

---

Here are the sections:

- Data Science Cheatsheets
- Data Science EBooks
- Data Science Question Bank
- Data Science Case Studies
- Data Science Portfolio
- Data Journalism Portfolio
- Downloadable Cheatsheets

## **Data Science Cheatsheets**

This section contains cheatsheets of basic concepts in data science that will be asked in interviews:

- SQL
- Statistics and Probability
- Mathematics
- Machine Learning Concepts
- Deep Learning Concepts
- Supervised Learning
- Unsupervised Learning
- Computer Vision
- Natural Language Processing
- Stanford Materials

## **Data Science EBooks**

This section contains books that I have read about data science and machine learning:

- Intro To Machine Learning with Python
- Machine Learning In Action
- Python Data Science Handbook
- Doing Data Science - Straight Talk From The Front Line
- Machine Learning For Finance
- Practical Statistics for Data Science
- A/B Testing

---

## Data Science Question Bank

This section contains sample questions that were asked in actual data science interviews:

- Data Interview Qs
- Data Science Prep
- Interview Query
- Analytics Vidhya
- Springboard
- Elite Data Science
- Workera
- 150 Essential Data Science Questions and Answers

## Data Science Case Studies

This section contains case study questions that concern designing machine learning systems to solve practical problems.

## Data Science Portfolio

This section contains portfolio of data science projects completed by me for academic, self learning, and hobby purposes.

For a more visually pleasant experience for browsing the portfolio, check out [jameskle.com/data-portfolio](https://jameskle.com/data-portfolio)

- 

## Recommendation Systems

- Transfer Rec: My ongoing research work that intersects deep learning and recommendation systems.
- Movie Recommendation: Designed 4 different models that recommend items on the MovieLens dataset.

*Tools: PyTorch, TensorBoard, Keras, Pandas, NumPy, SciPy, Matplotlib, Seaborn, Scikit-Learn, Surprise, Wordcloud*

-

---

## Machine Learning

- Trip Optimizer: Used XGBoost and evolutionary algorithms to optimize the travel time for taxi vehicles in New York City.
- Instacart Market Basket Analysis: Tackled the Instacart Market Basket Analysis challenge to predict which products will be in a user's next order.

*Tools: Pandas, NumPy, Matplotlib, XGBoost, Geopy, Scikit-Learn*

•

## Computer Vision

- Fashion Recommendation: Built a ResNet-based model that classifies and recommends fashion images in the DeepFashion database based on semantic similarity.
- Fashion Classification: Developed 4 different Convolutional Neural Networks that classify images in the Fashion MNIST dataset.
- Dog Breed Classification: Designed a Convolutional Neural Network that identifies dog breed.
- Road Segmentation: Implemented a Fully-Convolutional Network for semantic segmentation task in the Kitty Road Dataset.

*Tools: TensorFlow, Keras, Pandas, NumPy, Matplotlib, Scikit-Learn, TensorBoard*

•

## Natural Language Processing

- Classifying Tweets with Weights & Biases: Developed 3 different neural network models that classify tweets on a crowdsourced dataset in Figure Eight.

•

## Data Analysis and Visualization

- World Cup 2018 Team Analysis: Analysis and visualization of the FIFA 18 dataset to predict the best possible international squad lineups for 10 teams at the 2018 World Cup in Russia.

- 
- Spotify Artists Analysis: Analysis and visualization of musical styles from 50 different artists with a wide range of genres on Spotify.

*Tools: Pandas, NumPy, Matplotlib, Rspotify, httr, dplyr, tidyr, radarchart, ggplot2*

## **Data Journalism Portfolio**

This section contains portfolio of data journalism articles completed by me for freelance clients and self-learning purposes.

