



Pandas Cheat Sheet

This cheat sheet—part of our Complete Guide to NumPy, pandas, and Data Visualization—offers a handy reference for essential pandas commands, focused on efficient data manipulation and analysis. Using examples from the Fortune 500 Companies Dataset, it covers key pandas operations such as reading and writing data, selecting and filtering DataFrame values, and performing common transformations.

You'll find easy-to-follow examples for grouping, sorting, and aggregating data, as well as calculating statistics like mean, correlation, and summary statistics. Whether you're cleaning datasets, analyzing trends, or visualizing data, this cheat sheet provides concise instructions to help you navigate pandas' powerful functionality.

Designed to be practical and actionable, this guide ensures you can quickly apply pandas' versatile data manipulation tools in your workflow.

[Download PDF](#)

Table of Contents



Importing Data

IMPORT, READ_CSV, READ_TABLE, READ_EXCEL, READ_SQL, READ_JSON, READ_HTML, CLIPBOARD, DATAFRAME



Exporting Data

TO_CSV, TO_EXCEL, TO_SQL, TO_JSON, TO_HTML, TO_CLIPBOARD



Create Test Objects

DATAFRAME, SERIES, INDEX



Working with DataFrames

DATAFRAME BASICS, DATAFRAME VALUES, LOC, ILOC, BOOLEAN MASKS, BOOLEAN OPERATORS, DATA EXPLORATION, ASSIGNING VALUES, BOOLEAN INDEXING

View & Inspect Data



Data Cleaning

COLUMNS, ISNULL, NOTNULL, DROPNA, FILLNA, ASTYPE, REPLACE, RENAME, SET_INDEX, FINDING CORRELATION, CONVERTING A COLUMN TO DATETIME



Filter, Sort, & Group By

COLUMNS, SORT_VALUES, GROUP BY, PIVOT_TABLE, APPLY



Join & Combine

APPEND, CONCAT, JOIN



Statistics

DESCRIBE, MEAN, CORR, COUNT, MAX, MIN, MEDIAN, STD



Importing Data



import pandas as pd

Import the library using its standard alias

READ_CSV

```
pd.read_csv(filename)
```

Reads from a CSV file

READ_TABLE

```
pd.read_table(filename)
```

Reads from a delimited text file (like TSV)

READ_EXCEL

```
pd.read_excel(filename)
```

Reads from an Excel file

READ_SQL

```
pd.read_sql(query, connection_object)
```

Reads from a SQL table/database

READ_JSON



Reads from a JSON formatted string, URL or file

READ_HTML

```
pd.read_html(url)
```

Parses an html URL, string or file and extracts tables to a list of dataframes

CLIPBOARD

```
pd.read_clipboard()
```

Reads the contents of your clipboard

DATAFRAME

```
pd.DataFrame(dict)
```

Reads from a dict; keys for columns names, values for data as lists



Exporting Data

TO_CSV

```
df.to_csv(filename)
```



TO_EXCEL

```
df.to_excel(filename)
```

Writes to an Excel file

TO_SQL

```
df.to_sql(table_name, connection_object)
```

Writes to a SQL table

TO_JSON

```
df.to_json(filename)
```

Writes to a file in JSON format

TO_HTML

```
df.to_html(filename)
```

Writes to an HTML table

TO_CLIPBOARD

```
df.to_clipboard()
```

Writes to the clipboard



DATAFRAME

```
pd.DataFrame(np.random.rand(20, 5))
```

5 columns and 20 rows of random floats

SERIES

```
pd.Series(my_list)
```

Creates a series from an existing list object

INDEX

```
df.index = pd.date_range('1900/1/30', periods=df.shape[0])
```

Adds a date index



Working with DataFrames

DATAFRAME BASICS

```
f500 = pd.read_csv('f500.csv', index_col=0)
```



```
col_types = f500.dtypes
```

Return the data type of each column in a **DataFrame**

```
dims = f500.shape
```

Return the dimensions of a **DataFrame**

SELECTING DATAFRAME VALUES

```
f500["rank"]
```

Select the **rank** column from **f500**

```
f500[["country", "rank"]]
```

Select the **country** and **rank** columns from **f500**

```
first_five = f500.head(5)
```

Select the first five rows from **f500**

LOC



```
bottom_companies = f500.loc["National Grid":"AutoNation",  
                             ["rank", "sector", "country"]]  
  
revenue_giants = f500.loc[["Apple", "Industrial & Commercial Bank of China",  
                           "China Construction Bank", "Agricultural Bank of C  
                           "revenues":"profit_change"]
```



Use `.loc[]` to select rows and columns from `f500` by label—rows are specified first, followed by columns. You can select individual rows/columns or multiple by passing a list, and label-based slicing includes both the start and end labels.

