**The Essential Data Science Reference Notebook**

**A single reference for EDA, feature engineering, model fitting, and more**



Photo by [James Harrison](https://unsplash.com/@jstrippa?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

I recently completed an internal transfer process to join Meta’s Core Data Science team. While studying for these interviews with my colleagues, Barry Zhang and Nicolas Lepore, we found that there was no singular resource for typical data science coding functions…

This article details the sections of our notebook and includes download links at the bottom for those who may also find this useful.

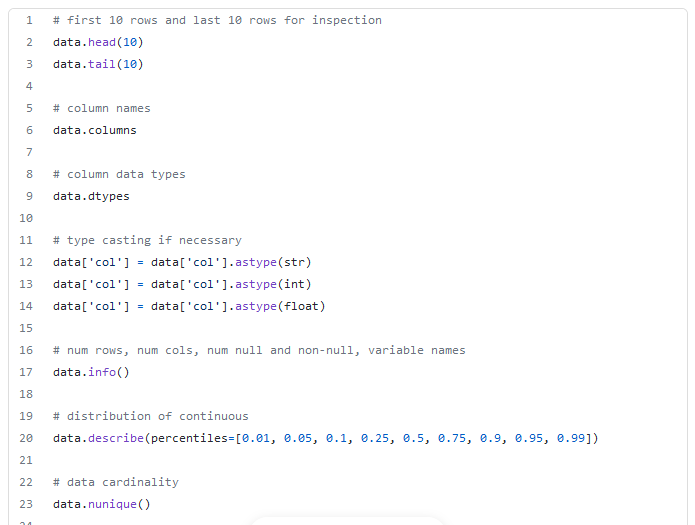
**Quick Disclaimer**

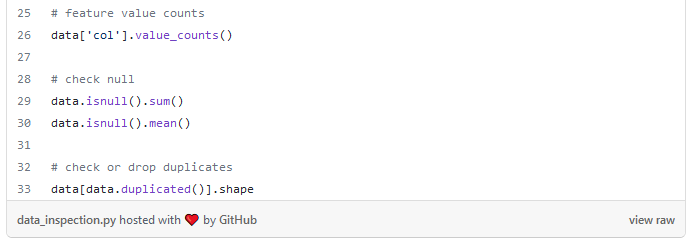
The order in which each section appears is generally the order in which these steps are typically carried out, but isn’t perfect. For example, most data transformations like scaling and one hot encoding should be done *after* splitting your data into train and test, but this notebook has them before for simplicity.

**Data Inspection**

The first step it to inspect the dataset. What we’re interested in here is the shape and size of the data, what types the features are (string/float/int ect.), the distribution or number of classes of those features, and whether there are missing or duplicated values.

<https://gist.github.com/eonofrey/b24144689af361078e35def3a574f135#file-data_inspection-py>



