— — — — —

READ ME

- How to use this guide
  - Currently the best way to search through this guide is to press CMD+F (if on Mac) or Ctrl+F (if on Windows/Linux) and to search for the following keywords
    - snippets, examples, python, plt, sns, statistics, sparsity, ETL, read_csv, get_dummies, probability, keras, tflearn, impute, onehotencod, dummy v, PCA, Hive, CNN, ensemble, validation, memory, usage, Hadoop, MAE, RSME, confusion, grid, function, matplotlib, plot, seaborn, facet, violin, histogram, bar, box, anomaly, count, label, title, libraries, sklearn. , keras. , tensor, from import, subplotting, regression, regressor, linear, logistic, SVM, clustering, classification, KNN, k-m, boosting, xgb, decision tree, random forest, bagging, deep, learning, convolution, dense, bias, variance, skew, heat, drop, probability, pandas, preprocessing, skew, correction, subsetting, dataframe, for i, apply, column, row, data engineering, concatenate, aggregate, terminology, important, feature, index, unique, value, naive bay, scaling, AB, A/B, predict, classifier, model, X_train, X, y_val, y_test, error, deviation, accurate, metric, melt, loc, iloc, .column, NaN, missing, lambda, np.expm1, np.log1p, split, CART, gradient, Series, Index, list, array, convert, lightgbm, xgboost, catboost, reinforcement, rule, selection, gaussian, numba, timeit, dendogram

— — — — —

COMMON DS QUESTIONS

- Probability
  - Probability of rolling three dice without getting a six
    - To obtain the probability that at least one 6 is rolled in the three tosses
      - 1- (5/6)^3 Note, the probability of rolling a 6 with one dice on one roll is 1/6.
  - Probability of rolling double sixes twice in a roll (rolling two dices at a time, two times)
    - Since each dice is being rolled independently
      - (1/6)^1 * (1/6)^1 * 1/6)^1 * (1/6)^1 = (1/6)^4 = 0.00077
  - Probability of rolling one dice twice and getting a six both times
    - (1/6)^1 * (1/6)^1 = (1/6)^2 = 1/36
  - https://sciencing.com/calculate-dice-probabilities-5858157.html

- Statistics

  - Multicollinearity (occurs when independent variables in a regression model are correlated.)

- https://towardsdatascience.com/multicollinearity-in-data-science-c5f6c0fe6edf
- Bias and Variance Tradeoff (Great)
  - https://www.youtube.com/watch?v=EuBBz3bI-aA
    - The first thing we do before start looking bias and variance is to split out data into a training and test set
      - Example
        - imagine viewing a scatter plot with data in a log10 shape and showing 80% of the markers/ data as blue and the remaining 20% as green. The blue dots are the training set and the green dots are the testing set
        -
    - The inability for a machine learning method (like linear regression) to capture the true relationship is called **bias**
      - Example
        - Think about trying to fit a linear line to a training set with log10 type scatter data
        - Because the linear line can't be curved like the "true" relationship, we say it has a relatively large amount of bias. In other words, the inability for a machine learning method (like linear regression) to capture the true relationship is called **bias**
      - Example
        - Another machine learning method, may try to fit a squiggly line (zig-zag type shape) to the training set. The squiggly line can handle to the arc of a true relationship with log10 type data so we can it would have a relatively low amount of **bias** compared to trying trying to fit a straight line to log10 type data.
          - To quantify this, we can compare the sum of squares between the squiggly line model and straight line model. But note, the squiggly line model fits the scatter plot so well, that the sum of squares will be zero! The squiggly line wins here at modeling the training set data.
          - Lets now look at the testing set and calculate the sum of squares
          - The straight line wins here! It's sum of squares is lower than that of the squiggly line fine
    - Variance

        - Example
            - Since the squiggly line model fitted the training set so well, but failed to fit the testing set well, we say it has a relatively large amount of variance. In other words, the difference in fits between the data sets is called variance.
            - In summary, the squiggly line had low bias with the training and test data since it is flexible and can adapt to the log10 shaped data. But…the squiggly line has high variability/variance because it results in a vastly different sums of squares for different data sets. In other words, it's hard to predict how well the squiggly line will perform with future data sets. It might do well sometimes, and other times it might do terribly.
            - In contrast, the straight line has relatively high balance when trying to capture the true relationship of the log10 shaped data. But…. The straight line has relatively low variability/ variance because the sums of squares are very similar between different datasets. In other words, the straight line might only give good predictions, and not *great* predictions. But they will be consistently good predictions
            - Terminology alert:
                - If the model fits the training set better than the test set we call this overfitting.
- Overall
    - Example
        - In machine learning, the ideal algorithm has low bias and can accurately model the true relationship (imagine a model fitting a training set that has log10 type shape data). Also, the ideal model will have low variability, by producing consistently predictions across different data sets (imagine a representation of the same model but now with testing data. The model will ideally fit it the same. The sums of squares are about equal between data sets if it has low variability.) An ideal model for your data to create is done by finding the sweet spot between a simple model (think about the straight line fitting the training set, ~high bias) and complex model (think about the squiggly line model fitting the training set perfectly, zero bias)

        - Terminology alert:
            - Three commonly used methods for finding the sweet spot between simple and complicated models are: **regularization**, **boosting**, and **bagging**
- Finance
    - Black–Scholes model
    - Definitions:
        - Risk
            - How do you measure Risk
        - Stocks
        - Portfolio
- Difference between linear and logistic regression
    - https://techdifferences.com/difference-between-linear-and-logistic-regression.html
-

— — — — —

SLANG TERMS/TERMINOLOGY

- Long Data
    - Bunch of columns, little rows
- Wide Data
    - Bunch of rows, little columns

— — — —

MATHEMATICS

- Common Functions
    - Sigmoid
    - Threshold
    - Rectifier
    - Hyperbolic tangent
    - Softmax Function
    - Cross-Entropy
- Common Distributions
    - Gaussian

— — — — —

RANDOMs

- Python
  - Error Handing!
  - OpenCV
  - Looping through files in a directory
  - sparsity (can come up a lot in NLP, we want to avoid it)
  - https://realpython.com/python-modules-packages/
- Watch Later
  - YouTube
    - Image Classifier using VGG16 Model (Great playlist to go through by deeplizard)
      - https://www.youtube.com/watch?v=oDHpqu52soI&list=PLZbbT5o_s2xrwRnXk_yCPtnqqo4_u2YGL&index=13
- Big Data Tools
  - Hadoop
  - Spark
  - H2O
  - CDSW
  - Domino Labs
  - HIVE
  - PIG
  - and unstructured data
  - etc.
  - CPLEX, IBM-DOC & AnyLogic (Bayer mentioned this in a data science job posted)
  - Developing and/or applying linear, mixed integer, stochastic programming to solve demand planning, supply chain, production scheduling (The Goodyear Tire & Rubber Company)
  - Discrete Event Simulation, Factor Analysis, Genetic Algorithms, Bayesian Probability Models, Hidden Markov Models and Sensitivity Analysis. (Cotiviti)
  -
- Matrices
  - scipy.sparse matrices vs numpy matrices?
- Feature Selection
  - Heatmaps
- Leaf-wise vs Level-wise Decision Tree Algorithms
  - Leaf-Wise
    - lightgbm
- XGBoost, Lightgbm, and CatBoost
  - https://www.youtube.com/watch?v=V5158Oug4W8
  - https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db
  - these are are gradient boosting algorithms for decision trees
    - xgboost
      - https://xgboost.readthedocs.io/en/latest/tutorials/index.html

        - loss function term
        - regularization term
            - xgboost has show to work very welling practice
    - LightGBM
        - written by Microsoft
        - much faster than xgboost
        - works about the same
    - CatBoost
        - great if you have a lot of categorical variables
        - uses something called mean target encoding (https://www.youtube.com/watch?v=V5158Oug4W8, see around ~13:00)
- GridSearchCV vs random search
    - learning rate
-
- DecisionTreeRegressor
- BaggingRegressor
- RandomForestRegressor
- ExtraTreesRegressor
- AdaBoostRegressor
- GradientBoostingRegressor
- XGBRegressor
- KerasRegressor
- How to determine how many n_estimators we need for most of the regressors
- Memory Usage Feature in Jupyter Notebook
- Survival Analysis
    - https://stats.idre.ucla.edu/stata/seminars/stata-survival/
- Data Engineering
    - Goals of a Data Engineer
        - Developing data pipelines (most undergrad courses don't teach you how to do this)
            - Taking data from an operational system and moving it to something so it can be used by analysts and data scientists
        - Manage tables and data sets for analysts and data scientists
        - Design with the product in mind
            - What data is being tracked, what questions are the users going to be asking while they are using the product (for example, dashboards)
    - Skills of a Data Engineer
        - Data Modeling
            - Relational Databases
                - Note, databases prefer data in long format
            - Data Warehouses
        - Automation
            - Timing (isolated task/function at a specific time routinely)
            - Dependencies (for example, we want to make sure task/

        table A comes before task/table B)
    - Failures (you want to make sure you capture your failures like a notification system (email, conbon?)
- ETL Development (there is another type called ELT, essentially Data Pipeline, means Extract Transformation Label)
    - For Example, taking data from an operational system such as Workday (an HR system) or MySQL (if its a product), and moving this data into a data warehouse. The old school way is to use SQL Server, Oracle databases, etc as the data warehouse. The more modern way is Hadoop, Redshift, etc.
    - Also in ETL development, data engineers can also be in charged of implementing an analytical data layer on the data to aggregate it so it can be easily fed to dashboards.
- Product Understanding (data is our product, we want to understand what the data scientists want)
    - Dashboards
    - Datasets
- Data engineers develop data pipelines to
    - improve ease of data access
    - produce analytical layers for dashboards
    - clean up data (eliminate duplicate data, etc)
        - we want data scientists to know that every observation they get is that its accurate/valid)
- Tools Used
    - Pipeline Framework (SSIS (GUI, hard to customize with Python), Infromatica, Airflow (AirBnB's version of a data pipeline management system, easy to customize)), Spark
    - SQL
        - It is easier to hire someone who knows SQL instead of map/reduce in Hadoop, Hive, etc
    - Python, Powershell, Linux and/or Java (Java is for map/reduce stuff)
    - Tableau (heavily used, easiest), D3.js (fun and customizable, but painful if you are not Java script fan), SSRS, etc.
- How does a Data Engineer make an impact?
    - Creating optimized pipelines
    - influencing
    - designing
    - maintaining
- Note, Business Intelligence and Data Engineering are now more of the same.
-

