[The Data Scientist's Python Toolbox] [cheatsheet]

1. Data Manipulation and Analysis

- **Pandas**: Core library for data manipulation and analysis.
  - pd.read_csv(): Read data from a CSV file into a DataFrame.
  - pd.read_excel(): Read data from an Excel file into a DataFrame.
  - df.head(): View the first few rows of the DataFrame.
  - df.describe(): Get a summary of statistics.
  - df.info(): Get concise summary of the DataFrame.
  - df['column'].value_counts(): Count unique values in a column.
  - df.groupby(): Group data using a mapper or by a series of columns.
  - df.pivot_table(): Create a spreadsheet-style pivot table.
  - df.merge(): Merge DataFrame objects.
  - df.to_csv(): Write DataFrame to a comma-separated values (csv) file.
  - pd.DataFrame(): Create a DataFrame from various data sources.
  - df.filter(): Subset the data.
  - df.sort_values(): Sort data by a column.
  - df.groupby().agg(): Aggregation after grouping.
  - df.join(), df.merge(): Join/Merge operations.
  - df.plot(): Basic plotting.
  - df.apply(): Apply functions.
  - df.to_sql(), df.read_sql(): Interaction with SQL databases.
  - df.to_datetime(): Convert a column to DateTime.
  - pd.get_dummies(df): Convert categorical variable into dummy/indicator variables.

2. Numerical Operations

- **NumPy**: Fundamental package for numerical computations.
  - np.array(): Create an array.
  - np.reshape(): Change array shape.

  - `np.concatenate()`: Concatenate arrays.
  - `np.where()`: Return elements chosen from x or y depending on condition.
  - `np.linalg.inv()`: Compute the multiplicative inverse of a matrix.
  - `np.linalg.eig()`: Compute the eigenvalues and right eigenvectors of a square array.
  - `np.arange()`: Return evenly spaced values within a given interval.
  - `np.zeros()`, `np.ones()`: Create arrays of zeros or ones.
  - `np.linspace()`: Create evenly spaced numbers over a specified interval.
  - `np.random.rand()`, `np.random.randn()`: Create arrays of random values.
  - `np.dot()`: Dot product of two arrays.
  - `np.sqrt()`, `np.log()`, `np.exp()`: Square root, logarithm, exponentiation.

3. Data Visualization

- **Matplotlib**: Basic plotting library.
  - `plt.plot()`: Plot y versus x as lines and/or markers.
  - `plt.scatter()`: Make a scatter plot of x vs y.
  - `plt.hist()`: Plot a histogram.
  - `plt.bar()`: Make a bar plot.
  - `plt.xlabel()`, `plt.ylabel()`: Set the labels for x and y axes.
  - `plt.title()`: Set a title for the axes.
  - `plt.legend()`: Place a legend on the axes.
  - `plt.figure()`: Create a new figure.
  - `plt.subplot()`: Add a subplot to the current figure.
  - `plt.xscale()`, `plt.yscale()`: Set the scaling of the x-axis or y-axis.
  - `plt.xlim()`, `plt.ylim()`: Get or set the x/y limits of the current axes.
  - `plt.colorbar()`: Add a colorbar to a plot.

  - plt.errorbar(): Plot y versus x as lines and/or markers with attached errorbars.
- **Seaborn**: Statistical data visualization based on Matplotlib.
  - sns.set(): Set aesthetic parameters in one step.
  - sns.pairplot(): Plot pairwise relationships in a dataset.
  - sns.distplot(): Flexibly plot a univariate distribution of observations.
  - sns.boxplot(): Draw a box plot to show distributions with respect to categories.
  - sns.heatmap(): Heatmap representation of data.
  - sns.lmplot(): Plot data and regression model fits.
  - sns.clustermap(): Clustered heatmap.
  - sns.jointplot(): Draw a plot of two variables with bivariate and univariate graphs.
  - sns.swarmplot(): Draw a categorical scatterplot with non-overlapping points.
  - sns.countplot(): Show the counts of observations in each categorical bin using bars.

4. Statistical Analysis

- **SciPy**: Library for scientific computing.
  - stats.ttest_ind(): Calculate the T-test for the means of two independent samples of scores.
  - stats.pearsonr(): Pearson correlation coefficient and p-value for testing non-correlation.
  - stats.norm(): Normal continuous random variable.
  - scipy.integrate.quad(): General purpose integration.
  - scipy.optimize.minimize(): Minimization of scalar functions of one or more variables.
  - scipy.signal.convolve(): Convolve two N-dimensional arrays.
  - scipy.interpolate.interp1d(): Interpolate a 1-D function.
  - scipy.spatial.distance.euclidean(): Computes the Euclidean distance between two 1-D arrays.