ILOC

```
third_row_first_col = f500.iloc[2, 0]
```

Select the third row, first column by integer location

```
second_row = f500.iloc[1]
```

Select the second row by integer location

BOOLEAN MASKS

```
rev_is_null = f500["revenue_change"].isnull()
```

Check for `null` values in the `revenue_change` column



Filtering using Boolean array

```
f500[f500["previous_rank"].notnull()]
```

Filter rows where `previous_rank` is not `null`

BOOLEAN OPERATORS

```
filter_big_rev_neg_profit = (f500["revenues"] > 100000) &  
                             (f500["profits"] < 0)
```

Create a Boolean filter for companies with `revenues` greater than 100,000 and `profits` less than 0

DATA EXPLORATION

```
revs = f500["revenues"]  
summary_stats = revs.describe()
```

Generate summary statistics for the `revenues` column in `f500`

```
country_freqs = f500["country"].value_counts()
```

Count the occurrences of each country in `f500`

ASSIGNING VALUES

```
top5_rank_revenue["year_founded"] = 0
```



```
f500.loc["Dow Chemical", "ceo"] = "Jim Fitterling"
```

Update the CEO of Dow Chemical to **Jim Fitterling**

BOOLEAN INDEXING

```
kr_bool = f500["country"] == "South Korea"  
top_5_kr = f500[kr_bool].head()
```

Filter rows for **South Korea** and display the top 5

```
f500.loc[f500["previous_rank"] == 0, "previous_rank"] = np.nan  
prev_rank_after = f500["previous_rank"].value_counts(dropna=False).head()
```

Replace **0** with **NaN** in the **previous_rank** column and shows the top 5 most common values



View & Inspect Data

FREQUENCY TABLE

```
Series.value_counts()
```

Generate a frequency table from a **Series** object



Generate a sorted frequency table from a **Series** object

HISTOGRAM

```
Series.plot.hist()  
plt.show()
```

Generate a histogram from a **Series** object

VERTICAL BAR PLOT

```
Series.plot.bar()  
plt.show()
```

Generate a vertical bar plot from a **Series** object

HORIZONTAL BAR PLOT

```
Series.plot.barh()  
plt.show()
```

Generate a horizontal bar plot from a **Series** object

LINE PLOT

```
DataFrame.plot.line(x='col_1', y='col_2')  
plt.show()
```

Generate a line plot from a **DataFrame** object



```
DataFrame.plot.scatter(x= col_1 , y= col_2 )  
plt.show()
```

Generate a scatter plot from a **DataFrame** object

HEAD

```
df.head(n)
```

First **n** rows of the DataFrame

TAIL

```
df.tail(n)
```

Last **n** rows of the DataFrame

SHAPE

```
df.shape()
```

Number of rows and columns

INFO

```
df.info()
```

Index, Datatype and Memory information

DESCRIBE



Summary statistics for numerical columns

VALUE_COUNTS

```
s.value_counts(dropna=False)
```

Views unique values and counts

APPLY

```
df.apply(pd.Series.value_counts)
```

Unique values and counts for all columns



Data Cleaning

COLUMNS

```
df.columns = ['a', 'b', 'c']
```

Renames columns

ISNULL

```
pd.isnull()
```



NOTNULL

```
pd.notnull()
```

Opposite of `pd.isnull()`

DROPNA

```
df.dropna()
```

Drops all rows that contain null values

```
df.dropna(axis=1)
```

Drops all columns that contain null values

```
df.dropna(axis=1, thresh=n)
```

Drops all rows have have less than `n` non-null values

FILLNA

```
df.fillna(x)
```

Replaces all null values with `x`

```
s.fillna(s.mean())
```



ASTYPE

```
s.astype(float)
```

Converts the datatype of the Series to `float`

REPLACE

```
s.replace(1, 'one')
```

Replaces all values equal to 1 with `one`

REPLACE

```
s.replace([1, 3], ['one', 'three'])
```

Replaces all 1 with 'one' and 3 with 'three'