For a more visually pleasant experience for browsing the portfolio, check out [jameskle.com/data-journalism](https://jameskle.com/data-journalism)

- 

### **Statistics**

- The 10 Statistical Techniques Data Scientists Need to Master
- Logistic Regression Tutorial
- Decision Trees Tutorial
- Support Vector Machines Tutorial
- A Friendly Introduction to Data-Driven Marketing for Business Leaders

- 

### **Machine Learning**

- The 10 Algorithms Machine Learning Engineers Need to Know
- 12 Useful Things to Know About Machine Learning
- A Tour of The Top 10 Algorithms for Machine Learning Newbie
- The 10 Data Mining Techniques Data Scientists Need For Their Toolbox
- Clustering and Classification in E-Commerce
- The ABCs of Learning to Rank
- 6 Ways to Debug a Machine Learning Model

-

---

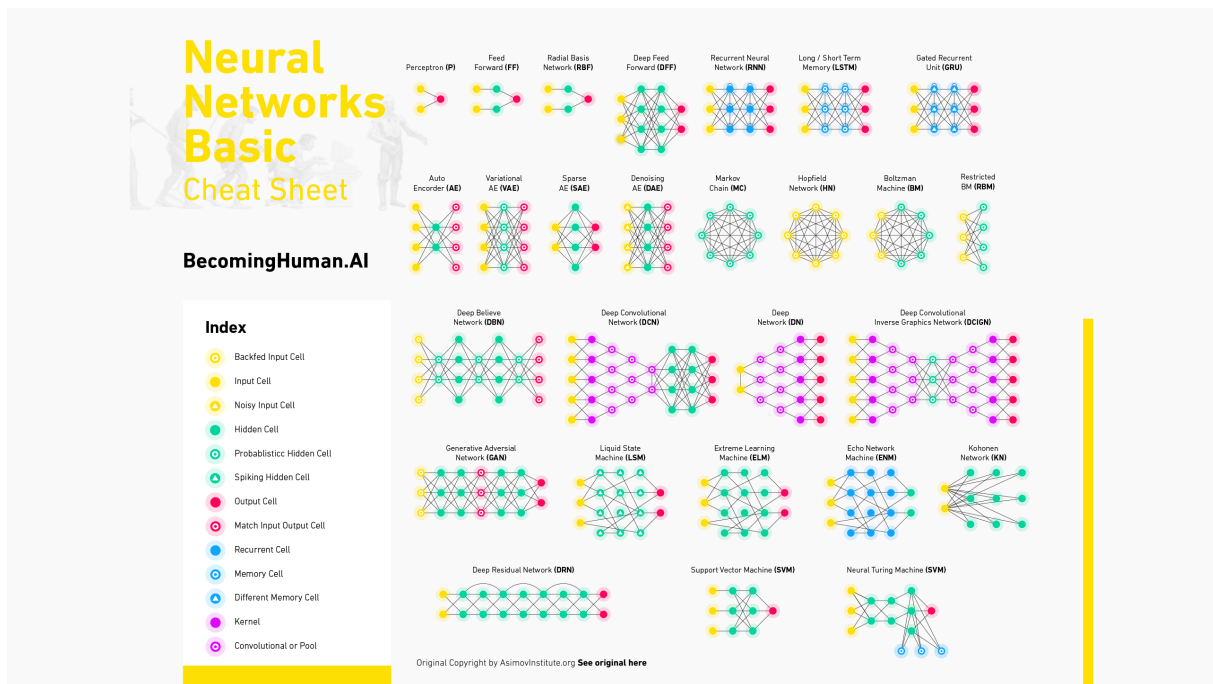
## **Deep Learning**

- The 10 Deep Learning Methods AI Practitioners Need to Apply
- The 8 Neural Network Architectures ML Researchers Need to Learn
- The 5 Deep Learning Frameworks Every Serious Machine Learner Should Be Familiar With
- The 5 Computer Vision Techniques That Will Change How You See The World
- Convolutional Neural Networks: The Biologically-Inspired Model
- Recurrent Neural Networks: The Powerhouse of Language Modeling
- The 7 NLP Techniques That Will Change How You Communicate in the Future
- The 5 Trends Dominating Computer Vision in 2018
- The 3 Deep Learning Frameworks For End-to-End Speech Recognition That Power Your Devices
- The 5 Algorithms for Efficient Deep Learning Inference on Small Devices
- The 4 Research Techniques to Train Deep Neural Network Models More Efficiently
- The 2 Hardware Architectures for Efficient Training and Inference of Deep Nets
- 10 Deep Learning Best Practices to Keep in Mind in 2020