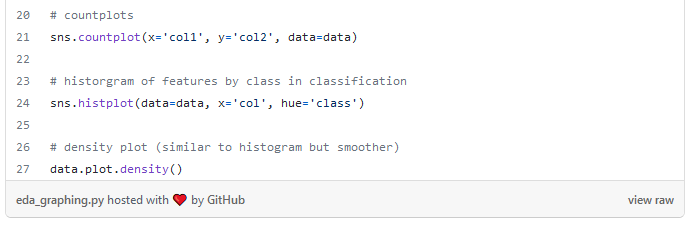


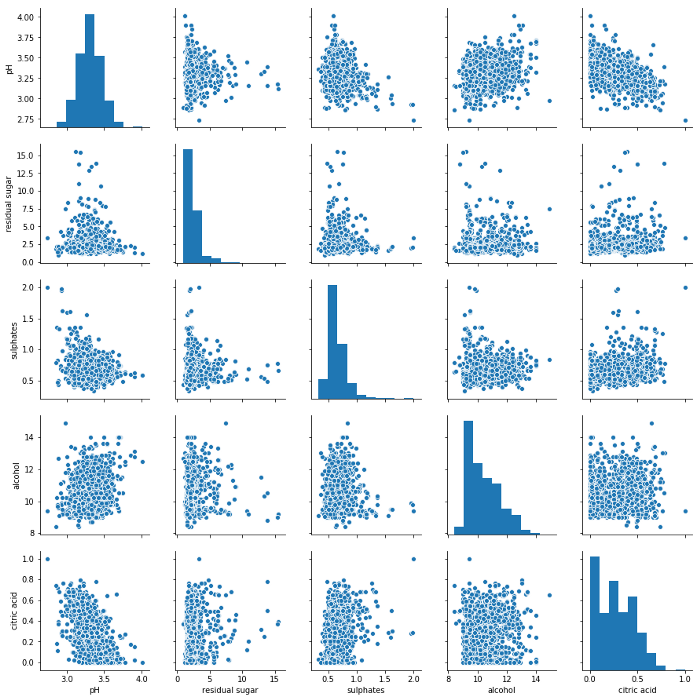
**EDA Plotting**

Once we know the basic things about our data like what the column names are, how large it is, if there are null values etc. we can move onto visualizing it to gain better understanding. Below we use heatmaps to visualize correlations in the data and to see if there are patterns in the distributions of null values. In addition to this, a scatterplot matrix will display a grid of pairwise scatterplots for all numeric values. For simple bivariate analysis, we also include one-off scatter plots, hex plots, box plots, count plots, and histograms.

<https://gist.github.com/eonofrey/3500f71d0be3d50950c017d062714348#file-eda_graphing-py>







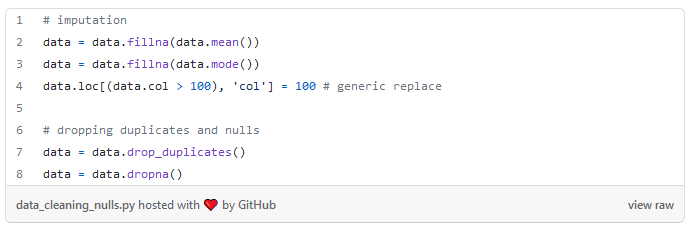
Scatterplot Matrix, Image by Author

**Data Cleaning**

Now we turn our attention to null values, duplicates and outliers. For null values, it’s always important to try to find out what happened to result in a null value and to see if they are distributed randomly or not. However, in the short time window of an interview, it’s usually fine to just quickly impute them with the mean/median (for continuous features) or the mode (for categorical features). More advanced techniques involve building a model with the remaining features to predict the null, but again this typically isn’t expected in an interview or take home assignment.

If the number of null values is small relative to the size of the dataset and they are randomly distributed, we can also drop them as we typically do with duplicates.

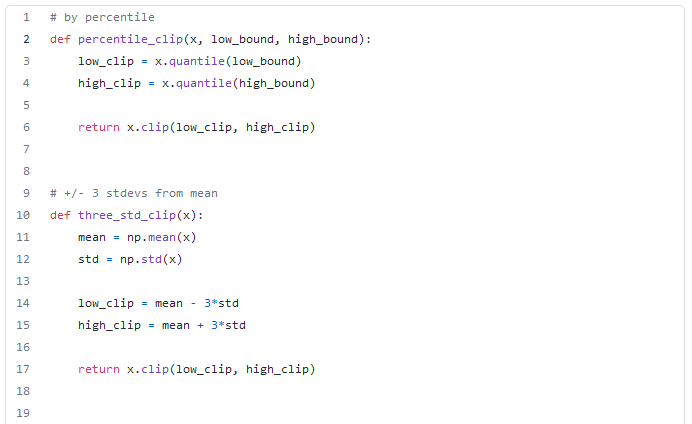
<https://gist.github.com/eonofrey/d1f9dbd6d8b86ce6d63d3f7bdfdaaad2#file-data_cleaning_nulls-py>

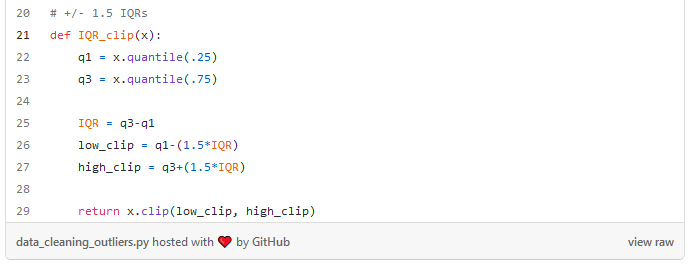


While tree-based models or neural networks are decently robust to outliers, it’s a good idea to winsorize them for linear models. We provide 3 functions for clipping clipping outliers based on:

***Percentiles:*** *For example clipping all values above the 99th percentile cutoff and below 1% cutoff.****Standard Deviations:*** *Clipping all values more than 3 standard deviations away from the mean.****Inter-Quartile Range:*** *Clipping all values 1.5\*IQR outside of the lower and upper bounds of the IQR (side note: Shivam Chaudhary has a great* [*article*](https://towardsdatascience.com/why-1-5-in-iqr-method-of-outlier-detection-5d07fdc82097) *on why we use 1.5 as the scale factor)*

<https://gist.github.com/eonofrey/261ac75ac7989a747063ba43151db9e7#file-data_cleaning_outliers-py>





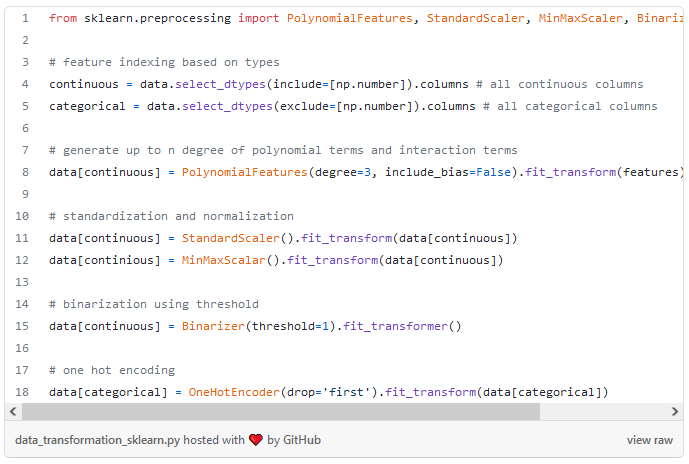
**Data Transformation**

Before fitting a model, we have to get the data in a usable form. This typically involves some sort of feature transformation on categorical and/or continuous features. We start by selecting the columns for each datatype using the .select\_dtypes() method.

For the continuous features we might want to generate polynomial features if we’re using a linear model that cannot fit non-linear trends on its own. It’s also a good idea to scale numeric features with either the StandardScaler or MinMaxScaler, which will help models converge faster and is necessary when doing regularizations like Lasso and Ridge regressions that penalize for the size of a coefficient. Last, you may want to transform continuous features to binary (“binarization”) by picking a threshold for the 1/0 cutoff.

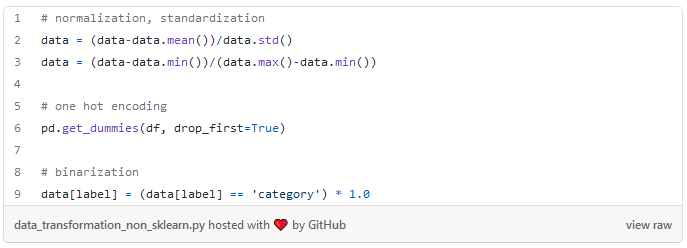
For categorical features, nearly all learning algorithms written in Python cannot use string features, so we need to one-hot-encode the columns. In this process, we use the OneHotEncoder class in sklearn to convert a string feature to a series of 1/0 indicator columns for each class of the feature. We include the “drop=’first’” parameter to avoid the [dummy variable trap](https://medium.com/nerd-for-tech/what-is-dummy-variable-trap-how-it-can-be-handled-using-python-78ec17246331).

<https://gist.github.com/eonofrey/b02fb82cc6b2b482bdf02231327815d5#file-data_transformation_sklearn-py>



Not everyone loves using sklearn for these transformations, however, so we also provide alternatives to standard scaling, min-max scaling, one hot encoding, and binarization for categorical variables below.