- Undersampling vs Oversampling

— — — — —

GENERAL

- Overfitting occurs when there is greater accuracy seen with the training set than the test set
- Anything model that is built with X_train and y_train is considered supervised learning. Any model but with just X_train is considered unsupervised learning.
- Features are column-wise data
- Supervised Learning vs Unsupervised Learning vs Reinforcement Learning (note, I need to un indent this over)
    - The main difference between the two types is that supervised learning is done using a ground truth, or in other words, we have prior knowledge of what the output values for our samples should be.
        - Supervised Learning
            - the goal of supervised learning is to learn a function that, given a sample of data and desired outputs, best approximates the relationship between input and output observable in the data.
                - Classification or Regression
                    - Classification:
                    - Regression:
                    - Common Algorithms:
                        - logistic regression
                        - naive bayes
                        - support vector machines
                        - artificial neural networks
                        - random forests
                - In both regression and classification, the goal is to find specific relationships or structure in the input data that allow us to effectively produce correct output data. Note that “correct” output is determined entirely from the training data, so while we do have a ground truth that our model will assume is true, it is not to say that data labels are always correct in real-world situations. Noisy, or incorrect, data labels will clearly reduce the effectiveness of your model. When conducting supervised learning, the main considerations are model complexity, and the bias-variance tradeoff. Note that both of these are interrelated.
                - When conducting supervised learning, the main considerations are model complexity, and the bias-variance tradeoff. Note that both of these are

interrelated.
    - The bias-variance tradeoff also relates to model generalization. In any model, there is a balance between bias, which is the constant error term, and variance, which is the amount by which the error may vary between different training sets. So, high bias and low variance would be a model that is consistently wrong 20% of the time, whereas a low bias and high variance model would be a model that can be wrong anywhere from 5%-50% of the time, depending on the data used to train it. Note that bias and variance typically move in opposite directions of each other; increasing bias will usually lead to lower variance, and vice versa. When making your model, your specific problem and the nature of your data should allow you to make an informed decision on where to fall on the bias-variance spectrum. Generally, increasing bias (and decreasing variance) results in models with relatively guaranteed baseline levels of performance, which may be critical in certain tasks. Additionally, in order to produce models that generalize well, the variance of your model should scale with the size and complexity of your training data — small, simple data-sets should usually be learned with low-variance models, and large, complex data-sets will often require higher-variance models to fully learn the structure of the data.
- Unsupervised Learning
    - Does not have labeled outputs, so its goal is to infer the natural structure present within a set of data points
    - The most common tasks within unsupervised learning are clustering, representation learning, and density estimation. In all of these cases, we wish to learn the inherent structure of our data without using explicitly-provided labels.
        - Common Algorithms:
            - K-Means Clustering
            - Principal Component Analysis (PCA)
            - Autoencoders
        - Some common use-cases for unsupervised learning are exploratory analysis (I thinkk, this is apriori) and dimensionality reduction (PCA, LDA)
- Reinforcement Learning

- Decision Tree Terminology
  - The very top of the tree is called the “Root Node” or just “The Root”. It represents the entire population or sample, and this further gets divided into two or more homogeneous sets.
  - “Internal/Child Nodes” have arrows pointing to them and away from them
  - “Leaf Nodes” only have arrows pointing to them
  - “Splitting” is dividing the root node/sub node into different parts on the basis of some condition
  - “Pruning” is the opposite of splitting, basically removing unwanted branches from the trees. In other words, pruning is a method of limiting tree depth to reduce overfitting in decision trees. There are two types of pruning:
    - Pre-pruning: A decision tree involves setting the parameters of a decision tree before building it. For example, setting the maximum tree depth, maximum number of terminal nodes, minimum samples for a node split (controls the size of the resultant terminal nodes), maximum number of features
    - Post-pruning: To post-prune, validate the performance of the model on a test set. Afterwards, cut back splits that seem to result from overfitting noise in the training set. Pruning these splits dampens the noise in the data set.
      - Post-pruning may result in overfitting the model and is currently not available in Pythons sklearn, but its available in R.
  - “Branch/SubTree” is formed by splitting the tree/node
  - “Entropy” & “Information Gain”
    - https://homes.cs.washington.edu/~shapiro/EE596/notes/InfoGain.pdf
    - https://www.youtube.com/watch?v=qDcl-FRnwSU&t=235s (great example, 27:00, 34:00)
    - Entropy
      - Common way to measure impurity. This value is used in the information gain formula.
    - Information Gain
      - Measures the reduction in entropy. Decides which attribute should be selected as the decision node. We select the attribute with the maximum gain to be the root node.
    - Also, in other words, which ever feature has the lowest Gini Impurity will be the one that provides the greatest information gain. Therefore, this feature will be selected as the node and so forth for the remaining nodes.
  - Important decision trees can be constructed using binary data, continuous data, and categorical/ranked data:

    - Example with binary data (3 independent vars, 1 dependent var, all binary data (0,1))
        - See @03:25 https://www.youtube.com/watch?v=7VeUPuFGJHk
        - To begin a decision tree, you calculate the Gini impurity using each independent var with the dependent var one at a time and keep track of all the true pos, false pos, false neg, true neg. Which ever feature shows the lowest *total* Gini impurity is the one selected to start the root node. Next, the *subsection* Gini impurity's that were created for the "true pos, false pos" block and "false neg, true neg" block are the splits for that node. Now let's start with going down the left side of the root node (the true pos, false pos) side. We must compare the Gini impurity found in this section to the Gini impurities of the other features (note, these features have to fall under the left side of the of the root node too). If a smaller Gini impurity is found in one of the other features, then it becomes the node for this level of the tree. Therefore, in a sense, it overrides the "true pos, false pos" section of the root node.
    - Example with continuous data (1 independent var (continuous data), 1 dependent var (binary data))
        - See @13:57 https://www.youtube.com/watch?v=7VeUPuFGJHk
        - To begin a decision tree with the independent var as continuous, you first have to sort the column data (eg, 125, 135, 155, and so forth (make sure the dependent vars stay with the appropriate ones)). Then you take an average between each observation (130, 145). From there, you then start calculating the *subsection* Gini Impurities, and then get the *total* Gini impurity. See the link directly above.
    - Example with ranked/categorical data (1 independent var (categorical data), 1 dependent var (binary data))
        - See @15:25 https://www.youtube.com/watch?v=7VeUPuFGJHk
        - Need to revisit
- Advantages: Decision trees are easy to interpret. To build a decision tree requires little data preparation from the user - there is no need to normalize the data
- Disadvantages: Decision trees are likely to overfit noisy data. The probability of overfitting on noise increases as a tree gets deeper.
- Ensembles
    - Creating ensembles involves aggregating the results of different models. Ensemble decision trees are used in bagging and random forests while ensemble regression trees are used in boosting
- Bagging/Bootstrap aggregating
    - Bagging involves creating multiple decision trees each trained on a

different bootstrap sample of the data. Because bootstrapping involves samples with replacement, some of the data in the sample is left out of each tree.
        - Consequently, the decision trees created are made using different samples which solves the problem of overfitting to the training sample. Ensembling decision trees in this way helps reduce the total error because variance of the model continues to decrease with each new tree added without an increase in the bias of the ensemble.
    - Random Forest
        - A bag of decision trees that uses subspace sampling is referred to as a random forest. Only a selection of the features is considered at each node split which decor relates the trees in the forest.
        - Another advantage of random forests is that they have an in-built validation mechanism. Because only a percentage of the data is used for each model, an out-of-bag error of the moles performance can be calculated using the 37% of the sample left out of each model
    - Boosting
        - Boosting involves aggregating a collection of weak learners(regression trees) to form a strong predictor. Ad boosted model is built over time adding a new tree into the model that minimizes the error by previous learners. This is done by fitting the end tree on the residuals of the previous trees.
        - If it isn't clear this far, for many real-world applications a single decision tree is not a preferable classification as it is likely to overfit and generalize very poorly to new examples. However, an ensemble of decision or regression trees minimizes the overfitting disadvantage and these models become stellar, state of the art classification and regression algorithms.

- Model Selection
    - Regression
        - Cross-Validation
            - MAE
            - RSME
    - Classification
        - Confusion Matrix
    - Selecting the best model in scikit-learn using cross-validation
        - What is the drawback of using the train/test split procedure for model evaluation?
        - How does K-fold cross-validation overcome this limitation?
        - How can cross validation be used for selecting tuning parameters, choosing between models, and selecting features?
        - What are some possible improvements to cross-validation?
    - K Fold Validation
        - This is used to over come importing just one random state and

testing it. For example, if cv=10 is set in the cross_val_score model, 10 increments will be tested as the test set with the remaining data being used as the training set each time.

— — — — —

DATA PREPROCESSING

- Terminology
  - feature scaling/normalization/
    - Checking for NaN values in Pandas
      - https://stackoverflow.com/questions/29530232/how-to-check-if-any-value-is-nan-in-a-pandas-dataframe
  - Data Transformation
    - Why should we transform data when it is already clean?
    - Different features in the data set may have values in different ranges. For example, in an employee data set, the range of salary feature may lie from thousands to lakhs but the range of values of age feature will be in 20- 60. That means a column is more weighted compared to other.
      - Two most common methods for normalization:
        - Min-Max
          - Min- Max tries to get the values closer to mean. But when there are outliers in the data which are important and we don't want to loose their impact ,we go with Z score normalization.
        - Z score
      - Skewness of data:
        - According to Wikipedia," In probability theory and statistics, skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean."
        - Generally, if the skewness value lies above +1 or BELOW -1, data is highly skewed. If it lies between +0.5 to -0.5, it is moderately skewed. If the value is 0, then the data is symmetric
        - It is also important to make sure the absolute value of the skewness is greater than twice the std error value (04:45, https://www.youtube.com/watch?v=_c3dVTRIK9c)
        - NOTE, we want to be very careful when sampling data sets if the results are highly skewed. For example, in the fraud analysis I am going through https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets there are 99.83% of cases that are not fraud and 0.17 % of cases that are.

Therefore, if we did sampling with this dataset, we may not get any of the fraud cases in our training set.