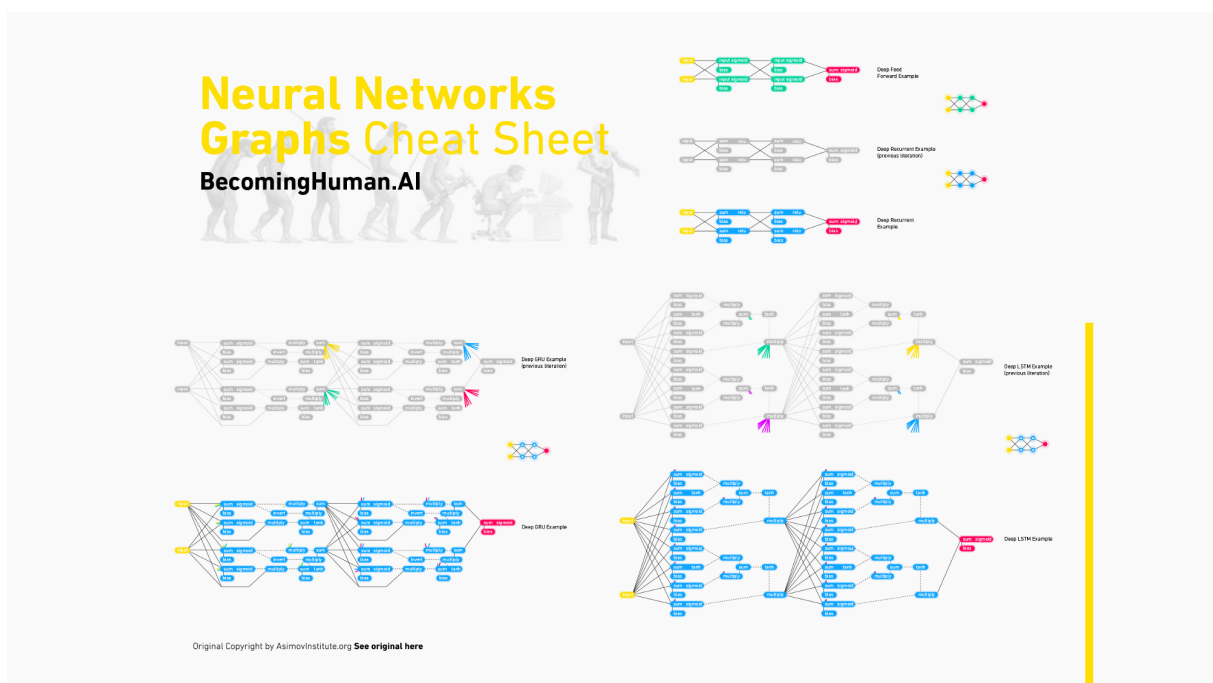
## **Downloadable Cheatsheets**

These PDF cheatsheets come from [BecomingHuman.AI](https://becominghuman.ai).

## 1 - Neural Network Basics



## 2 - Neural Network Graphs



### 3 - Machine Learning with Emojis

## Machine Learning Overview

# MACHINE LEARNING IN EMOJI

## Becoming Human.AI

**SUPERVISED** human builds model based on input / output

**UNSUPERVISED** human input, machine output

**REINFORCEMENT** human input, machine output  
human reward/punish, cycle continues

### BASIC REGRESSION

**LINEAR**  
`linear_model.LinearRegression()`  
Lots of numerical data

**LOGISTIC**  
`linear_model.LogisticRegression()`  
Target variable is categorical

### CLUSTER ANALYSIS

**K-MEANS**  
`cluster.KMeans()`  
Similar datum into groups based on centroids

**ANOMALY DETECTION**  
`covariance.EllipticEnvelope()`  
Finding outliers through grouping

### CLASSIFICATION

**NEURAL NET**  
`neural_network.MLPClassifier()`  
Complex relationships. Prone to overfitting. Basically magic.

**K-NN**  
`neighbors.KNeighborsClassifier()`  
Group membership based on proximity

**DECISION TREE**  
`tree.DecisionTreeClassifier()`  
If/then/else. Non-contiguous data. Can also be regression.

**RANDOM FOREST**  
`ensemble.RandomForestClassifier()`  
Find best split randomly. Can also be regression.

**SVM**  
`svm.SVC()` / `svm.LinearSVC()`  
Maximum margin classifier. Fundamental Data Science algorithm

**NAIVE BAYES**  
`naive_bayes.MultinomialNB()` / `naive_bayes.BernoulliNB()`  
Updating knowledge step by step with new info