RENAME

```
df.rename(columns=lambda x: x + 1)
```

Mass renaming of columns

```
df.rename(columns={'old_name': 'new_name'})
```

Selective renaming of columns



Mass renaming of index

SET_INDEX

```
df.set_index('column_one')
```

Selectively sets the index

FINDING CORRELATION

```
f500['revenues'].corr(f500[profits])
```

Calculate Pearson's r correlation between **revenues** and **profits**

```
f500.corr()
```

Calculate the Pearson's r correlation matrix between all columns of **f500**

```
f500.corr()[['revenues', 'profits', 'assets']]
```

Calculate the correlation matrix for **f500** and select the correlations for the **revenues** , **profits** , and **assets** columns

CONVERTING A COLUMN TO DATETIME

```
f500['founding_date'] = f500.to_datetime(f500['founding_date'])
```

Convert the **founding_date** column in **f500** to **datetime** format



Filter, Sort, & Group By

COLUMNS

```
df[df[col] > 0.5]
```

Rows where the `col` column is greater than `0.5`

```
df[(df[col] > 0.5) & (df[col] < 0.7)]
```

Rows where `0.7 > col > 0.5`

SORT_VALUES

```
df.sort_values(col1)
```

Sorts values by `col1` in ascending order

```
df.sort_values(col2, ascending=False)
```

Sorts values by `col2` in descending order

```
df.sort_values([col1, col2], ascending=[True, False])
```

Sorts values by `col1` in ascending order then `col2` in descending order



```
df.groupby(col1)
```

Returns a **groupby** object for values from one column

```
df.groupby([col1, col2])
```

Returns a **groupby** object values from multiple columns

```
df.groupby(col1)[col2].mean()
```

Returns the mean of the values in **col2**, grouped by the values in **col1** (mean can be replaced with almost any function from the statistics section)

PIVOT_TABLE

```
df.pivot_table(index=col1,  
                values=[col2, col3],  
                aggfunc=mean)
```

Creates a pivot table that groups by **col1** and calculates the mean of **col2** and **col3**

GROUPBY

```
df.groupby(col1).agg(np.mean)
```

Finds the average across all columns for every unique **col1** group

APPLY



Applies a function across each column

```
df.apply(np.max, axis=1)
```

Applies a function across each row



Join & Combine

APPEND

```
df1.append(df2)
```

Adds the rows in **df1** to the end of **df2** (number of columns should be identical)

CONCAT

```
pd.concat([df1, df2], axis=1)
```

Adds the columns in **df1** to the end of **df2** (number of rows should be identical)

JOIN

```
df1.join(df2, on=col1, how='inner')
```



Statistics

DESCRIBE

```
df.describe()
```

Summary statistics for numerical columns

MEAN

```
df.mean()
```

Returns the mean of all columns

CORR

```
df.corr()
```

Returns the correlation between columns in a DataFrame

COUNT

```
df.count()
```

Returns the number of non-null values in each DataFrame column



```
df.max()
```

Returns the highest value in each column

MIN

```
df.min()
```

Returns the lowest value in each column

MEDIAN

```
df.median()
```

Returns the median of each column

STD

```
df.std()
```

Returns the standard deviation of each column





Mike Levy

Mike is a life-long learner who is passionate about mathematics, coding, and teaching. When he's not sitting at the keyboard, he can be found in his garden or at a natural hot spring.

[Terms of Use](#)

[Privacy Policy](#)



About

[For Business](#)

[For Educators](#)

[About Dataquest](#)

[Learner Stories](#)

[Contact Us](#)

[Partnership Programs](#)

[Sitemap](#)

Career Paths

[Data Scientist](#)



Data Analyst R

Business Analyst Power BI

Business Analyst Tableau

Junior Data Analyst

Skill Paths

SQL Courses

AI Courses

Machine Learning Courses

Deep Learning Courses

Excel Courses

Statistics Courses

Explore

Course Catalog

Projects

Teaching Method

Project-first Learning

How to Learn Python

The Dataquest Download