<https://gist.github.com/eonofrey/13e4a50df5b1220b04c659f9fa669d10#file-data_transformation_non_sklearn-py>



Finally, if during the EDA process you observe that there is a large label imbalance, it may be necessary to upsample or down-sample your dataset to force those classes to be more equally represented. There are more complex ways to achieve this (Oversampling: SMOTE & ADASYN algorithms, Undersampling: Tomek links and cluster-based) but again for the purposes of an interview or take home assignment just taking random samples with replacement should be sufficient.

<https://gist.github.com/eonofrey/a7d39d05d39ff347c75f06b0041580e8#file-sampling-py>



**Supervised Learning**

Now we can start the modeling process. Sklearn’s API is standardized in the sense that every modeling processes is essentially:

model = Model()  
model.fit(X\_train, y\_train)  
model.predict(X\_test)

Below we import and instantiate some of the most common models from sklearn and xgboost, along with some of their main hyperparameters.

<https://gist.github.com/eonofrey/633b026c9af4ec93d2ba01516266b84f#file-model_import_instantiate-py>



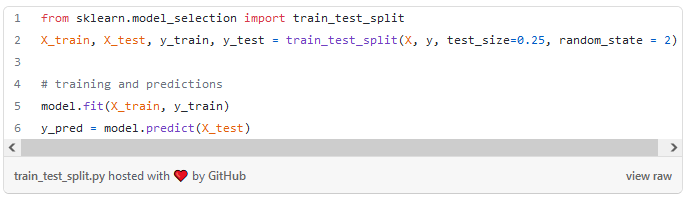
The above hyperparameters we use in the models are arbitrarily chosen. One might want to perform a grid search to chose the best hyperparameters for that model and that dataset. Here we show a simple grid search using the GridSerachCV library to pick the best *n\_estimators* and *max\_depth* hyperparameters for a random forest classifier. In practice, a full grid search is pretty compute-intensive, so it may be best to opt for a random grid search or a bayesian search as described by Maria Gorodetski in her [article](https://towardsdatascience.com/bayesian-optimization-for-hyperparameter-tuning-how-and-why-655b0ee0b399). For simplicity we just show a simple grid search with few parameters.

<https://gist.github.com/eonofrey/c4e36719bbad4be4ab5927cff037f736#file-grid_search-py>



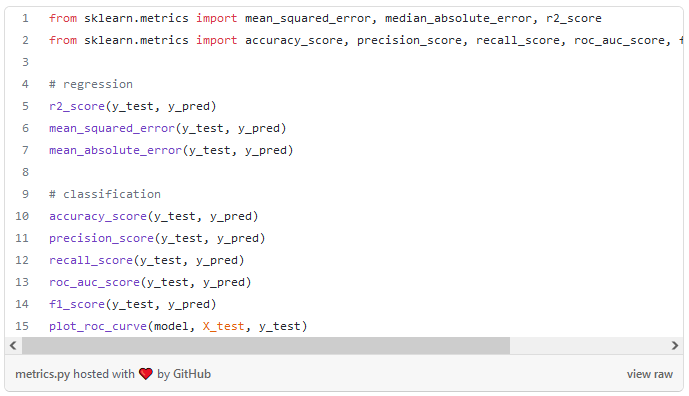
If we aren’t doing cross validation, typically we will split our data into train and test to train the model and then evaluate it on data its never seen before to get a fair assessment of its performance. Sklearn’s train\_test\_split is the perfect function for this as shown below:

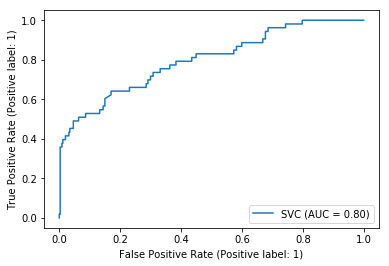
<https://gist.github.com/eonofrey/2baae4570c588bf238d6d4d72e0eaff4#file-train_test_split-py>



Once we have our model trained, we use it to predict values on the test set. For regression we can look at r², MSE, and MAE. For classification, we have accuracy, precision, recall, F1, and AUC. Sklearn also has a convenient function that will plot the ROC curve if given the model, and test features and labels.

