

Summer 2024 Data C100 Final Reference Sheet

Pandas

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

Function	Description
<code>df.shape</code>	Returns a tuple containing the number of rows and columns, in that order
<code>df.index</code>	Returns the index (row labels) of <code>df</code> as an Index object
<code>df[col]</code>	Returns the column labeled <code>col</code> from <code>df</code> as a Series
<code>df[[col1, col2]]</code>	Returns a DataFrame containing the columns labeled <code>col1</code> and <code>col2</code>
<code>s.astype(dtype)</code>	Returns a Series casted to the specified type <code>dtype</code>
<code>s.loc[rows] / df.loc[rows, cols]</code>	Returns a Series/DataFrame with rows (and columns) selected by their index values
<code>s.iloc[rows] / df.iloc[rows, cols]</code>	Returns a Series/DataFrame with rows (and columns) selected by their positions
<code>s.isnull() / df.isnull()</code>	Returns boolean Series/DataFrame identifying missing values
<code>s.fillna(value) / df.fillna(value)</code>	Returns a Series/DataFrame where missing values are replaced by <code>value</code>
<code>s.isin(values) / df.isin(values)</code>	Returns a Series/DataFrame of booleans indicating if each element is in <code>values</code> .
<code>df.drop(labels, axis)</code>	Returns a DataFrame without the rows or columns named <code>labels</code> along <code>axis</code> (either 0 or 1)
<code>df.rename(index=None, columns=None)</code>	Returns a DataFrame with renamed columns from a dictionary <code>index</code> and/or <code>columns</code>
<code>df.sort_values(by, ascending=True)</code>	Returns a DataFrame where rows are sorted by the values in columns <code>by</code>
<code>s.sort_values(ascending=True)</code>	Returns a sorted Series
<code>s.unique()</code>	Returns a NumPy array of the unique values of <code>s</code> in the order that they appear
<code>s.value_counts()</code>	Returns the number of times each unique value appears in a Series
<code>pd.merge(left, right, how='inner', left_on=col1, right_on=col2)</code>	Returns a DataFrame joining <code>left</code> and <code>right</code> on columns labeled <code>col1</code> and <code>col2</code> ; the join is of type <code>inner</code>
<code>left.merge(right, left_on=col1, right_on=col2)</code>	Returns a DataFrame joining <code>left</code> and <code>right</code> on columns labeled <code>col1</code> and <code>col2</code>
<code>df.pivot_table(values=None, index=None, columns=None, aggfunc='mean', fill_value=None)</code>	Returns a DataFrame pivot table where columns are unique values from <code>columns</code> (column name or list), and rows are unique values from <code>index</code> (column name or list); cells are collected <code>values</code> using <code>aggfunc</code> . If <code>values</code> is not provided, cells are collected for each remaining column with multi-level column indexing.
<code>df.set_index(col)</code>	Returns a DataFrame that uses the values in the column labeled <code>col</code> as the row index
<code>df.reset_index()</code>	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
<code>grouped.count()</code>	Return a DataFrame containing the size of each group, excluding missing values
<code>grouped.size()</code>	Return a Series containing size of each group, including missing values
<code>grouped.mean().min().max()</code>	Return a Series/DataFrame containing mean/min/max of each group for each column, excluding missing values
<code>grouped.first().last()</code>	Return a Series/DataFrame containing first/last entry of each group for each column, excluding missing values
<code>grouped.filter(f)</code> <code>grouped.agg(f)</code>	Filters or aggregates using the given function <code>f</code>

Function	Description
<code>s.str.len()</code>	Returns a Series containing length of each string.
<code>s.str[a:b]</code>	Returns a Series where each element is a slice of the corresponding string indexed from <code>a</code> (inclusive, optional) to <code>b</code> (non-inclusive, optional).

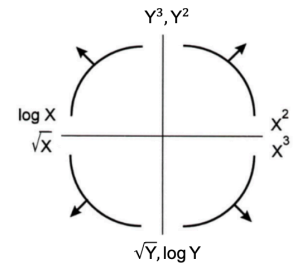
Function	Description
<code>s.str.lower()/s.str.upper()</code>	Returns a Series of lowercase/uppercase versions of each string.
<code>s.str.replace(pat, repl, regex=False)</code>	Returns a Series that replaces occurrences of substrings matching <code>pat</code> with string <code>repl</code> . When <code>regex=False</code> , <code>pat</code> is treated as a literal string; when <code>regex=True</code> , <code>pat</code> is treated as a RegEx pattern.
<code>s.str.contains(pat)</code>	Returns a boolean Series indicating if a substring matching the regex <code>pat</code> is contained in each string.
<code>s.str.extract(pat)</code>	Extracts capture groups in the regex <code>pat</code> from each string. Returns a DataFrame with the extracted substrings, where the columns in the returned DataFrame correspond to the capture groups in <code>pat</code> .
<code>s.str.split(pat=" ")</code>	Splits the strings in <code>s</code> at the delimiter <code>pat</code> (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
<code>plt.plot(x, y)</code>	Creates a line plot of <code>x</code> against <code>y</code>
<code>plt.scatter(x, y)</code>	Creates a scatter plot of <code>x</code> against <code>y</code>
<code>plt.hist(x, bins=None)</code>	Creates a histogram of <code>x</code> ; <code>bins</code> can be an integer or a sequence
<code>plt.bar(x, height)</code>	Creates a bar plot of categories <code>x</code> and corresponding heights <code>height</code>