- Heat maps
  - are great for seeing correlated variables (<myDataFrameName1>.corr() —> with the Pandas dataframe, this gives a print out of all the correlation coefficients).
- Copy a dataframe
  - temp3 = <myDataFrameName1>.copy() # our dataframe
  -

- Data Frame Types
  - Calling <myDataFrameName1>.info() will show us information related to the Dataframe
    - “Object” are typically string values
- Imputer
  - import Imputes module to handle missing data
- Encoding variables
  - Terminology
    - Same Thing
      - Dummy Encoding (if you are coming from the statistics field)
      - One Hot Encoding (if you are coming from the computer science or electrical engineering field)
  - eg, male and female into 1 and 0
  - eg, France, Spain, and Germany into 0, 2,and 1 (note, we have to be careful of the dummy variable trap here. We must use OneHotEncoder to create ONLY TWO new columns that show only 1 and 0’s). We do not need three columns for each country. We need one minus for some reason. I think this has something to do with the degrees of freedom. We are avoiding what is called the Dummy Variable Trap.
- Feature scaling
  - 5.3.1.3. Scaling data with outliers
  - If your data contains many outliers, scaling using the mean and variance of the data is likely to not work very well. In these cases, you can use robust_scale and RobustScaler as drop-in replacements instead. They use more robust estimates for the center and range of your data. (see https://scikit-learn.org/stable/modules/preprocessing.html#scaling-data-with-outliers)
  - Feature extraction is very different from Feature selection: the former (feature extraction) consists in transforming arbitrary data, such as text or images, into numerical features usable for machine learning. The latter is a machine learning technique applied on these features.
  - Examples of Algorithms where Feature Scaling matters
    - K-Means uses the Euclidean distance measure here feature scaling matters.
    - K-Nearest-Neighbours also require feature scaling.
    - Principal Component Analysis (PCA): Tries to get the feature with

maximum variance, here too feature scaling is required.
    - Gradient Descent: Calculation speed increase as Theta calculation becomes faster after feature scaling.
    - Note: Naive Bayes, Linear Discriminant Analysis, and Tree-Based models are not affected by feature scaling. In Short, any Algorithm which is Not Distance based is Not affected by Feature Scaling.

- Good Reads
    - https://stats.stackexchange.com/questions/189652/is-it-a-good-practice-to-always-scale-normalize-data-for-machine-learning
    - https://stats.stackexchange.com/questions/342140/standardization-of-continuous-variables-in-binary-logistic-regression (eg, 2 regressors are continuous and 2 are binary)
    - https://datascience.stackexchange.com/questions/20237/why-do-we-convert-skewed-data-into-a-normal-distribution
- Parametric and Nonparametric: Demystifying the Terms
    - Parametric
        - We have to normally distributed datasets (not positively or negatively skewed)
        - Common transformations of this data include square root, cube root, and log10.
            - The cube root transformation involves converting x to x^(1/3). This is a fairly strong transformation with a substantial effect on distribution shape: but is weaker than the logarithm. It can be applied to negative and zero values too. Negatively skewed data.
            - The square root transformation, x^2 can only be applied to positive values only. Hence, observe the values of column before applying.
            - The logarithm transformation , x to log base 10 of x, or x to log base e of x (ln x), or x to log base 2 of x, is a strong transformation and can be used to reduce right skewness (positively skewed).
        - If tail is on the right as that of the second image in the figure, it is right skewed data. It is also called positive skewed data.
            - In order to fix a positively skewed distribution using a log10 distribution, the following assumptions have to be met:
                - no negative values, no zeroes, and the positively skewed (if you have negative values or zeros, see around ~08:00 mins, https://www.youtube.com/watch?v=_c3dVTRIK9c)
        - If the tail is to the left of data, then it is called left skewed data. It is also called negatively skewed data.
            - Similarly, to fix a negatively skewed distribution using a log10 distribution, the same conditions have to be met:
                - but this time we call the log10 function as
                    - log10(max(x) - 1 + x)

-

- Resolving outliers
  - Anomaly Detection (best seen from boxplot)
    - Interquatile Range Method
- Aggregation of data

— — — — —

CLASSIFICATION

- Decision Tree Classification
  - graphical representation of all the possible solutions to a decision
  - decisions are based on some conditions
  - decisions made can be easily explained
    - Example
      - Independent variables: “Am I hungry?” “Do I have $25?”
        - Data is in binary form (yes or no)
      - Dependent variable: “Outcome”
        - Data is “Go to restaurant” “Buy a hamburger” “Go to sleep” (this is a multiple classification problem)
  - NOTE: When building a decision tree you first have to individually compare each independent variable to the dependent variable in order to determine which independent variable will begin the decision tree (classification problem, yes or no for all results for the independent and dependent variable, you keep track of which independent variables that has the lowest “Gini Impurity” (aka the one that is most pure) and you begin with that one at the top of the tree.. See https://www.youtube.com/watch?v=7VeUPuFGJHk&list=PLsCzljdLz2iR66dIIpz9E9veNj7wijPS7 . I thinkkk you can also look at the true pos, false pos, true neg, false neg.
  - Important: Decision trees for classification use Gini Impurity and/or Information Gain. Decision trees for regression use Standard Deviation Reduction (see http://saedsayad.com/decision_tree_reg.htm).
- Random Forest Classification
  - Builds multiple decision trees and merges them together
  - more accurate and stable prediction
  - random decision forests correct for decision trees’ habit of overfitting to their training set
  - trained with the “bagging” method

- Naïve Bayes Classification
  - Classification technique based on Bayes’ Theorem
  - Assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature

- K Nearest Neighbors Classifier
  - Stores all the available cases and classifies new cases based on a

similarity measure
    - The “K” is the KNN algorithm is the nearest neighbors we wish to take a vote from
        - Example:
            - Imagine a scatter plot with real training data on it representing 5 classes. Each dot on the scatter plot is a color. Generally, the color dots in this training set data are grouped by similar color dots. So visually, you can see five classes. Now, I think the KNN algorithm builds the model by looking at super small segments of the plot and starts a computational radius going outward from each segment. So if K is specified to be “3.” The computational radius will stop expanding for each segment when it hits three dots of the same color and will give that segment of the plot the appropriate color.
- from lightgbm import LightGBMClassifier

—————
REGRESSION

- Decision Tree Regression
    - For example, when visualizing two independent variables on a scatter plot, a decision tree algorithm “splits” your dataset based off of information entropy (need to look up more) and based off of these splits is how the tree is made.
        - https://www.saedsayad.com/decision_tree_reg.htm (example)
    - Important: Decision trees for classification use Gini Impurity and/or Information Gain. Decision trees for regression use Standard Deviation Reduction (see http://saedsayad.com/decision_tree_reg.htm).

- Random Forest Regression
    - a form of ensemble learning, ensemble learning is when you take the same algorithm multiple times to make your model much more powerful than the original. Gradient boosting is a form of ensemble learning.
    - Not very sensitive to hyperparameters (need to verify still)
- Support Vector Regression
    - Great video —> https://www.youtube.com/watch?v=Y6RRHw9uN9o&t=366s
        - Dog and cat example. X axis there is snout length and on the Y axis there is ear length. SVR algorithm looks at the outliers within the data set and applies a linear linear / nonlinear line according to them. These outliers are none as support vectors. Note, in the example in the video the data is separable between the cat and dog classes.
        - When using Support Vector algorithms it is often helpful to map your vectors to a higher dimension in order to fully (better?) separate the data. However, this is often computationally

expensive.
        - See Kernel function or Kernel Trick. The dot product can be computed to project the vectors into a higher dimension.
- Logistic Regression
    - Note, logistic regression is probably better placed under CLASSIFICATION.
    - Great explanation of how to interpret the coefficients in a logistic or linear regression model (see ~02:48:00, https://www.youtube.com/watch?v=5rNu16O3YNE&list=PLsCzljdLz2iR9pnDlZkTqSvbUZqZenhcK&index=5)
    - Has to do only with probability. Think about example of where a bank decides they should allow you to get a loan or not depending on your credit score, income, age, etc..
    - Or think about an image data set of digits where the target variable is 0-9. You can do classification binary or multi class problems with LogisticRegression.
    - Not very sensitive to hyperparameters (need to verify still)
- Linear Regression
    - Great explanation of how to interpret the coefficients in a logistic or linear regression model (see ~02:48:00, https://www.youtube.com/watch?v=5rNu16O3YNE&list=PLsCzljdLz2iR9pnDlZkTqSvbUZqZenhcK&index=5)
    -

— — — — —

CLUSTERING:

- Hierarchial Clustering
- Clustering
    - K-Means
        - Very good for discovering categories or groups in your data that you may of not been able to recognize yourself.
        - Can work for multiple dimension problems
        - How it works (imagine a 2D scatter plot with a bunch of gray markers and we want to separate our data into 3 clusters (red, green, and blue)):
            - https://www.udemy.com/course/machinelearning/learn/lecture/5714416#overview (great example)
            - Step 1. Choose the number K of clusters
            - Step 2. Select **at random** K points. The centroids (they don't not necessarily have to be from or around your dataset).
            - Step 3. Assign each data point on the plot to the closest centroid that will in result form K clusters (generally in the form of Euclidean distance (there are other types, need to look up))
            - Step 4. Compute and place the new centroid of each cluster.

        - Step 5. Reassign each data point to the new closest centroid. If any reassignment took place, go back to Step 4, otherwise go to finish.
        - This is an iterative process.
    - The algorithm runs quickly by drawing a straight line connecting to each centroid and then drawing a perpendicular line from that line easily separate the data set and see which data points are closest (well, Im not sure if that's how the algorithm actually runs, but that is a great visual way to think about it, this would be really good to show on a technical presentation).

MACHINE LEARNING ALGORITHMS

- A/B Testing
    - Example, comparing which landing page of a website performs best.
    - A/A tests should be conducted first to make sure there's nothing wrong on the backend such as if the data is messy, sampling problem because not randomizing properly, or too much noise.
- KNeighborsRegressor (aka KNN)
    - https://scikit-learn.org/stable/auto_examples/neighbors/plot_regression.html#sphx-glr-auto-examples-neighbors-plot-regression-py
    - https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm (great explanation of the weights parameter)
- DecisionTreeRegressor (aka CART)
- SVM
    - C-Parameter:
        - https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel see Kent Munthe Caspersen answer
            - "In general, having few training instances and many attributes make it easier to make a linear separation of the data."
- xgboost
    - https://xgboost.readthedocs.io/en/latest/tutorials/index.html

— — — — —

ASSOCIATED RULE LEARNING

- Apriori Algorithm (think of grocery cart example —> diapers and beer relationship)
- FP Growth Model

— — — — —
REINFORCEMENT LEARNING

- Upper Confidence Bound Algorithm (REVISIT 2019-08-10 21:33:37, DONT UNDERSTAND FULLY)
  - Multi Armed Bandit Problem
- Thompson Sampling
  - Multi Armed Bandit Problem

- Important Terminology
  - Explore vs. Exploitation

— — — — —
METRICS

- Use the sklearn.metrics to create a confusion matrix to analyze true negatives, false negatives, true positives, and false positives
- See the section Model Selection & Boosting below for a better way to look at the performance of your models (GridSearchCV)
- https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/
  - Lower values of RMSE indicate better fit.