### FEATURE REDUCTION

**T-DISTRIB STOCHASTIC NEIB EMBEDDING**  
Visual high dimensional data. Convert similarity to joint probabilities

**PRINCIPLE COMPONENT ANALYSIS**  
`decomposition.PCA()`  
Distill feature space into components that describe greatest variance

**CANONICAL CORRELATION ANALYSIS**  
`decomposition.CCA()`  
Making sense of cross-correlation matrices

**LINEAR DISCRIMINANT ANALYSIS**  
`lda.LDA()`  
Linear combination of features that separates classes

### OTHER IMPORTANT CONCEPTS

**BIAS VARIANCE TRADEOFF**

**UNDERFITTING / OVERFITTING**

**INERTIA**

**ACCURACY FUNCTION**  
`metrics.accuracy_score()`

**PRECISION FUNCTION**  
`metrics.precision_score()`

**SPECIFICITY FUNCTION**  
`metrics specificity_score()`

**SENSITIVITY FUNCTION**  
`metrics.sensitivity_score()`

Originally Created by Emily Barry. See original here

### 4 - Scikit-Learn With Python

## Cheat-Sheet Skicit learn

# Phyton For Data Science

## Becoming Human.AI

### Scikit Learn

Scikit Learn is an open source Python library that implements a range of machine learning, processing, cross validation and visualization algorithms using a unified API

**A basic Example**

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.cross_validation import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, random_state=0)
>>> scaler = preprocessing.StandardScaler()
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

**Prediction**

**Supervised Estimators**

```
>>> y_pred = knn.predict(X_test)
>>> y_pred = knn.predict_proba(X_test)
```

**Unsupervised Estimators**

```
>>> y_pred = knn.predict(X_test)
```

**Loading the Data**

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrix, other types that they are convertible to numeric arrays, such as Pandas Dataframes, are also acceptable

```
>>> import numpy as np
>>> X = np.random.random(10,10)
>>> y = np.array(['A','B','C','D','E','F','G','H','I','J'])
>>> X[X < 0.75] = 0
```

### Preprocessing The Data

**Standardization**

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

**Normalization**

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer()
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

**Binarization**

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer()
>>> binarized_X = binarizer.transform(X_train)
>>> binarized_X_test = binarizer.transform(X_test)
```

**Encoding Categorical Features**

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> encoder = OneHotEncoder()
>>> encoded_X = encoder.fit_transform(X_train)
>>> encoded_X_test = encoder.transform(X_test)
```

**Imputing Missing Values**

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
>>> imp.fit(X_train, y_train)
```

**Generating Polynomial Features**

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures()
>>> poly.fit(X_train, y_train)
```

### Evaluate Your Model's Performance

**Classification Metrics**

**Accuracy Score**

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

**Classification Report**

```
>>> from sklearn.metrics import classification_report
>>> precision_recall_fscore_support = classification_report(y_test, y_pred)
```

**Confusion Matrix**

```
>>> from sklearn.metrics import confusion_matrix
>>> precision_recall_fscore_support = confusion_matrix(y_test, y_pred)
```

**Regression Metrics**

**Mean Absolute Error**

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_test = [1, 5, 5, 2]
>>> mean_absolute_error(y_test, y_pred)
```

**Mean Squared Error**

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

**R<sup>2</sup> Score**

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_test, y_pred)
```

**Clustering Metrics**

**Adjusted Rand Index**

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_test, y_pred)
```

**Homogeneity**

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_test, y_pred)
```