5. Machine Learning

- **Scikit-learn**: Core library for machine learning.
  - train_test_split(): Split arrays or matrices into random train and test subsets.
  - LinearRegression(), LogisticRegression(): Linear and Logistic Regression models.
  - RandomForestClassifier(), RandomForestRegressor(): Random Forest models for classification and regression.
  - KMeans(): K-Means clustering.
  - cross_val_score(): Evaluate a score by cross-validation.
  - GridSearchCV(): Search over specified parameter values for an estimator.
  - confusion_matrix(), classification_report(): Compute confusion matrix and a text report showing the main classification metrics.
  - sklearn.preprocessing.StandardScaler(): Standardize features by removing the mean and scaling to unit variance.
  - sklearn.decomposition.PCA(): Principal component analysis (PCA).
  - sklearn.cluster.KMeans(): K-Means clustering.
  - sklearn.model_selection.cross_val_score(): Evaluate a score by cross-validation.
  - sklearn.metrics.accuracy_score(), roc_auc_score(): Classification metrics.
- **XGBoost**: Gradient boosting framework.
  - XGBClassifier(), XGBRegressor(): XGBoost classifier and regressor.
  - xgb.train(): Train a gradient boosting model.
  - xgb.DMatrix(): Optimized data structure for XGBoost.
- **LightGBM**: Light Gradient Boosting Machine.
  - LGBMClassifier(), LGBMRegressor(): LightGBM classifier and regressor.
  - lgb.train(): Train a gradient boosting model.
  - lgb.Dataset(): Dataset for LightGBM.
- **Statsmodels**: Library for statistical models, hypothesis tests, and data exploration.

  - sm.OLS(), sm.Logit(): Models for linear regression and logistic regression.
  - sm.tsa.ARIMA(): ARIMA model for time series analysis.

6. Deep Learning

- **TensorFlow**: Open-source machine learning framework.
  - tf.keras.models.Sequential(): Sequential model for linear stack of layers.
  - tf.keras.layers.Dense(): Regular densely-connected NN layer.
  - tf.keras.layers.Conv2D(), tf.keras.layers.MaxPooling2D(): 2D Convolutional and Pooling layers.
  - tf.GradientTape(): Record operations for automatic differentiation.
  - tf.data.Dataset: Create a dataset from tensors.
  - tf.keras.Sequential(): Linear stack of layers.
  - tf.keras.models.Model(): Model class with Keras functional API.
  - tf.keras.layers.LSTM(): Long Short-Term Memory layer.
  - tf.train.AdamOptimizer(): Adam optimizer.
- **Keras**: High-level neural networks API.
  - keras.models.load_model(): Load a Keras model.
  - keras.preprocessing.image.ImageDataGenerator(): Generate batches of tensor image data with real-time data augmentation.
  - keras.Model(): Group layers into an object with training and inference features.
  - keras.layers.Conv2D(): 2D convolution layer.
  - keras.activations.relu, sigmoid: Activation functions.
  - keras.callbacks.ModelCheckpoint, EarlyStopping: Callbacks for model training.
- **PyTorch**: Open source machine learning library.
  - torch.nn.Module: Base class for all neural network modules.
  - torch.Tensor: Multi-dimensional matrix containing elements of a single data type.
  - torch.optim: Optimization algorithms.

    - torch.utils.data.DataLoader: Combine a dataset and a sampler.

7. Natural Language Processing (NLP)

- **NLTK**: Leading platform for building Python programs to work with human language data.
    - nltk.tokenize.word_tokenize(): Tokenize a string to split off punctuation other than periods.
    - nltk.corpus.stopwords.words(): List of stopwords.
    - nltk.FreqDist(): Frequency distribution of words within a text.
    - nltk.tag.pos_tag(): Part-of-speech tagging.
    - nltk.corpus: Access to large text corpora.
    - nltk.tokenize.sent_tokenize(): Tokenizer for sentences.
    - nltk.stem.PorterStemmer(): Porter word stemmer.
    - nltk.pos_tag(): Part-of-speech tagging.
    - nltk.NaiveBayesClassifier(): Naive Bayes classifier.
    - nltk.chunk(): Chunking for entity recognition.
- **spaCy**: Industrial-strength Natural Language Processing.
    - spacy.load(): Load a model.
    - doc = nlp(text): Process a text.
    - doc.ents: Named entities.
    - doc.sents: Sentence segmentation.
    - nlp(): Process raw text.
    - doc.sents: Generate sentence spans.
    - doc.ents: Named entity recognition.
    - doc.similarity(): Similarity between two documents.
    - spacy.lang: Language-specific models.