<https://gist.github.com/eonofrey/9195e883f523163259ebd81de18aa79d#file-metrics-py>

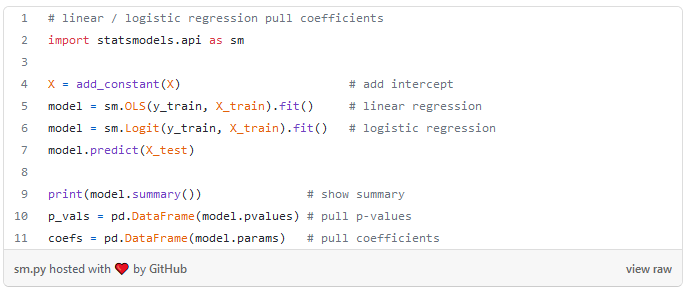




sklearn’s ROC curve, Image by Author

While sklearn is the most ubiquitous library used for ML, its outputs for linear and logistic regression are somewhat limited in the sense that they just give the coefficients of features. Statsmodel will provide coefficients along with their p-values and other valuable information such as the F-statistic, Durbin-Watson statistic, AIC, BIC, r², adjusted r² etc.

<https://gist.github.com/eonofrey/7512bbe5c7c06f0b4c26d383ee854fc9#file-sm-py>



With random forest models, we can’t get coefficients like with linear models but we are able to pull feature importances:

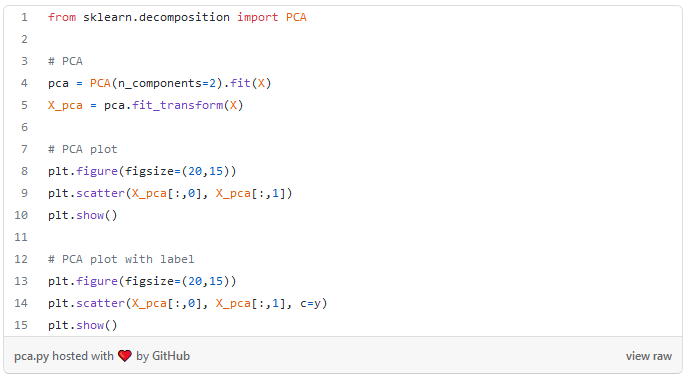
<https://gist.github.com/eonofrey/bbad5ce540d5262303493207e11f5f64#file-feature_importance-py>



**Unsupervised Learning**

To round out the notebook, we briefly include some unsupervised techniques. Principal component analysis (PCA) is helpful to reduce the dimensionality of a dataset while still preserving the majority of the information in it. This can help with visualize data, combine correlated features , and speed up training for large learning algorithms. In the code snippet below we reduce a dataset “X” to 2 components and plot it.

<https://gist.github.com/eonofrey/d7c8bbfde28460a4245859f8566cd7d4#file-pca-py>



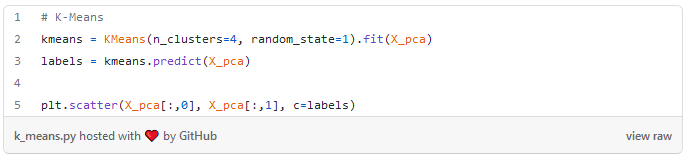
Finally, we close out with KMeans clustering. Unless you have a reason stemming from domain knowledge to pick the number of clusters for kmeans, in practice most people use the [Elbow Method](https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/) as demonstrated below. In this method, one plots the number of cluster by the inertia (sum of squared distances of data points to their nearest cluster center point) and choses the number of clusters at the kink in the graph.

<https://gist.github.com/eonofrey/7957c0b162078a00ad414b2b0c5f267f#file-elbow_method-py>



Once the number of clusters is chosen, fitting and predicting is similar to the other models in sklearn except for the fact that it has no labels since this is unsupervised learning.

<https://gist.github.com/eonofrey/f6a1396549ea6dd73441f8e54ac55ff0#file-k_means-py>



The above code is a distillation of learnings from dozens of books, countless Kaggle submissions, and years of experience. None of it is incredibly elegant or complex, but it serves as a fantastic single reference for the majority of data science functions. You can download the full Python notebook from [my](https://github.com/eonofrey/ds_reference_notebook) or [Barry’s](https://github.com/nub3Ar) Github. To get full access to Medium, click [here](https://eonofrey.medium.com/membership)!