Tukey-Mosteller Bulge Diagram.



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

Function	Description
<code>sns.countplot(data=None, x=None)</code>	Create a barplot of value counts of variable <code>x</code> from <code>data</code>
<code>sns.histplot(data=None, x=None, stat='count', kde=False)</code> <code>sns.displot(data=None, x=None, kind='hist', rug=False)</code>	Creates a histogram of <code>x</code> from <code>data</code> , where bin statistics <code>stat</code> is one of 'count', 'frequency', 'probability', 'percent', and 'density'; optionally overlay a kernel density estimator. <code>displot</code> is similar but can optionally overlay a rug plot and/or a KDE plot
<code>sns.rugplot(data=None, x=None)</code>	Adds a rug plot on the x-axis of variable <code>x</code> from <code>data</code>
<code>sns.boxplot(data=None, x=None, y=None)</code> <code>sns.violinplot(data=None, x=None, y=None)</code>	Create a boxplot of a numeric feature (e.g., <code>y</code>), optionally factoring by a category (e.g., <code>x</code>), from <code>data</code> . <code>violinplot</code> is similar but also draws a kernel density estimator of the numeric feature
<code>sns.scatterplot(data=None, x=None, y=None)</code>	Create a scatterplot of <code>x</code> versus <code>y</code> from <code>data</code>
<code>sns.lmplot(data=None, x=None, y=None, fit_reg=True)</code>	Create a scatterplot of <code>x</code> versus <code>y</code> from <code>data</code> , and by default overlay a least-squares regression line
<code>sns.jointplot(data=None, x=None, y=None, kind='scatter')</code>	Combine a bivariate scatterplot of <code>x</code> versus <code>y</code> from <code>data</code> , with univariate density plots of each variable overlaid on the axes; <code>kind</code> determines the visualization type for the distribution plot, can be <code>scatter</code> , <code>kde</code> or <code>hist</code>

Regular Expressions

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

Operator	Description	Operator	Description
{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+ .+', 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.
re.split(pattern, string)	Splits string on occurrences of pattern, returning a list of substrings.

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}) = y - \hat{y} $	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: $\hat{Y} = X\theta$ with design matrix X, response vector Y, and predicted vector \hat{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 (X, Y).

Concept	Formula	Concept	Formula
Mean squared error	$R(\theta) = \frac{1}{n} \ Y - X\theta\ _2^2$	Normal equation	$X^T X \hat{\theta} = X^T Y$
Least squares estimate, if X is full rank	$\hat{\theta} = (X^T X)^{-1} X^T Y$	Residual vector, e	$e = Y - \hat{Y}$
		Multiple R^2 (coefficient of determination)	$R^2 = \frac{\text{variance of fitted values}}{\text{variance of } y}$
Ridge Regression L2 Regularization	$\frac{1}{n} \ Y - X\theta\ _2^2 + \alpha \ \theta\ _2^2$	Squared L2 Norm of $\theta \in \mathbb{R}^d$	$\ \theta\ _2^2 = \sum_{j=1}^d \theta_j^2$
Ridge regression estimate (closed form)	$\hat{\theta}_{\text{ridge}} = (X^T X + n\alpha I)^{-1} X^T Y$		

Concept	Formula	Concept	Formula
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