8. Working with Databases

- **SQLAlchemy**: SQL toolkit and Object-Relational Mapping (ORM) library.
    - create_engine(): Database engine.
    - sessionmaker(): Session factory.
    - Base: Declarative base class for ORM.

- **sqlite3**: SQLite database library.
  - sqlite3.connect(): SQLite database connection.
  - cursor.execute(): Execute a SQL command.
  - cursor(): Create a cursor object to call its execute() method to perform SQL commands.

9. Web Scraping

- **BeautifulSoup**: Library for pulling data out of HTML and XML files.
  - BeautifulSoup(): Parse an HTML/XML document.
  - .find(), .find_all(): Find elements by tags.
- **Scrapy**: Open source and collaborative framework for extracting data from websites.
  - scrapy.Spider: Base class for spiders.
  - response.css(), response.xpath(): Querying the data.
  - yield scrapy.Request(): Generate Requests.
  - parse(): Method to handle responses.

10. Data Visualization (Advanced)

- **Plotly**: Interactive graphing library.
  - plotly.graph_objs.Scatter(), plotly.graph_objs.Bar(): Create scatter and bar plots.
  - plotly.subplots.make_subplots(): Create subplots.
  - plotly.express.scatter(), bar(), line(): Quick functions for scatter, bar, and line plots.
  - plotly.graph_objs.Figure(): Create figures for a more custom approach.
  - plotly.io.write_html(): Save plot as HTML file.
  - plotly.subplots.make_subplots(): Make figures with subplots.
- **Bokeh**: Interactive visualization library.
  - figure(): Create a new figure for plotting.
  - output_file(), output_notebook(): Output to static HTML file or Jupyter Notebook.
  - ColumnDataSource(): Map names of columns to sequences or arrays.

    - show(), save(): Display or save plots.
    - widgets: Interactive widgets for plots.
- **Altair**: Declarative statistical visualization library.
    - alt.Chart(): Create a Chart object.
    - mark_point(), mark_line(), mark_bar(): Different types of marks for visualization.
    - encode(): Encode visual channels.
- **Folium**: Map plotting library.
    - folium.Map(): Create a base map.
    - folium.Marker(), folium.CircleMarker(): Add markers to the map.

11. Data Reporting and Business Intelligence

- **Dash**: Web application framework.
    - dash.Dash(): Create a Dash application.
    - dash.html.Div, dash_core_components: HTML components and core components for Dash.
    - dash.dcc.Graph: Graph components for Dash.
    - app.callback(): Decorator for callbacks.
- **Streamlit**: App framework for Machine Learning and Data Science teams.
    - st.write(): Write data or text.
    - st.dataframe(): Display a dataframe.
    - st.plotly_chart(): Display a Plotly chart.
    - st.sidebar.selectbox(): Add a select box to the sidebar.

12. Advanced Machine Learning

- **CatBoost**: Gradient boosting on decision trees library.
    - CatBoostClassifier(), CatBoostRegressor(): CatBoost models for classification and regression.
    - catboost.Pool(): Data structure to store dataset.
- **Hyperopt**: Distributed Asynchronous Hyperparameter Optimization.
    - fmin(): Minimize a function over a hyperparameter space.
    - hp.choice(), hp.uniform(): Define hyperparameter space.

- **Optuna**: Hyperparameter optimization framework.
    - create_study(): Create a study for hyperparameter optimization.
    - optimize(): Optimize the objective function.

13. Model Interpretability and Explainability

- **SHAP (SHapley Additive exPlanations)**: Explain the output of machine learning models.
    - shap.TreeExplainer(): Explain predictions of tree-based models.
    - shap_values(): Compute SHAP values.
- **LIME (Local Interpretable Model-agnostic Explanations)**: Explain individual predictions.
    - lime.lime_tabular.LimeTabularExplainer(): Explainer for tabular data.
    - explain_instance(): Explain an individual instance.