**V-measure**

```
>>> from sklearn.metrics import v_measure_score
>>> v_measure_score(y_test, y_pred)
```

**Cross-Validation**

```
>>> from sklearn.cross_validation import cross_val_score
>>> cross_val_score(estimator, X_train, y_train, cv=5)
>>> cross_val_score(estimator, X_test, y_test, cv=5)
```

**Model Fitting**

**Supervised Learning**

```
>>> from sklearn import svm
>>> svm = svm.SVC()
>>> svm.fit(X_train, y_train)
```

**Unsupervised Learning**

```
>>> from sklearn import cluster
>>> cluster = cluster.KMeans()
>>> cluster.fit(X_train, y_train)
```

### Create Your Model

**Supervised Learning Estimators**

**Linear Regression**

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression()
>>> lr.fit(X_train, y_train)
```

**Support Vector Machines (SVM)**

```
>>> from sklearn.svm import SVC
>>> svm = SVC()
>>> svm.fit(X_train, y_train)
```

**Naive Bayes**

```
>>> from sklearn.naive_bayes import GaussianNB
>>> nb = GaussianNB()
>>> nb.fit(X_train, y_train)
```

**KNN**

```
>>> from sklearn.neighbors import KNeighborsClassifier
>>> knn = KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
```

**Unsupervised Learning Estimators**

**Principal Component Analysis (PCA)**

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=2)
>>> pca.fit(X_train, y_train)
```

**K-Means**

```
>>> from sklearn.cluster import KMeans
>>> km = KMeans(n_clusters=3, random_state=0)
>>> km.fit(X_train, y_train)
```

### Training And Test Data

```
>>> from sklearn.cross_validation import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

### Tune Your Model

**Grid Search**

```
>>> from sklearn.grid_search import GridSearchCV
>>> param_grid = {'n_neighbors': [3, 5, 7],
>>>               'weights': ['uniform', 'distance']}
>>> grid = GridSearchCV(estimator=knn, param_grid=param_grid)
>>> grid.fit(X_train, y_train)
>>> grid.best_estimator_
```

**Randomized Parameter Optimization**

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> param_grid = {'n_neighbors': [3, 5, 7],
>>>               'weights': ['uniform', 'distance']}
>>> rs = RandomizedSearchCV(estimator=knn, param_grid=param_grid, n_iter=10, random_state=0)
>>> rs.fit(X_train, y_train)
>>> rs.best_estimator_
```

Content Copyright by DataCamp.com. Design Copyright by BecomingHuman.AI. See Original here.

## 6 - NumPy Basics

8



## 7 - Pandas Basics

# Pandas Basics

## Cheat Sheet

BecomingHuman.AI  
DataCamp

Use the following import convention: `>>> import pandas as pd`

The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.

### Pandas Data Structures

**Series**

- 1-Dimensional labeled array
- capable of holding any data type
- `>>> s = pd.Series([1, 5, 7, 4], index=['a', 'b', 'c', 'd'])`

**DataFrame**

- 2-Dimensional labeled data structure with columns of potentially different types
- `>>> data = {'Country': ['Thailand', 'India', 'Brazil'], 'Capital': ['Bangkok', 'New Delhi', 'Brasilia'], 'Population': [11700000, 1303171026, 207847129]}`
- `>>> df = pd.DataFrame(data, columns=['Country', 'Capital', 'Population'])`

### Dropping

- `>>> s.drop('c')` Drop values from rows (axis=0)
- `>>> df.drop('Country', axis=1)` Drop values from columns (axis=1)

### Sort & Rank

- `>>> df.sort_index()` Sort by index along an axis
- `>>> df.sort_values('Country')` Sort by the values along an axis
- `>>> df.rank()` Assign ranks to entries

### Retrieving Series/DataFrame Information

- `>>> df.shape` Dimensions
- `>>> df.dtypes` Describe DataFrame columns
- `>>> df.columns` Info on DataFrame
- `>>> df.index` Index on DataFrame
- `>>> df.count()` Number of non-NA values