Package: `sklearn.linear_model`

Linear Regression	Logistic Regression	Function(s)	Description
✓	-	<code>LinearRegression(fit_intercept=True)</code>	Returns an ordinary least squares Linear Regression model.
-	✓	<code>LogisticRegression(fit_intercept=True, penalty='l2', C=1.0)</code>	Returns an ordinary least squares Linear Regression model. Hyperparameter C is inverse of regularization parameter, C = 1/λ.
✓	-	<code>LassoCV()</code> , <code>RidgeCV()</code>	Returns a Lasso (L1 Regularization) or Ridge (L2 regularization) linear model, respectively, and picks the best model by cross validation.
✓	✓	<code>model.fit(X, y)</code>	Fits the scikit-learn <code>model</code> to the provided <code>x</code> and <code>y</code> .
✓	✓	<code>model.predict(X)</code>	Returns predictions for the <code>x</code> passed in according to the fitted <code>model</code> .
✓	✓	<code>model.predict_proba(X)</code>	Returns predicted probabilities for the <code>x</code> passed in according to the fitted <code>model</code> . If binary classes, will return probabilities for both class 0 and 1.
✓	✓	<code>model.coef_</code>	Estimated coefficients for the linear model, not including the intercept term.
✓	✓	<code>model.intercept_</code>	Bias/intercept term of the linear model. Set to 0.0 if <code>fit_intercept=False</code> .

Package: `sklearn.model_selection`

Function	Description
<code>train_test_split(*arrays, test_size=0.2)</code>	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 $\text{SD}(X) = \sqrt{\text{Var}(X)}$.

Notes	Property of Expectation	Property of Variance
X is a random variable.	$\mathbb{E}[X] = \sum_x xP(X = x)$	$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = E[X^2] - (E[X])^2$
X is a random variable, $a, b \in \mathbb{R}$ are scalars.	$\mathbb{E}[aX + b] = a\mathbb{E}[X] + b$	$\text{Var}(aX + b) = a^2\text{Var}(X)$
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.	$\mathbb{E}[X] = p$	$\text{Var}(X) = p(1 - p)$

Parameter Estimation and Gradient Descent

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

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

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];
```

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

$X = USV^T$ is the SVD of X if U and V^T are matrices with orthonormal columns and S is a diagonal matrix. The diagonal entries of S , $[s_1, \dots, s_r, 0, \dots, 0]$, 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.

Syntax	Description
<code>np.linalg.svd(X, full_matrices = True)</code>	SVD of X with shape (M, N) that returns u, s, vt , where s is a 1D array of X 's singular values. If <code>full_matrices=True</code> , u and vt have shapes (M, M) and (N, N) respectively; otherwise shapes are (M, K) and (K, N) , respectively, where $K = \min(M, N)$.

Classification and Logistic Regression

Confusion Matrix

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

	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$	True negative (TN)	False Positive (FP)
$y = 1$	False negative (FN)	True Positive (TP)

Classification Performance

Suppose you predict n datapoints.

Metric	Formula	Other Names
Accuracy	$\frac{TP+TN}{n}$	
Precision	$\frac{TP}{TP+FP}$	
Recall/TPR	$\frac{TP}{TP+FN}$	True Positive Rate, Sensitivity
FPR	$\frac{FP}{FP+TN}$	False Positive Rate, FPR = 1 - Specificity

An ROC curve visualizes TPR vs. FPR for different thresholds T .

Logistic Regression Model: For input feature vector x , $\hat{P}_\theta(Y = 1|x) = \sigma(x^T\theta)$, where the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$. 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.

Properties of the Sigmoid Function:

- **Reflection/Symmetry:** $1 - \sigma(t) = \frac{e^{-t}}{1+e^{-t}} = \sigma(-t)$
- **Inverse:** $t = \sigma^{-1}(p) = \log\left(\frac{p}{1-p}\right)$
- **Derivative:** $\frac{d}{dt} \sigma(t) = \sigma(t)(1 - \sigma(t)) = \sigma(t)\sigma(-t)$

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:

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

Decision Trees and Random Forests

Suppose you have a **decision tree classifier** for k classes. For each node, define the probability for class $C \in \{1, \dots, k\}$ as $p_C = \frac{d_C}{d}$, where d_C is the number of datapoints in class C (of the d total in the node). Then the entropy of the node is defined as:

$$S = - \sum_C p_C \log_2 p_C$$

The weighted entropy L of a split resulting in two nodes X and Y with N_1 and N_2 total samples each is given by:

$$L = \frac{N_1 S(X) + N_2 S(Y)}{N_1 + N_2}$$

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.

To evaluate a K-Means clustering, we minimize a loss function. Two common ones are:

- **Inertia:** the sum of squared distances from each datapoint to its center. It is defined as $\sum_{i=1}^N (x_i - C_k)^2$, where N is the total number of datapoints, x_i represents datapoint i , and C_k is x_i 's closest center.
- **Distortion:** the weighted sum of squared distances from each data point to its center. It is defined as $\sum_{k=1}^K \frac{1}{n} \sum_{i=1}^n (x_{k,i} - C_k)^2$, where K represents the total number of clusters. For each cluster k , we sum the squared distances from each datapoint $x_{k,i}$ to its

center C_k and divide it by the total number of datapoints in that cluster, denoted as n . We add up these weighted sums to obtain the final value.

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.