— — — — —
NATURAL LANGUAGE PROCESSING

- rule learning
- sparsity
- sparse matrices

— — — — —
DEEP LEARNING

- Important Terminology
  - Output values of the neural network can either be continuous (eg, price), binary (eg, yes/no), or categorical
  - weighted sum, activation function (sigmoid, threshold, rectifier, hyperbolic tangent)
  - Think about churn modeling problem (trying to predict which customers will leave the bank or not)

— — — — —
ARTIFICIAL NEURAL NETWORKS

- NEURAL NETWORKS
  - Additional Reading
    - https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications

  - How do neural networks learn?
    - You can either hard code them or let them learn on their own. We are always trying to let the models learn on their own.
    - Basically the only thing we have control over in the programming are the weights
    - Batch gradient descent (think of the smooth parabolic cost function curve with the ball bouncing back and forth trying to find the minimum)
    - Stochastic gradient descent (this method is used to find the global minimum of a rough shaped parabolic cost function curve) - this method is faster than batch (there is also a mini-batch method)
      - https://iamtrask.github.io/2015/07/27/python-network-part2/

  - Important Terminology
    - Back propagation (the process of optimizing the cost function (aka adjusting the weights to find optimal value to agree with the known value))
    - Normally the rectifier function is used for the development of the hidden layers and for the output layer the sigmoid function is used.

- CONVOLUTIONAL NEURAL NETWORKS

  - Think about the following images: duck or rabbit, man looking at you or away from you, and image of the girl with duplicate facial features (eyes, nose, mouth, eyebrows)
  - Steps —> Convolution, Max Pooling, Flattening, and Full Connection
    - Convolution: The key take away of what convolution is, is to find features in your image by using a feature detector (think of a feature detector as a specific feature that distinguishes an image from the other images) and putting them into a feature map. Then from the feature map, it preserves the spatial relationships between pixels
    - Convolution (Cont.): ReLU Layer —> we introduce this Rectifier Linear Units algorithm to break up linearity.
      - Good read on convolution filters: http://www.roborealm.com/help/Convolution.php
    - Max Pooling: Looking for a specific feature of an image. For example, the tears/marks on a cheetah. It is one feature that makes the cheetah very distinct. Note, there are other forms of pooling (mean pooling, sum pooling, etc). In the example I was following on Udemy, the instructor mentioned that max pooling eliminates the need of unnecessary features. In the cheetah example, we were

able to get rid of 75% of the information. Max pooling allows us to instantly see specific distinct regions. In summary, we are able to preserve distinct features, inducing spatial variance, and reducing the size of information and parameters (this helps us prevent over fitting)
        - http://scs.ryerson.ca/~aharley/vis/conv/flat.html
        - http://ais.uni-bonn.de/papers/icann2010_maxpool.pdf
        - http://scs.ryerson.ca/~aharley/vis/conv/
        - http://www.cs.cmu.edu/~aharley/
    - Flattening: For example, making a 3x3 feature map display vertically 1-9
    - Full Connection: The middle/inner hidden layers (these develop attributes that describe each output). The final inner layer nodes each pass on a vote/probability from 0.0 to 1.0 on whether the image presented is a cat or dog (for example)
    - SUMMARY: https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html
- Softmax & Cross-Entropy (Loss Function)
    - These functions behave similar to how the the mean square error (MSE) / cost function works when linear fitting
    - Classification Error, Mean Squared Error, Cross-Entropy
        - Cross-Entropy is typically the best value to look at to evaluate your neural network (note, cross-entropy involves a logarithmic function. Also, cross-entropy is only the best for classification problems)
- Image Preprocessing
    - https://keras.io/preprocessing/image/
    - To avoid overfitting, data augmentation is used.

————
DIMENSIONALITY REDUCTION

- Two Types
    - Feature Selection
        - Backward elimination, forward selection, bidirectional elimination, score comparison, and more (these topics were covered in the Regression lecture)
    - Feature Extraction
        - Principal Component Analysis (PCA)
            - see —> https://www.youtube.com/watch?v=_UVHneBUBW0 and https://www.youtube.com/watch?v=FgakZw6K1QQ (great) and https://www.youtube.com/watch?v=HMOI_lkzW08 (NOTE: I think the cells are considered the classes/dependent variables and the genes are considered as the independent variables)
            - PCA is a form of data preprocessing I thinkk. It has nothing to do with the model/classifier. PCA is a method of

compressing a lot of data into something that captures the essence of the original data. PCA reduces dimensions by focusing on the genes with the most variation.The business example I followed in the Udemy course involved analyzing 178 data observations (12 independent variables, 1 dependent variable (3 classes, each class represents a different wine)). The business was trying to predict based off the ingredients what type of wine the drink should be classified as. PCA was applied to reduce the independent variables to the ones that show the most variance. This also allows us to visualize the data in 2D or 3D better.

- Linear Discriminant Analysis (LDA)
  - see —> https://www.youtube.com/watch?v=azXCzI57Yfc (great)
  - LDA is very similar to PCA. It is also used in the preprocessing step for pattern classification. Its goal is to project a dataset onto a lower-dimensional space. However, LDA differs because in addition to finding the component axises with LDA. In other words, we are interested in maximizing the separability between the two (or more) groups/categories so we can make the best decisions. We are interested in the axes that maximize the separation between classes (think of the diagram that shows gaussian distributions on both the x-axis and y-axis). The goal of LDA is to project a feature space (a dataset n-dimensional samples) onto a small subspace k(where k is less than or equal to n-1) while maintaining the class-discriminatory information.
    - Both PCA and LDA are linear transformation techniques used for dimensional reduction. PCA is described as unsupervised but LDA is supervised because of the relation to the dependent variable.
    - PCA: Component axes that maximize the variance
    - LDA: Maximizing the component axes for class-separation.

- Kernel PCA (this is a nonlinear reduction strategy)
  - This is for nonlinear problems. Note, the same linear model can be used with the normal PCA object but this time we must use an additional argument "rbf.' This accounts for the nonlinearity.

————
MODEL SELECTION & BOOSTING

- Two Types
  - Cross Vaidation (multiple methods)

        - https://www.youtube.com/watch?v=L_dQrZZjGDg
    - k-Fold Cross Validation
        - https://www.youtube.com/watch?v=gJo0uNL-5Qw
    - Grid Search
        - This technique involves grid search to find optimal hyperparameters. Note, hyperparameters are the parameters you specify when you build your model.
- Boosting
    - XGBoost
        - It is the most powerful implementation of gradient boosting
        - When using XGBoost, you don't have to use feature scaling,
- Mean Absolute Error (MAE) vs Root mean squared error (RMSE)
    - For continuous dependent variables only
        - great read
        - https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

———————

PLOTS

- Violin plots
    - https://www.youtube.com/watch?v=M6Nu59Fsyyw
- Line plot
    - ax = sns.lineplot(x="pickup_hour", y="Trip_distance", legend="full" , data=summary_wdays_avg_duration, estimator=None, hue="day_of_week")

———————

STATISTICS

- Pearson Coefficient
- Skewness
    - measures the symmetry of a distribution
- Kurtosis
    - measures how peaked or flat a distribution is
- Standard Error
    - https://www.investopedia.com/terms/s/standard-error.asp
- Standard Deviation
    - https://www.investopedia.com/terms/s/standarddeviation.asp
- Homoscedasticity
    - https://datascience.stackexchange.com/questions/20237/why-do-we-convert-skewed-data-into-a-normal-distribution
        - The errors your model commits should have the same variance, i.e. you should ensure the linear regression does not make small errors for low values of X and big errors for higher values of X. In other

words, the difference between what you predict $\hat{Y}$ and the true values Y should be constant. You can ensure that by making sure that Y follows a Gaussian distribution. (The proof is highly mathematical.)

—————

PYTHON

- Commonly used functions
  - list()
  - set()
    - https://realpython.com/python-sets/
      - see the “Operators vs Methods” section (the or operator)
      - that structure comes up a lot while getting unique elements for encoding with both a training and test set
  - len()
  - range()
  - append()
  - apply()
- Syntax
  - ebola_long["variable"].str.split("_", expand=True)
    - to view the syntax better, you can rewrite as
      - (ebola_long["variable’]
      - .str
      - .split("_", expand=True)
- Functions
- Libraries
  - tflearn
  - keras
  - numba
    - Tries to make all the computations numpy does really fast
  - seaborn
  - matplotlib.pyplot
    - Anatomy of a figure
      - https://matplotlib.org/3.1.1/gallery/showcase/anatomy.html

————

NUMPY

- Common functions
  - np.expm1()

- np.linspace

— — — — —

PANDAS

- Dataframe

- Dataframe
  - Terminology
    - Mean the same thing
      - Example 1
        - <myDataFrameName1>[“<myFeatureName1>”] and <myDataFrameName1>.<myFeatureName1>
  - There are three parts to a dataframe
    - <myDataFrameName1>.columns (feature names)
    - <myDataFrameName1>.index (row names)
    - <myDataFrameName1>.values (body values, this is good for extracting data from the Dataframe to use the data in another library that may just use numpy arrays)
  - Dataframe information
    - <myDataFrameName1>.info()
      - “object” are typically string values
    - <myDataFrameName1>.types
    - If you extract a single column of data from a DataFrame set that new data then becomes a Series object
      -
  - aggregate()

— — — — —

Q&A STACKEXCHANGE

https://datascience.stackexchange.com/questions/26776/how-to-calculate-ideal-decision-tree-depth-without-overfitting

https://stackoverflow.com/questions/34162443/why-do-many-examples-use-fig-ax-plt-subplots-in-matplotlib-pyplot-python

https://stackoverflow.com/questions/3584805/in-matplotlib-what-does-the-argument-mean-in-fig-add-subplot111

https://www.geeksforgeeks.org/python-pandas-dataframe-ix/

https://stackoverflow.com/questions/31593201/how-are-iloc-ix-and-loc-different

https://stats.stackexchange.com/questions/56302/what-are-good-rmse-values

———
GREAT YOUTUBE CHANNELS

- Data School (https://www.youtube.com/channel/UCnVzApLJE2ljPZSeQylSEyg)
- codebasics
  - https://www.youtube.com/watch?v=gJo0uNL-5Qw (Machine Learning Tutorial Python 12 - K Fold Cross Validation)
  - https://github.com/codebasics/py (see all repositories)

---———
MEDIUM

- https://towardsdatascience.com/understanding-input-and-output-shapes-in-convolution-network-keras-f143923d56ca

———
INTERVIEW

———
PROBLEM SOLVING TIPS

- Classification Problem
  - Typically its good to test logistic regression vs random forest classifier to see which one performs best. If the random forest classifier performs best, then we should look into boosting with either xgboost, lightgbm, or catboost

Overall process (from Allstate Severity Example):

- Data statistics
  - Shape
  - Peek
  - Description
  - Skew
- Transformation
  - Correction of skew
- Data Interaction

        - Correlation
        - Scatter plot
- Data Visualization
    - Box and density plots
    - Grouping of one hot encoded attributes
- Data Preparation
    - One hot encoding of categorical data
    - Train-test split
- Evaluation, Prediction, and Analysis
    - Linear Regression (Linear algo)
    - Ridge Regression (Linear algo)
    - LASSO Linear Regression (Linear algo)
    - Elastic Net Regression (Linear algo)
    - KNN (non-linear algo)
    - CART (non-linear algo)
    - SVM (Non-linear algo)
    - Bagged Decision Trees (Bagging)
    - Random Forest (Bagging)
    - Extra Trees (Bagging)
    - AdaBoost (Boosting)
    - Stochastic Gradient Boosting (Boosting)
    - MLP (Deep Learning)
    - XGBoost
    - Make Predictions

