Summer 2022 Data C100 Final Reference Sheet

Pandas

Suppose `df` is a DataFrame; `s` is a Series. `pd` is the Pandas package.

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`
`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')`	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()`/`grouped.min()`/`grouped.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

Function	Description
`s.str.len()`	Returns a Series containing length of each string
`s.str.lower()`/`s.str.upper()`	Returns a Series containing lowercase/uppercase version of each string
`s.str.replace(pat, repl)`	Returns a Series after replacing occurences of substrings matching regular expression `pat` with string `repl`
`s.str.contains(pat)`	Returns a boolean Series indicating whether a substring matching the regular expression `pat` is contained in each string
`s.str.extract(pat)`	Returns a Series of the first subsequence of each string that matches the regular expression `pat`. If `pat` contains one group, then only the substring matching the group is extracted

Visualization

Matplotlib: `x` and `y` are sequences of values.

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`.

Function	Description
`sns.countplot(data, x)`	Create a barplot of value counts of variable `x` from `data`
`sns.histplot(data, x, kde=False)` `sns.displot(x, data, rug = True, kde = True)`	Creates a histogram of `x` from `data`; optionally overlay a kernel density estimator. `displot` is similar but can optionally overlay a rug plot.
`sns.boxplot(data, x=None, y)` `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.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

List of all metacharacters: `. ^ $ * + ? ] [ \ | ( ) { }`

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`

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`

Modified lecture example for a single capturing group:

```
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']
```

Modeling

Concept	Formula	Concept	Formula
L_1 loss	$L_1(y, \hat{y}) = \mid y - \hat{y} \mid$	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_2 loss	$L_2(y, \hat{y}) = (y - \hat{y})^2$	Linear regression prediction of y	$\hat{y} = a + bx$
Empirical risk with loss L	$R(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i)$	Least squares linear regression, slope $\hat{b}$	$\hat{b} = r\frac{\sigma_y}{\sigma_x}$
		Least squares linear regression, intercept $\hat{a}$	$\hat{a} = \bar{y} - \hat{b}\bar{x}$

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

Scikit-Learn

Suppose `sklearn.model_selection` and `sklearn.linear_model` are both imported packages.

Package	Function(s)	Description
`sklearn.linear_model`	`LinearRegression(fit_intercept=True)`	Returns an ordinary least squares Linear Regression model.
	`LassoCV(fit_intercept=True)`, `RidgeCV(fit_intercept=True)`	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.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`.
`sklearn.model_selection`	`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 $\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$, and standard deviation $\text{SD}(X) = \sqrt{\text{Var}(X)}$.

The covariance of two random variables X and Y is $\mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$. If X and Y are independent, then $\text{Cov}(X, Y) = 0$.

Notes	Property of Expectation	Property of Variance
X is a random variable.		$\text{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$
X is a random variable. $a, b \in \mathbb{R}$ are scalars.	$\mathbb{E}[aX + b] = a\mathbb{E}[X] + b$	$\text{Var}(aX + b) = a^2\text{Var}$
X, Y are random variables.	$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$	$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y)$
X is a Bernoulli random variable that takes on value 1 with probability p and 0 otherwise.	$\mathbb{E}[X] = p$	$\text{Var}(X) = p(1 - p)$
Y is a Binomial random variable representing the number of ones in n independent Bernoulli trials with probability p of 1.	$E[Y] = np$	$\text{Var}(Y) = np(1 - p)$

Central Limit Theorem

Let $(X_1, \ldots, X_n)$ be a sample of independent and identically distributed random variables drawn from a population with mean μ and standard deviation σ. The sample mean $\overline{X}_n = \sum_{i=1}^n X_i$ is normally distributed, where $\mathbb{E}[\overline{X}_n] = \mu$ and $\text{SD}(\overline{X}_n) = \sigma/\sqrt{n}$.

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 $\hat{Y}(x)$:

$$\mathbb{E}[(Y - \hat{Y}(x))^2] = \sigma^2 + \Big(\mathbb{E}[\hat{Y}(x)] - g(x)\Big)^2 + \text{Var}(\hat{Y}(x)).$$

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

Gradient Descent

Let $L(\theta, \mathbb{X}, \mathbb{Y})$ be an objective function to minimize over θ, with some optimal $\hat{\theta}$. 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