### Summary

- `>>> df.sum()` Sum of values
- `>>> df.cumsum()` Cumulative sum of values
- `>>> df.min()` Minimum/maximum values
- `>>> df.max()` Minimum/maximum index value
- `>>> df.describe()` Summary statistics
- `>>> df.mean()` Mean of values
- `>>> df.median()` Median of values

### Selection

**Getting**

- `>>> df[1]` Get one element
- `>>> df[1:3]` Get subset of a DataFrame

**Selecting, Boolean Indexing & Setting**

- By Position**
  - `>>> df.ix[0:100]` Select single row by row & column
  - `>>> df.ix[0:100, 0:100]` Select single row by row & column index
- By Label**
  - `>>> df.ix[0:100, 'Country']` Select single row of subset of rows
  - `>>> df.ix[0:100, ['Country', 'Population']]` Select a single column of subset of columns
- By Label/Position**
  - `>>> df.iat[0, 0]` Select row and column
  - `>>> df.iat[0:100, 0:100]` Select row and column
- Boolean Indexing**
  - `>>> df[df['Country'] == 'India']` Series a where value is not 1
  - `>>> df[df['Population'] > 1000000000]` a where value is 1 > 2
- Setting**
  - `>>> df['c'] = 4` Set index a of Series a to 4

### Asking For Help

`>>> help(pd.Series)`

### Applying Functions

- `>>> s.apply(lambda x: x**2)` Apply function
- `>>> df.applymap()` Apply function element-wise

### Data Alignment

**Internal Data Alignment**

NA values are introduced in the indices that don't overlap:

- `>>> s1 = pd.Series([1, 2, 3], index=['a', 'b', 'c'])`
- `>>> s2 = pd.Series([4, 5, 6], index=['b', 'c', 'd'])`
- `>>> s1 + s2`

**Arithmetic Operations with Fill Methods**

You can also do the internal data alignment yourself with the help of the fill methods:

- `>>> s1.fillna(0)`
- `>>> s2.fillna(0)`
- `>>> s1 + s2`

## 8 - Data Wrangling With Pandas

# Pandas

## Cheat Sheet

BecomingHuman.AI

### Pandas Data Structures

**Pivot**

- `>>> df.pivot(index='Year', columns='Type', values='Value')`

**Pivot Table**

- `>>> df.pivot_table(index='Year', columns='Type', values='Value')`

**Melt**

- `>>> df.melt(id_vars='Year', value_vars=['Type', 'Value'], var_name='Dimension', value_name='Value')`

### Advanced Indexing

**Selecting**

- `>>> df[df['a'] > 1]`
- `>>> df[df['a'] > 1 & df['b'] < 10]`
- `>>> df[df['a'] > 1 & df['b'] < 10 & df['c'] > 100]`

**Indexing With Labels**

- `>>> df[df['a'] == 'a']`
- `>>> df[df['a'] == 'a' & df['b'] < 10]`

**Where**

- `>>> df[df['a'] == 'a']`

**Query**

- `>>> df.query('a == "a"')`

**Setting/Resetting Index**

- `>>> df.set_index('Country')`
- `>>> df.reset_index()`

**Reindexing**

- `>>> df.reindex([0, 1, 2, 3])`
- `>>> df.reindex([0, 1, 2, 3], method='bfill')`

**Multindexing**

- `>>> df.groupby(['Year', 'Country']).sum()`
- `>>> df.groupby(['Year', 'Country']).sum().reset_index()`

### Combining Data

**Pivot**

- `>>> pd.merge(df1, df2, on='Year')`

**Join**

- `>>> df1.join(df2, how='right')`

**Concatenate**

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

**Dates**

- `>>> df['Year'] = pd.to_datetime(df['Year'])`
- `>>> df['Year'] = pd.to_datetime(df['Year']).year`

**Visualization**

- `>>> df.plot()`
- `>>> df.plot(kind='line')`

### Duplicate Data

- `>>> df.drop_duplicates()`
- `>>> df.drop_duplicates(inplace=True)`

### Grouping Data

- `>>> df.groupby('Year').sum()`
- `>>> df.groupby('Year').sum().reset_index()`

### Missing Data

- `>>> df.isnull()`
- `>>> df.fillna(0)`



## 11

## 12 - Big-O