######################
CODE SNIPPETS
        -
- Jupyter Notebook / Markdown
    - Github
        - This is a self-exercise notebook that was created while following along the Natural Language Processing section in the following Udemy course by SuperDataScience:
        - This is a self-exercise notebook that was created while following along in the following YouTube video by codebasics:
- Python
    - Important
        - There seems to be a lot of issues with copying snippets over from TextEdit to Jupyter. It's something to do with single quotes. Change them to double quotes.
        - Snippet Template Names
            - <myDataFrameName1>
            - <>
            - <myFeatureName1>

        - <myFeatureName1_categorical>
        - <myFeatureName1_continuous>
    - <myVarName1>
    - <myModelName1>
    - <myXTestName1>
    - <myFunctionName1>
    - <myInt1>
    - <myIndexName1>
    -
    -
    -
    -
    -
- Pandas
    - Info
        - pandas.__version__ # this is really useful when trying to get help from Google or StackOverflow
        - [] # single brackets are for specifying if you are in rows or columns
        - [[]] # inner brackets are typically used when you want to implement a list/multiple things
            - [["year', "date']] # example
        - <myDataFrameName1>[“<myFeatureName1>”] is the same thing as <myDataFrameName1>.<myFeatureName1>
    - Set Options
        - pd.set_option('display.max_columns', 999) # allows us to see all columns in Jupyter notebook
    - Converting Data Types
        - <myDataFrameName1>[<myFeatureNameList1>] = <myDataFrameName1>[<myFeatureNameList1>].apply(pd.to_numeric, errors='coerce')
- Commonly used
    - timeit example
        - %%timeit # line 1 in Jupyter cell
        - <myFunctionName1>(<myDataFrameName1>[“<myFeatureName1>”].values, <myDataFrameName1>[“<myFeatureName2>”].values) # line 2 in Jupyter cell
    - print(train_X.shape)
    - print(type(train_X.shape))
- Commonly used libraries
    - import pandas as pd
    - import numpy as np
    - import matplotlib.pyplot as plt
    - import seaborn as sns

-
- import warnings
- warnings.filterwarnings('ignore')
-
- import tensorflow as tf
- tf.logging.set_verbosity(tf.logging.ERROR) # To surpress any TensorFlow warnings.
-
-
-
- from sklearn.preprocessing import LabelEncoder, OneHotEncoder
- from sklearn.model_selection import train_test_split
- from sklearn.metrics import mean_absolute_error
-
- from sklearn.linear_model import LinearRegression
- from sklearn.neighbors import KNeighborsRegressor # KNN (Non-linear Algo)
- from sklearn.tree import DecisionTreeRegressor # CART
- from sklearn.svm import SVR
- from sklearn.ensemble import BaggingRegressor
- from sklearn.ensemble import RandomForestRegressor
- from sklearn.ensemble import ExtraTreesRegressor
- from sklearn.ensemble import AdaBoostRegressor
- from sklearn.ensemble import GradientBoostingRegressor
- from xgboost import XGBRegressor
-
-
- import re # used often for NLP, removes and replaces characters
-
-
-
-
- from keras.models import Sequential
- from keras.layers import Convolution2D
- from keras.layers import MaxPooling2D
- from keras.layers import Flatten
- from keras.layers import Dense
- from keras.preprocessing.image import ImageDataGenerator
  - https://keras.io/preprocessing/image/
-
- #Import libraries for deep learning
- from keras.wrappers.scikit_learn import KerasRegressor
- from keras.models import Sequential
- from keras.layers import Dense
-
- # Optimize using dropout and decay
- from keras.optimizers import SGD

- from keras.layers import Dropout
- from keras.constraints import maxnorm
-
-
-
-
-
-
-
-
-
-
- Encoding
  - pandas
    - Example 1
      -
      - <myDataFrameName1> = sns.load_dataset("tips")
      - <myDataFrameName1>.head()
      - <myDataFrameName2> = pd.get_dummies(<myDataFrameName1>, drop_first=True)
      - <myDataFrameName2>.head()
      -
      -
      -
      -
      -
  - sklearn
    - Example 1
      -
      - from sklearn.preprocessing import LabelEncoder, OneHotEncoder
      -
      - <myDataFrameName1> = pd.read_csv("train.csv")
      - <myDataFrameName2> = pd.read_csv("test.csv")
      -
      - cols = <myDataFrameName1>.columns
      - split = <myIntOfCategoricalColumnsToEncode>
      -
      -
      -
      - labels = []
      -
      - # Making sure we account for all of the unique variables that show up in both the training and test set provided.
      - # For instance, this ensures we dont run into any

```
        unforeseen variables when going from the training set
        to test set.
-       for i in range(0, split):
-           train = <myDataFrameName1>[cols[i]].unique()
-           test = <myDataFrameName2>[cols[i]].unique()
-           labels.append(list(set(train) | set(test))) #note the
        OR operator!  (See how sets work https://
        realpython.com/python-sets/)
-
-       print("labels %s" % labels)
-
-
-
-
-
-       cats = []
-
-       for i in range(0, split):
-           label_encoder = LabelEncoder()  # Label Encode
-           label_encoder.fit(labels[i])
-           feature =
        label_encoder.transform(dataset_train.iloc[:,i])
-           feature = feature.reshape(dataset_train.shape[0],
        1)
-           onehot_encoder =
        OneHotEncoder(sparse=False,n_values=len(labels[i])
        )        # One Hot Encode
-           feature = onehot_encoder.fit_transform(feature)
-           cats.append(feature)
-
-
-       print("################")
-       print("List of 1D array of cats--> %s" % cats)
-       print("################")
-
-       # Make a 2D array from a list of 1D arrays
-       encoded_cats = np.column_stack(cats)
-
-       print("################")
-       print("List of 2D array of cats--> %s" % encoded_cats)
-       print("################")
-
-       print("################")
-       print(encoded_cats.shape)  # These are the
        categorical variables we just encoded. Now, we just
        need to concatenate it with the continuous vars.
-       print("################")
```

-
- #Concatenate encoded attributes with continuous attributes
- <myDataFrameName3> = np.concatenate((encoded_cats,dataset_train.iloc[:,split:].values),axis=1)
-
- del cats
- del dataset_train
- del encoded_cats
-

- Data Set Details
  - <myDataFrameName1>.info() # data info. ex: datatypes, size etc.
  - <myDataFrameName1>.dtypes
  - <myDataFrameName1>.describe()
  - <myDataFrameName1>.skew() # Values close to 0 show less skew.
  - <myDataFrameName1>.shape
    - Column Details
      - <myVarName1> = <myDataFrameName1>["<myFeatureName1>'].min() # minimum trip distance, same for max()
      - <myVarName2> = <myDataFrameName1>["<myFeatureName1>'].value_counts() # counts unique occurrences
      - <myVarName2>.head(10)
  - Searching for NaN Values
    - https://stackoverflow.com/questions/29530232/how-to-check-if-any-value-is-nan-in-a-pandas-dataframe
      - <myDataFrameName1>.isnull().sum() (searches for which features contain NaN values)
  - Renaming columns
    - <myDataFrameName1> = <myDataFrameName1>.rename(columns={"<myOldName1>': "<myNewName1>', "<myOldName2>': "<myNewName2>'})
  - To see a random sample of data
    - <myDataFrameName1>.sample(30)
  - Return names of columns where a certain character shows up
    -
      <myDataFrameName1>.columns[<myDataFrameName1>.isin(['?']).any()]
    -
    -
- Shuffling the DataFrame

        - <myDataFrameName1> = <myDataFrameName1>.sample(frac=1).reset_index(drop=True)
- Reading In Data
    - CSV
        - Example 1
            - df = pd.read_csv("green_tripdata_2015-09.csv", low_memory=False, parse_dates=["lpep_pickup_datetime"]) # the parse_dates argument is turning "lpep_pickup_datetime" from an object type into a datetime64 type
    - TSV
        - Example 1
            - dataset = pd.read_csv("Restaurant_Reviews.tsv", delimiter="\t")
    -
- Converting data types
    - series and numpy array
        - <myDataFrameName1>[“<myFeatureName1>”] # this gives a series
        - <myDataFrameName1>[“<myFeatureName1>”].values # this gives a numpy array
- Creating a DataFrame
    - Manually
        - <myDataFrameName1> = pd.DataFrame({
        - "<myFeatureName1>": [10, 20, 30],
        - "<myFeatureName2>": [20, 30, 40]
        - })
- Operations on a DataFrame
    - By Feature (this is called Broadcasting)
        - <myDataFrameName1>[“<myFeatureName1>”] ** 2 # this squares every row
    - Adding rows
        - <myDataFrameName1>["<myFeatureName1>"] + <myDataFrameName1>["<myFeatureName2>"]
- Copy a data frame
    - <myDataFrameName2> = <myDataFrameName1>.copy() # our dataframe
- Subsetting a data frame
    - By columns
        - Into a series
            - <myDataFrameName2> = <myDataFrameName1>["<myFeatureName1>’]
            - <myDataFrameName2>.head()
        - Into a dataframe
            - <myDataFrameName2> = <myDataFrameName1>[["<myFeatureName1>",

"<myFeatureName2>", "<myFeatureName3>"]]
        - <myDataFrameName2>.head() # note if we did a single column it would be a series
- By rows
    - Example 1
        - <myDataFrameName1>.loc[[2,0]] # label based indexing, note the algorithm isn't actually looking for the index. It is doing string matching instead.
        - loc is typically used
    - Example 2
        - <myDataFrameName1>.iloc[[0, 1, -1]] # another example of positional indexing, note this prints the first two rows and the last row of the data frame
- Range Selection
    - By Columns
        - Example 1
            - <myDataFrameName1>[["<myFeatureName1>","<myFeatureName2>"]]
        - Example 2
            - # Get the column names
            - <myColumnNames1> = <myDataFrameName1>.columns[<myIndexInt1>:<myIndexInt2>]
            - # Then get the data columns
            - <myDataFrameName1>[<myColumnNames1>]
    - By Rows
        - Example 1
            - <myDataFrameName1>.iloc[<myIndexInt1>:<myIndexInt2>]
    - By Columns and Rows
        - Example 1
            - <myIndexName1> = <myDataFrameName1>.columns[<myInt1>:<myInt2>] # returns an Index of column names
            - <myDataFrameSeriesName1> = <myDataFrameName1>[<myIndexName1>] # returns a data frame series
            - <myDataFrameName2> = <myDataFrameSeriesName1>.iloc[<myInt3>:<myInt4>] # returns data frame subsetted by rows and columns
        - Example 2
            - <myDataFrameName2> = <myDataFrameName1>[<myDataFrameName