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

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

SQL

SQLite 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];
```

Syntax	Description
`SELECT column_expression_list`	List is comma-separated. Column expressions may include aggregation functions (`MAX`, `FIRST`, `COUNT`, 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.

Principal Component Analysis (PCA)

The i-th Principal Component of the matrix X is defined as the i-th column of $U\Sigma$ defined by Singular Value Decomposition (SVD).

$X = U\Sigma V^T$ is the SVD of X if U and V^T are orthonormal matrices and Σ is a diagonal matrix. The diagonal entries of Σ, $[s_1, \ldots, s_r, 0, \ldots, 0]$, are known as singular values of X, where $s_i > s_j$ for $i < j$ and $r = \text{rank}(X)$.

Define the design matrix $X \in \mathbb{R}^{n \times p}$. Define the total variance of X as the sum of individual variances of the p features. The amount of variance captured by the i-th principal component is equivalent to s_i^2/n, where n is the number of datapoints.

Logistic Regression and Classification

Logistic Regression Model: For input feature vector x, $\hat{P}_\theta(Y = 1|x) = \sigma(x^T\theta)$. The estimate $\hat{\theta}$ is the parameter θ that minimizes the average cross-entropy loss on training data. 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.

Logistic Regression Classifier: For a given input x and trained logistic regression model with parameter θ, compute $p = \hat{P}(Y = 1|x) = \sigma(x^T\theta)$. predict response $\hat{y}$ with classification threshold T as follows:

$$\hat{y} = \text{classify}(x) = \begin{cases} 1 & p \geq T \\ 0 & \text{otherwise} \end{cases}$$

Confusion Matrix

Columns are the predicted values $\hat{y}$ and rows are the actual classes y.

Classification Performance

Suppose you predict n datapoints.

	0	1
0	True negative (TN)	False Positive (FP)
1	False negative (FN)	True Positive (TP)

Metric	Formula	Other Names	Visualization	Plot
Accuracy	$\frac{TP+TN}{n}$		Precision-Recall Curve	Precision vs. Recall for different thresholds T
Precision	$\frac{TP}{TP+FP}$		ROC Curve	TPR vs. FPR for different thresholds T
Recall/TPR	$\frac{TP}{TP+FN}$	True Positive Rate, Sensitivity		
FPR	$\frac{FP}{FP+TN}$	False Positive Rate, Specificity		

Scikit-Learn

Suppose `linear_model` is an imported `sklearn` package.

Class/Attribute	Description	Function	Description
`linear_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.	`model.fit(X, y)`	Fits the scikit-learn `model` to the provided `X` and `y`.
`model.coef_`	Estimated coefficients for the model, not including the intercept term.	`model.predict_proba(X)`	Returns predicted probabilities for the `X` passed in according to the fitted `model`. If binary classes, will return probabilities for both class 0 and 1.
`model.intercept_`	Bias/intercept term of the model. Set to 0.0 if `fit_intercept=False`.	`model.predict(X)`	Returns predictions for the `X` passed in according to the fitted `model`.
		`model.score(X, y)`	Returns the average `model` accuracy on the given test data `X` and labels `y`.

Suppose `tree` and `ensemble` are imported `sklearn` packages.

Class/Function	Description
`tree.DecisionTreeClassifier(criterion='entropy', max_depth=None)`	Returns a decision tree model which uses `criterion` to measure the quality of a split. `max_depth` is the maximum depth of the tree; if `None`, then nodes are expanded until all leaves are pure.
`ensemble.RandomForestClassifier(n_estimators=100, criterion='entropy', max_depth=None)`	Fit `n_estimators` decision tree classifiers on sub-samples of the dataset.
`model.fit(X, y)`	Decision tree: Fit a decision tree `model` to the provided `X` and `y`. Random forest classifier: Build a forest `model` of decision trees fit to the provided `X` and `y`.
`model.predict(X)`	Decision tree: Returns predicted response for the `X` passed in according to the fitted `model`. Random forest classifier: Returns the predicted class by highest mean probability estimate according to the trees in the forest `model`.

Clustering

K-Means Clustering: Pick an arbitrary k, and randomly place k “centers”, each a different color. Then repeat until convergence:

1. Color points according to the closest center (defined as squared distance).
2. Move center for each color to center of points with that color.

K-Means minimizes inertia, defined as the sum of squared distances from each datapoint to its center.

Agglomerative Clustering: Assign each datapoint to its own cluster. Then, recursively merge pairs of clusters together until there are k clusters remaining.

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.

Decision Trees and Random Forests

Suppose you have a **decision tree classifier** for k classes. For each node, define the probability for class $C \in \{1, \ldots, k\}$ as $p_C = d_C/d$, where d_C is the number of datapoints in class C (of the d total in the node). Then the entropy of the node (in bits) is defined as $S = -\sum_C p_C \log_2 p_C$, and the weighted entropy of the node is its entropy scaled by the fraction of datapoints in that node.

Decision tree generation algorithm: All of the data starts in the root node. Repeat until every node is either pure or unsplittable:

- Pick the best feature x and best split value β, where β is picked to maximize the change in weighted entropy between the parent node and the child nodes.
- Split data into two nodes, one where x < β, and one where x ≥ β.

A node that has only one samples from one class is called a “pure” node. A node that has overlapping data points from different classes and thus that cannot be split is called “unsplittable”.

A **random forest** is a collection of many decision trees fit to variations of the same training data (e.g., bootstrapped samples, also called bagging; or random subsets of features). It is an ensemble method.