1}$. Then $Y = f(x) = \theta_0 + \sum_{j=1}^p \theta_j x_j + \epsilon$ and $\hat{Y} = \hat{f}(x)$ for an estimate $\hat{\theta}$ fit to the observed sample $(\mathbb{X}, \mathbb{Y})$.

Gradient Descent

For a learning rate α, the gradient update step is:

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

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

SQL

SQL syntax:

```
SELECT [DISTINCT]
    {* | expr [[AS] c_alias]
    {, expr [[AS] c_alias] ...}}
FROM tableref {, tableref}
[[INNER | LEFT ] JOIN table_name
    ON qualification_list]
[WHERE search_condition]
[GROUP BY colname {, colname...}]
[HAVING search_condition]
[ORDER BY column_list]
[LIMIT number]
[OFFSET number of rows];
```

Strings in SQL should use single quotes. Column names that contain a space should use double quotes.

Syntax	Description
`SELECT column_expression_list`	List is comma-separated. Column expressions may include aggregation functions (`MAX`, `FIRST`, `COUNT`, `AVG`, etc). `AS` renames columns. `DISTINCT` selects only unique rows.
`FROM s INNER JOIN t ON cond`	Inner join tables `s` and `t` using `cond` to filter rows; the `INNER` keyword is optional.
`FROM s LEFT JOIN t ON cond`	Left outer join of tables `s` and `t` using `cond` to filter rows.
`FROM s, t`	Cross join of tables `s` and `t`: all pairs of a row from `s` and a row from `t`
`WHERE a IN cons_list`	Select rows for which the value in column `a` is among the values in a `cons_list`.
`ORDER BY RANDOM() LIMIT n`	Draw a simple random sample of `n` rows.
`ORDER BY a, b DESC`	Order by column `a` (ascending by default) , then `b` (descending).
`CASE WHEN pred THEN cons ELSE alt END`	Evaluates to `cons` if `pred` is true and `alt` otherwise. Multiple `WHEN`/`THEN` pairs can be included, and `ELSE` is optional.
`WHERE s.a LIKE 'p'`	Matches each entry in the column `a` of table `s` to the text pattern `p`. The wildcard `%` matches at least zero characters.
`LIMIT number`	Keep only the first `number` rows in the return result.
`OFFSET number`	Skip the first `number` rows in the return result.

Logistic Regression and Classification

Logistic Regression Model: For input feature vector x, $\hat{P}_\theta(Y = 1|x) = \sigma(x^T\theta)$, where $\sigma(z) = 1/(1 + e^{-z})$. For a single datapoint, define cross-entropy loss as $-[y\log(p) + (1-y)\log(1-p)]$, where p is the probability that the response is 1.

An ROC curve has the false positive rate (FPR) on the x-axis and true positive rate (TPR) on the y-axis.

Classification Performance

Suppose you predict n datapoints.

Metric	Formula
Accuracy	$\frac{TP+TN}{n}$
Precision	$\frac{TP}{TP+FP}$
Recall, True Positive Rate (TPR)	$\frac{TP}{TP+FN}$
False Positive Rate (FPR)	$\frac{FP}{FP+TN}$
F1 Score	$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Clustering

A datapoint's **silhouette score** S is defined as $S = (B - A)/\max(A, B)$, where A is the mean distance to other points in its cluster, and B is the mean distance to points in its closest cluster.