1>.columns[<myInt1>:<myInt2>]].iloc[<myInt3>:<myInt4>]

- Filtering dataframe
  - df.loc[(df[“smoker"]=="No”)&(df["total_bill"] >= 10)]
    # Filter rows by smoker == "No" and total_bill >= 10
  - df (["smoker","day","time"])["total_bill"].mean()
    # What is the average total_bill for each value of smoker, day, and time?
  - df.groupby(["smoker","day","time"])["total_bill"].mean().reset_index()
    # To get back to a Dataframe
  - df.loc[df["year"]==1967,["year", "pop"]]
  - df.loc[(df["year"]==1967) & (df["pop"] > 1_000_000),["year", "pop"]]
    # Note: Python has a neat feature where it ignores underscores in integers to help visualize.
  - df.loc[(df["year"]==1967) | (df["pop"] > 1_000_000),["year", "pop"]]
  - df[billboard_melt["track"] == "Loser”]
  - df.groupby(["year"])["lifeExp"].mean()
    - The aggregate function in the numpy library can do the same thing
      - df.groupby(["year"])["lifeExp"].agg(np.mean)
      - df.groupby(["year"])["lifeExp"].agg(np.std)
      - df.groupby(["year", "continent"])[["lifeExp", "gdpPercap"]].agg(np.mean) # note, the created Dataframe becomes wonky when using a list
- Fixing Skewed Data
  - #log1p function applies log(1+x) to all elements of the column
  - <myDataFrame1>[“<myFeatureName1>”] = np.log1p(<myDataFrame1>[“<myFeatureName1>”])
- Unskewing Skewed Data
  - <myVarName1> = np.expm1(<myModelName1>.predict(<myXTest1>))
  -
  -
  -
- Tidy data (cleaning data)
  - http://vita.had.co.nz/papers/tidy-data.pdf (first three sections are a good read)
    - Criteria 1 (what we don"t want): Column headers are values, not variable names
      - aka going from wide data to long data
        - pew_long = pd.melt(pew, id_vars="religion")
          # doesn’t touch the religion column, and condenses the rest of the data frame into a “variable” and “value” column
        - pew_long = pd.melt(pew, id_vars="religion", var_name="income", value_name="count") # renames the variable and value heading.

        - billboard_melt = pd.melt(billboard, id_vars=["year","artist","track","time","date.entered"], var_name="week", value_name="rating") # many column example
        - billboard_melt = pd.melt(billboard, id_vars=["year","artist","track","time","date.entered"], var_name="week", value_name="rating").groupby("artist")["rating'].mean()
        - billboard_melt = pd.melt(billboard, id_vars=["year","artist","track","time","date.entered"], var_name="week", value_name="rating").groupby("artist")["rating'].mean().reset_index()
- Criteria 2 (what we don"t want): Multiple variables are stored in one column. #random keywords: parse
    - variable_split = ebola_long["variable"].str.split("_") # Note, the ".str." is called an accessor. There is a commonly used ".dt." accessor too.
    - ebola_long["status"] = variable_split.str.get(0)
    - ebola_long["country"] = variable_split.str.get(1)
        - or, you can do the above code all in one line with
            - ebola_long[["status", "country"]] = (ebola_long["cd_country"].str.split("_", expand=True)) #this is also an example of concatenating
- Criteria 3 (what we don"t want): Variables are stored in both rows and columns
    - Note, # if we see a lot of rows with repeated values, this is a symptom of this type of problem.
    - weather_melt = pd.melt(weather, id_vars=["id","year", "month", "element"], var_name="day", value_name="temp")
    - weather_tidy = weather_melt.pivot_table(index=["id","year", "month","day"], columns="element",values="temp") # note, pivot_table is the opposite of melt. Also, by default, this drops NaN values.
    - weather_tidy
    - weather_tidy.reset_index()
- Criteria 4 (what we don"t want): Multiple types of observational units are stored in the same table / This is also called "normalization"
    - billboard_melt.loc[billboard_melt["track"] == "Loser"]
    - billboard_songs = billboard_melt[["year", "artist", "track", "time"]]

            - billboard_songs = billboard_songs.drop_duplicates()
            - billboard_songs["id"] = range(len(billboard_songs))
            - billboard_ratings = billboard_melt.merge(billboard_songs, on = ["year", "artist", "track", "time"]) # merges billboard_melt (left) and billboard_songs (right)
            - billboard_ratings = billboard_ratings[["id", "date.entered","week","rating"]]
        - Criteria 5 (what we don't want): A single observational unit is stored in multiple tables
- Joining Tables/Datasets
    - billboard_songs["id"] = range(len(billboard_songs))
    - billboard_ratings = billboard_melt.merge(billboard_songs, on = ["year", "artist", "track", "time']) # merges billboard_melt (left) and billboard_songs (right)
    - billboard_ratings = billboard_ratings[["id", "date.entered","week","rating"]]
    - billboard_ratings.head()
- Saving to csv
    - billboard_songs.to_csv("billboard_songs.csv", index=False)
- Concatenating (similar to joining)
    - X = np.concatenate((X_train,X_test), axis=0)
- Functions
    - We use what is called an assert statement to test a function. This is the basis of unit testing.
        - Example
            - assert my_sq(4) == 16 # if you run this code nothing happens, but if you change "16" to "15" the code will crash
        - Broadcasting
            - df["a"] ** 2 # this squares every row in the column labeled a
        - With a data series (specific to one feature/column)
            - Example 1
                - def my_sq(x)
                - return x ** 2
                - df["a"].apply(my_sq)
            - Example 2
                - def my_sq(x, e)
                - return x ** e
                - df["a"].apply(my_sq, e)
        - With a data frame (applies a function to each entire column of a dataframe)
            - Example 1 # the apply statement here will print all data out in each column of the dataframe
                - def print_me(x):
                - return x

        - df.apply(print_me)
    - Example 2 # Get the mean of every column
        - def avg_apply(col):
        - return np.mean(col)
        - df.apply(avg_apply)
    - Example 3 # Get the mean of every column
        - def avg_apply(col):
        - x = col[0]
        - y = col[1]
        - z = col[2]
        - return (x+y+z)/3
        - df.apply(avg_apply) # or we could do df.apply(avg_apply, axis="columns") to get the average of each row
    - Example 4 # A more realistic example
        - @np.vectorize # Important step. We have to vectorize our functions so it can pass rows one at a time. We write this at the top right before the function.
        - def avg_2_mod(x, y):
        - if (x==20):
        - return np.NaN
        - else:
        - return (x + y) / 2
        - avg_2_mod(df["a"], df["b"]) #if we use number we have to do avg_2_mod(df["a"].values, df["b"].values)
    - Example 5
        - def extract_population(rate):
        - population = rate.split("/")[1]
        - return population

          # if we want to return population as an integer we can just return it as int(population)
        - assert extract_population("123/456") == "456"

          # quick way to check to make sure the function won't fail # random keywords: unit testing
        - tbl3["population"] = tbl3["rate"].apply(extract_population)

          # to apply the function to the column and create a new column called "population"
        -
        -
-
- Splitting into train and test set
    - IMPORTANT: Split into X_train, X_test, y_train, y_test like this (MAKE sure X and y are data frames! Not np.ndarray's! It keeps X_train, etc. as data frames so you can still run X_train.head() after

splitting)

- X = df_model # entire data frame
- y = df_model.tip_percent # tip_percent (dependent / target var)
-
-
- from sklearn.model_selection import train_test_split
- X_train, X_test, y_train, y_test = train_test_split(
- X, y, test_size=0.2, random_state=seed)

- Example 2
  -
  - # Getting the number of rows and columns
  - r, c = <myDataFrameName1>.shape
  -
  - # Creating an array which has indexes of columns
  - i_cols = []
  -
  - for i in range(0, c-1):
  - i_cols.append(i)
  -
  - # Y is the target column, X has the rest
  - X = <myDataFrameName1>[:, 0:(c-1)]
  - y = <myDataFrameName1>[:, (c-1)]
  -
  - del <myDataFrameName1>
  -
  - # Validation chunk size
  - val_size = 0.1
  -
  - # Using a common seed in all experiments so that same chunk is used for validation
  - seed = 0
  -
  - # Splitting the data
  - from sklearn.model_selection import train_test_split
  - X_train, X_test, y_train, y_test = train_test_split(
  - X, y, test_size=val_size, random_state=seed)
  -
  - del X
  - del y
  -
  - # All features
  - X_all = []
  -
  - # List of combinations
  - comb = []
  -

        - # Dictionary to store the Mean Absolute Error for all algorithms
        - mae = []
        -
        - #Scoring parameter
        - from sklearn.metrics import mean_absolute_error
        -
        - #Add this version of X to the list
        - n = “All”
        -
        - X_all.append([n, i_cols])
        -
        -
        - print(X_all) # A list of all the columns along with dummy vars
        -
        -
        -
        -
        -
        -
        -
        -
        -
        -
        -
        -
        -
        -
        -
        -
        -

- Grabbing specific feature values
    - df["Trip_distance"][df["Trip_distance"]<50]
- Duplicating time column, then striping hour field of it, and putting the hour value into a new column
    - df["pickup"] = df["lpep_pickup_datetime"].apply(lambda x: datetime.strptime(x, "%Y-%m-%d %H:%M:%S"))
    - df["pickup_hour"] = df["pickup"].apply(lambda x: x.hour)
- Creating a new feature from an existing date time column (datetime64 type):
    - X_train["pickup_day"] = X_train["pickup_datetime"].dt.day
- This puts, the average trip distance of every pickup hour into a new column
    - df["avg_trip_dist_per_pickup_hour"] = df[["Trip_distance","pickup_hour"]].groupby("pickup_hour").mean()
- List Unique Values In A pandas Column
    - https://chrisalbon.com/python/data_wrangling/