14. Data Imputation

- **Imputer from Scikit-learn**: Imputation for completing missing values.
    - SimpleImputer(): Imputation transformer for completing missing values.
- **KNNImputer from Scikit-learn**: Imputation for filling in missing values using the k-Nearest Neighbors approach.
    - KNNImputer(): Impute missing values using k-NN.

15. Feature Engineering and Selection

- **Feature-engine**: Feature engineering library.
    - CategoricalImputer(), NumericalImputer(): Impute missing categorical or numerical values.
    - MathematicalCombination(): Create new features by combining mathematical operations.
- **SelectKBest from Scikit-learn**: Select features according to the k highest scores.

    - SelectKBest(): Select features according to the top k scores.
- **RFE (Recursive Feature Elimination) from Scikit-learn**: Feature ranking with recursive feature elimination.
    - RFE(): Recursive feature elimination.

16. Time Series Analysis

- **Prophet**: Forecasting time series data.
    - Prophet(): Create a new Prophet object.
    - model.fit(): Fit the Prophet model.
    - model.predict(): Make a future prediction.
- **tsfresh**: Automatic extraction of relevant features from time series.
    - extract_features(): Automatically extract time series features.

17. Image and Video Processing

- **OpenCV (cv2)**: Open Source Computer Vision Library.
    - cv2.imread(), cv2.imshow(): Read and display images.
    - cv2.VideoCapture(): Capture video from a camera or a file.
    - cv2.cvtColor(): Color space conversion.
    - cv2.CascadeClassifier(): Haar cascade classifiers for object detection.
    - cv2.findContours(): Find contours in a binary image.
- **Pillow (PIL)**: Python Imaging Library.
    - Image.open(), Image.save(): Open and save images.
    - Image.filter(), ImageEnhance: Apply image filters and enhancements.
    - image.rotate(), image.resize(): Rotate or resize an image.
    - ImageDraw.Draw(): Create object to draw on the image.

18. Model Deployment

- **Flask**: Micro web framework for building web applications.
    - flask.Flask(): Create a Flask application.
    - app.route(): Define routes for your application.

    - flask.request: Request object to handle query parameters, URLs, etc.
- **FastAPI**: Modern, fast (high-performance) web framework.
    - FastAPI(): Create a FastAPI application.
    - @app.get(), @app.post(): Define GET and POST endpoints.
    - pydantic.BaseModel: Define data models.

19. Working with Data Streams

- **Apache Kafka for Python (confluent_kafka)**: Client for Apache Kafka.
    - Producer(), Consumer(): Produce and consume messages.
- **PySpark Streaming**: Processing real-time data streams.
    - StreamingContext(): Main entry point for streaming functionality.
    - DStream: Discretized stream for processing.
    - SparkContext(): Entry point for Spark functionality.
    - RDD: Resilient Distributed Dataset for fault-tolerant processing.
    - spark.sql(): Running SQL queries.
    - DataFrame: Distributed collection of data organized into named columns.

20. Geospatial Data Analysis

- **Geopandas**: Work with geospatial data in Python.
    - geopandas.read_file(): Read geospatial data.
    - GeoDataFrame(): Geospatial dataframe.
- **Rasterio**: Access to geospatial raster data.
    - rasterio.open(): Open raster files.
- **Folium (continued)**: Build interactive maps.
    - folium.Map(): Create a base map.
    - folium.features.GeoJson(): Add GeoJSON to a map.

21. Advanced Data Storage and Retrieval

- **HDF5 for Python (h5py)**: Work with HDF5 binary data format.
    - h5py.File(): Open an HDF5 file.

  - `create_dataset()`: Create a new dataset in an HDF5 file.
- **PyTables**: Manage large datasets and hierarchical databases.
  - `tables.open_file()`: Open an HDF5 file.
  - `create_table()`, `create_array()`: Create tables and arrays in the file.

22. Cloud Services and APIs

- **Boto3 for AWS**: Amazon Web Services SDK for Python.
  - `boto3.client()`, `boto3.resource()`: Access AWS services.
- **google-cloud-python**: Client libraries for Google Cloud services.
  - `from google.cloud import storage`: Access Google Cloud Storage.

23. Optimization **and** Solvers

- **CVXPY**: Domain-specific language for convex optimization problems.
  - `cvxpy.Problem()`: Create an optimization problem.
  - `problem.solve()`: Solve the optimization problem.