pandas_list_unique_values_in_column/

- select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix
  - https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/
- Groups two features, and provides the average of a third feature, and stores it into a dataframe
  - summary_wdays_avg_duration = pd.DataFrame(df.groupby(["day_of_week","pickup_hour"])["Trip_distance"].mean()) (line 1)
  - summary_wdays_avg_duration (line 2)
  - summary_wdays_avg_duration.reset_index(inplace = True) (resets the index for the data frame / makes one?)
  - summary_wdays_avg_duration["unit"]=1 (makes a new column with all one's)
- Drop row or drop column
  - train_X = X_train.drop(["Tip_amount", "tip_percent", "log_tip_percent"], axis=1) # how to drop multiple columns
  - https://chrisalbon.com/python/data_wrangling/pandas_dropping_column_and_rows/
  - df = df[df["Tip_amount"] > 0] # removes all instances in dataframe where Tip_amount <= 0
  - Calculations with two columns
    - df["tip_percent"] = df["Tip_amount"]/df["Total_amount"] # calculate tip percentage
    - df["tip_percent"] = df["tip_percent"].apply(lambda x: x * 100) # multiply by 100 to get the %
  - Python Pandas : Drop columns in DataFrame by label Names or by Index Positions
    - https://thispointer.com/python-pandas-drop-columns-in-dataframe-by-label-names-or-by-index-positions/
  - Drop Column
    - Example 1
      - <myDataFrameName1>.drop('<myFeatureName1>', axis=1, inplace=True)
  - Drop Index
    - See "Drop Column"
- Good To Know's When Plotting
  - Always pay attention to what your objects are returning while looking at the documentation for matplotlib, seaborn, etc.
- Preparing data to plot with matplotlib, etc (note all features that go into the plot arguments have the size of "(some#,)")
  - grouped_df = pd.DataFrame(df.groupby(["pickup_hour", "Airport"])["tip_percent"].aggregate(np.mean).reset_index())
- Quick plotting with Pandas
  - Example 1
    - <myDataFrameName1>.<myFeatureName1_categorical>.pl

ot(kind="bar")
    - Example 2
        -
            <myDataFrameName1>.<myFeatureName1_continuous>.pl
            ot(kind="hist")
- Heatmaps
    - https://docs.scipy.org/doc/numpy/reference/generated/numpy.zeros_like.html
    - https://docs.scipy.org/doc/numpy/reference/generated/numpy.triu_indices.html#numpy.triu_indices
    - Example 1 (shows one heat map)
        - sns.set(style="white")
        - # Generate a large random dataset
        - df_model_copy = <myDataFrameName1>.copy() # our dataframe
        -
        - # Compute the correlation matrix
        - corr = df_model_copy.corr() # corr calculation
        -
        - # Generate a mask for the upper triangle. Note, we only want to show the relevant part
        - # of the heat map
        - mask = np.zeros_like(corr, dtype=np.bool)
        - mask[np.triu_indices_from(mask)] = True
        -
        - # Generate a custom diverging colormap
        - cmap = sns.diverging_palette(220, 10, as_cmap=True)
        -
        - # Draw the heatmap with the mask and correct aspect ratio
        - sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
        - square=True, linewidths=.5, cbar_kws={"shrink": .5})
        -
        - plt.show()
    - Example 2 (shows two different heat maps, one before data skewing, and one after)
        - # Make sure we use the subsample in our correlation
        - f, (ax1, ax2) = plt.subplots(2, 1, figsize=(24,20))
        -
        - # Entire DataFrame
        - corr = <myDataFrameName1>.corr()
        - sns.heatmap(corr, cmap="coolwarm_r", annot_kws={"size": 20}, ax=ax1)
        - ax1.set_title("Imbalanced Correlation Matrix \n (don"t use for reference)", fontsize=14)
        -
        -

        - sub_sample_corr = <myDataFrameName2_withSkewCorrection>.corr()
        - sns.heatmap(sub_sample_corr, cmap="coolwarm_r", annot_kws={"size":20}, ax=ax2)
        - ax2.set_title("SubSample Correlation Matrix \n (use for reference)", fontsize=14)
        - plt.show()
- Violin Plots
    - Example 1
        - # We will visualize all the continuous attributes using Violin Plot - a combination of box and density plots
        - # Creating a dataframe with only continuous features
        - data = <myDataFrameSetForContinuousVarOnly>.iloc[:, 116:] # Creating a dataframe with only continuous features
        -
        - cols=data.columns
        -
        - # Plot violin for all attributes in a 7x2 facetgrid
        - n_cols = 2
        - n_rows = 7
        -
        - for i in range(n_rows):
        - fg, ax = plt.subplots(nrows=1,ncols=n_cols,figsize=(12,8))
        - for j in range(n_cols):
        - sns.violinplot(y=cols[i*n_cols+j], data=dataset_train, ax=ax[j])
        -
    - Example 2
        -
        - # Single Violin Plot
        - sns.violinplot(data=<myDataFrameName1>, y="<myFeatureName1_continuous>")
- Subplotting (subplots)

- Pair Plots
    - Example 1 (sns.pairplot)
        -
        - data = <myDataFrameSetForContinuousVarOnly>.iloc[:,

116:] # Creating a dataframe with only continuous features
- cols=data.columns # Getting the names of all the columns
- data_corr = data.corr() # Calculating Pearson coefficient for all combinations
- threshold = <myThresholdValue> # Setting the threshold to select only highly correlated attributes. Eg, 0.5.
- corr_list = [] # List of pairs along with correlation above threshold
- size = <myIntegerOfFeaturesToBeConsidered> # Eg, 15.
-
- # Searching for the highly correlated pairs
- for i in range(0, size): #for continuous features
- for j in range(i+1, size):
- if (data_corr.iloc[i,j] >= threshold and data_corr.iloc[i,j] < 1) or (data_corr.iloc[i,j] < 0 and data_corr.iloc[i,j] <= -threshold):
- corr_list.append([data_corr.iloc[i,j],i,j]) # stores coefficient and appropriate column indexes
-
- # Sorting to show higher ones first
- s_corr_list = sorted(corr_list,key=lambda x: -abs(x[0])). # See key function, https://docs.python.org/3/howto/sorting.html
-
- for v, i, j in s_corr_list:
- print("%s and %s = %.2f" % (cols[i],cols[j],v))
-
- # Scatter plot of all the highly correlated pairs
- for v, i, j in s_corr_list:
- sns.pairplot(dataset_train, size=6, x_vars=cols[i], y_vars=cols[j])
- plt.show

- Boxplots
  - Example 1 (box plots with hours)
    - f, axes = plt.subplots(ncols=4, figsize=(20,4))
    - # Negative Correlations with our <myFeatureName1_categorical> (The lower our feature value the more likely it will be a fraud transaction)
    - sns.boxplot(x="<myFeatureName1_categorical>", y="<myFeatureName2_continuous>", data=<myDataFrameName1>, palette=colors, ax=axes[0])
    - axes[0].set_title("<myFeatureName2_continuous> vs <myFeatureName1_categorical> Negative Correlation")

        -
        - sns.boxplot(x="<myFeatureName1_categorical>", y="<myFeatureName3_continuous>", data=<myDataFrameName1>, palette=colors, ax=axes[1])
        - axes[1].set_title("<myFeatureName3_continuous> vs <myFeatureName1_categorical> Negative Correlation")
        -
        - sns.boxplot(x="<myFeatureName1_categorical>", y="<myFeatureName4_continuous>", data=<myDataFrameName1>, palette=colors, ax=axes[2])
        - axes[2].set_title("<myFeatureName4_continuous> vs <myFeatureName1_categorical> Negative Correlation")
        -
        - sns.boxplot(x="<myFeatureName1_categorical>", y="<myFeatureName5_continuous>", data=<myDataFrameName1>, palette=colors, ax=axes[3])
        - axes[3].set_title("<myFeatureName5_continuous> vs <myFeatureName1_categorical> Negative Correlation")
        -
        - plt.show()
- Histograms
    - https://towardsdatascience.com/histograms-and-density-plots-in-python-f6bda88f5ac0
    - Example 1 # a quick way to generate a histogram within Pandas (plot)
        - <myDataFrameName1>.<myFeatureName1_continuous>.plot(kind="hist")
    - Example 2 # with seaborn, good for continuous variables (distplot)
        - sns.distplot(<myDataFrameName1>.<myFeatureName1_continuous>) # this plots a histogram plot along with the kernel density shape
        - sns.distplot(<myDataFrameName1>.<myFeatureName1_continuous>, kde=False) # plots only a histogram
- Scatter Plot (with Linear fits)
    - Example 1 # using seaborn, this shows a scatter plot and applies a linear fit
        - sns.lmplot(x="<myFeatureName1_continuous>", y="<myFeatureName2_continuous>", data=<myDataFrameName1>) # note, sns.lmplot returns back an entire figure whereas sns.regplot can be used for subplotting/in axes form (still a little confused, but overall I think sns.regplot is typically used when subplotting)
    - Example 2 # using seaborn, this shows the data as a scatter plot and colors the scattered data based on categories in

myFeatureName3 (note, there are two linear fits applied to this plot if <myFeatureName3> is binary).
    - sns.lmplot(x="<myFeatureName1_continuous>", y="<myFeatureName2_continuous>", data=<myDataFrameName1>, hue="<myFeatureName3_categorical>")
- Example 3 # using seaborn, this shows the data as a scatter plot and colors the scattered data based on categories in myFeatureName3 (note, there are two linear fits applied to this plot if <myFeatureName3> is binary). The addition of the col argument <myFeatureName4> separates the data into two plots. This categorical var must match the categorical var in <myFeatureName3>
    - sns.lmplot(x="<myFeatureName1_continuous>", y="<myFeatureName2_continuous>", data=<myDataFrameName1>, hue="<myFeatureName3_categorical>", col="<myFeatureName4_categorical>")
- Example 4 # similar to above, but including the row argument turns this into a FacetGrid type plot (think of an 8x2 grid, etc). This is a great way to plot using multiple vars. Note <myFeatureName3>, <myFeatureName4>, <myFeatureName5> can be binary, binary, multi categorical, respectively.
    - sns.lmplot(x="<myFeatureName1_continuous>", y="<myFeatureName2_continuous>", data=<myDataFrameName1>, hue="<myFeatureName3_categorical>", col="<myFeatureName4_categorical>", row="<myFeatureName5_categorical>")
- Exampe 5
    -

- FacetGrid / Subplotting
    - Scatter Plot
        - Example 1
            -
            - <someVar1> = sns.FacetGrid(<myDataFrameName1>, col="<myFeatureName1_categorical>", row="<myFeatureName2_categorical>", hue="<myFeatureName3_categorical>")
            - <someVar1>.map(plt.scatter, "<myFeatureName4_continuous>", "<myFeatureName5_continuous>") #think of "map()" similar to how "apply()" works. Here, map() provides/applies the data to each subplot in the facet grid
    - Histogram and Scatter Plot (with linear fit)

- Example 1
  - fig, (ax1, ax2) = plt.subplots(1,2)
  - sns.distplot(<myDataFrameName1>.<myFeatureName1_continuous>, ax=ax1)
  - sns.regplot(x="<myFeatureName2_continuous>",y="<myFeatureName1_continuous>", data=<myDataFrameName1>, ax=ax2) # note, this seems very similar to sns.lmplot
  -
- Bar Graph / Count Plot
  - Example 1 # a quick way to generate a bar plot within Pandas
    - cts = <myDataFrameName1>.<myFeatureName1_categorical>.value_counts
    - cts.plot(kind="bar")
  - Example 2 # with seaborn
    - sns.countplot(x="<myFeatureName1_categorical>", data = <myDataFrameName1>)
  - Example 3 # FacetGrid type with seaborn
    -
    -
    -
    - n_rows = <myIntOfRows>
    - n_cols = <myIntOfColumns>
    -
    - for i in range(n_rows):
    - fg, ax = plt.subplots(nrows=1,ncols=n_cols,figsize=(16,8))
    - for j in range(n_cols):
    - sns.countplot(x=cols[i*n_cols+j], data=<myDataFrameName1_categorical>, ax=ax[j])
    -
    -
- Dendogram
  - Example 1
  -
  -
- Modeling
  - Regression
    - Linear
      - Linear Regression
        - Example 1
          -
          - from sklearn.linear_model import LinearRegression
          - <myDataFrameName1> =

sns.load_dataset("tips")
    - <myDataFrameName1>.head()
    - <myModel1> = LinearRegression()
    -
      <myModel1>.fit(X=<myDataFrameNam
      e1>[["<myFeatureName1>","<myFeatur
      eName2>"]],
      y=<myDataFrameName1>["<myFeature
      Name3>"])
    - <myModel1>.coef_
    - <myModel1>.intercept_
    -
    - # LEARNING:
    - # For every one dollar increase in
      total_bill, given that
    - # the size remains the same, the tip
      increases by 9 cents.
    -
- Example 2
    -
    -
    - # Getting the number of rows and
      columns
    - r, c = <myDataFrameName1>.shape
    -
    - # Creating an array which has indexes
      of columns
    - i_cols = []
    -
    - for i in range(0, c-1):
    - i_cols.append(i)
    -
    - # Y is the target column, X has the rest
    - X = <myDataFrameName1>[:, 0:(c-1)]
    - y = <myDataFrameName1>[:, (c-1)]
    -
    - # Validation chunk size
    - val_size = 0.1
    -
    - # Using a common seed in all
      experiments so that same chunk is used
      for validation
    - seed = 0
    -
    - from sklearn.model_selection import
      train_test_split
    - X_train, X_test, y_train, y_test =

```
train_test_split(
-       X, y, test_size=val_size,
random_state=seed)
-
-   del X
-   del y
-
-   # All features
-   X_all = []
-
-   # List of combinations
-   comb = []
-
-   # Dictionary to store the Mean Absolute
Error for all algorithms
-   mae = []
-
-   #Scoring parameter
-   from sklearn.metrics import
mean_absolute_error
-
-   #Add this version of X to the list
-   n = “All”
-
-   X_all.append([n, i_cols])
-
-
-   print(X_all)     # A list of all the columns
along with dummy vars
-
-
-
-   # LINEAR REGRESSION (Linear Algo)
-   from sklearn.linear_model import
LinearRegression
-   lin_reg = LinearRegression(n_jobs=-1) #
using all processors
-   algo = "LR"
-
-
-
-
-
-
-   # Accuracy of the model using all
features
-   for name, i_cols_list in X_all:
-       print(name)
```

- lin_reg.fit(X_train[:, i_cols_list], y_train) #fitting all features to the target column
- result = mean_absolute_error(np.expm1(y_test), np.expm1(lin_reg.predict(X_test[:,i_cols_list])))
- mae.append(result)
- print(name + " %s" % result)
-
- comb.append(algo)
- print(comb)
-
- # #Plot the MAE of all combinations
- # fig, ax = plt.subplots()
- # plt.plot(mae)
- # #Set the tick names to names of combinations
- # ax.set_xticks(range(len(comb)))
- # ax.set_xticklabels(comb,rotation='vertical')
- # #Plot the accuracy for all combinations
- # plt.show()
-
-
-
-
-
-
-

- Logistic Regression
  - Example 1
    -
    - from sklearn.linear_model import LogisticRegression
    - <myDataFrameName1> = sns.load_dataset("titanic")
    - <myDataFrameName1>.head()
    - <myDataFrameName1>.info()
    - <myDataFrameName1>.drop([“<myFeatureName1>”], axis=1).head()
    - <myDataFrameName2> = <myDataFrameName1>[["<myFeatureName2>", "<myFeatureName3>",

"<myFeatureName4>"]]
- <myDataFrameName2>.head()
- <myDataFrameName3> = pd.get_dummies(<myDataFrameName2>, drop_first=True)
- <myDataFrameName3>.head()
- <myXTrain1> = <myDataFrameName3>.iloc[:,1:]
- <myXTrain1>.head()
- <myYTrain1> = <myDataFrameName3>.iloc[:,0]
- <myYTrain1>.head()
- <myModel1> = LogisticRegression(random_state=0, solver="lbfgs", multi_class="multinomial")
- <myModel1>.fit(<myXTrain1>, <myYTrain1>)
- <myModel1>.coef_
-

- Non-Linear
  - KNN
    - Example 1
      -
      - # Getting the number of rows and columns
      - r, c = <myDataFrameName1>.shape
      -
      - # Creating an array which has indexes of columns
      - i_cols = []
      -
      - for i in range(0, c-1):
      - i_cols.append(i)
      -
      - # Y is the target column, X has the rest
      - X = <myDataFrameName1>[:, 0:(c-1)]
      - y = <myDataFrameName1>[:, (c-1)]
      -
      - # Validation chunk size
      - val_size = 0.1
      -
      - # Using a common seed in all experiments so that same chunk is used for validation
      - seed = 0
      -

```
- from sklearn.model_selection import
  train_test_split
- X_train, X_test, y_train, y_test =
  train_test_split(
-     X, y, test_size=val_size,
  random_state=seed)
-
- del X
- del y
-
- # All features
- X_all = []
-
- # List of combinations
- comb = []
-
- # Dictionary to store the Mean Absolute
  Error for all algorithms
- mae = []
-
- #Scoring parameter
- from sklearn.metrics import
  mean_absolute_error
-
- #Add this version of X to the list
- n = “All”
-
- X_all.append([n, i_cols])
-
-
- print(X_all)     # A list of all the columns
  along with dummy vars
-
-
- # KNN (Non-linear Algo)
-
-
- # Evaluation of various combinations of
  KNN
-
- # Fitting Classifier to the Training set
- from sklearn.neighbors import
  KNeighborsRegressor
- # https://scikit-learn.org/stable/modules/
  generated/
  sklearn.neighbors.KNeighborsRegressor
  .html
```

```
-
-
-    # Add the N value to the below list if you
     want to run the algo
-
-    n_list = np.array([]) #note, when the list
     is empty, the algo doesnt run
-    '''
-    With n_list = np.array([5])
-
-    All 1434.788795356784
-    ['LR', 'KNN 5']
-    '''
-
-    '''
-    With n_list = np.array([2])
-
-    All 1526.7442802258656
-    ['LR', 'KNN 2']
-    '''
-
-    # we can use multiple values into n_list
     if we want to search for the optimal
     n_neighbors. However, xgboost is usally
     the best for parameter tuning.
-    for n_neighbors in n_list:
-        # Setting the base model
-        regressor =
     KNeighborsRegressor(n_neighbors=n_n
     eighbors,n_jobs=-1)
-
-        algo = "KNN"
-
-        #Accuracy of the model using all
     features
-        for name, i_cols_list in X_all:
-            regressor.fit(X_train[:, i_cols_list],
     y_train) #fitting all features to the target
     column
-            result =
     mean_absolute_error(np.expm1(y_test),
     np.expm1(regressor.predict(X_test[:,i_c
     ols_list])))
-            mae.append(result)
-            print(name + " %s" % result)
-        comb.append(algo + " %s" %
     n_neighbors)
```

-
-
print(comb)
-
-
-
since we know the outcome, we can skip the algorithm and append the result
-
if (len(n_list)==0):
-
mae.append(1527)
-
comb.append("KNN" + " %s" % 2)
-
-
-
-
##Set figure size, this figure compares mae for all of the algorithms ran
-
-
#plt.rc("figure", figsize=(25, 10))
-
-
##Plot the MAE of all combinations
-
#fig, ax = plt.subplots()
-
#plt.plot(mae)
-
##Set the tick names to names of combinations
-
#ax.set_xticks(range(len(comb)))
-
#ax.set_xticklabels(comb,rotation='vertical')
-
##Plot the accuracy for all combinations
-
#plt.show()
-
-
#Very high computation time
-
#Best estimated performance is 1745 for n=1
-
-
LEARNING:
-
KNN 5 performed the best. Lowest MAE.

- Decision Tree (CART)
  - Example 1
    -
    -
    -
    -
    # Getting the number of rows and columns
    -
    r, c = <myDataFrameName1>.shape
    -
    -
    # Creating an array which has indexes of columns

```
-    i_cols = []
-
-    for i in range(0, c-1):
-        i_cols.append(i)
-
-    # Y is the target column, X has the rest
-    X = <myDataFrameName1>[:, 0:(c-1)]
-    y = <myDataFrameName1>[:, (c-1)]
-
-    # Validation chunk size
-    val_size = 0.1
-
-    # Using a common seed in all
     experiments so that same chunk is used
     for validation
-    seed = 0
-
-    from sklearn.model_selection import
     train_test_split
-    X_train, X_test, y_train, y_test =
     train_test_split(
-        X, y, test_size=val_size,
     random_state=seed)
-
-    del X
-    del y
-
-    # All features
-    X_all = []
-
-    # List of combinations
-    comb = []
-
-    # Dictionary to store the Mean Absolute
     Error for all algorithms
-    mae = []
-
-    #Scoring parameter
-    from sklearn.metrics import
     mean_absolute_error
-
-    #Add this version of X to the list
-    n = “All”
-
-    X_all.append([n, i_cols])
-
-
```

```
-   print(X_all)     # A list of all the columns
    along with dummy vars
-
-
-   # CART (Non-linear Algo)
-
-
-
-    #Evaluation of various combinations of
    CART
-
-   #Import the library
-   from sklearn.tree import
    DecisionTreeRegressor
-
-   #Add the max_depth value to the below
    list if you want to run the algo
-   d_list = np.array([])
-
-   for max_depth in d_list:
-      #Set the base model
-      model =
    DecisionTreeRegressor(max_depth=ma
    x_depth,random_state=seed)
-
-      algo = "CART"
-
-      #Accuracy of the model using all
    features
-      for name,i_cols_list in X_all:
-
    model.fit(X_train[:,i_cols_list],Y_train)
-          result =
    mean_absolute_error(np.expm1(y_test),
    np.expm1(model.predict(X_test[:,i_cols_
    list])))
-          mae.append(result)
-          print(name + " %s" % result)
-
-      comb.append(algo + " %s" %
    max_depth )
-
-
-   # since we know the outcome, we can
    skip the algorithm and append the result
-   if (len(d_list)==0):
-      mae.append(1741)
```

- comb.append("CART" + " %s" % 5)
-
-
-
- ##Set figure size
- #plt.rc("figure", figsize=(25, 10))
-
- ##Plot the MAE of all combinations
- #fig, ax = plt.subplots()
- #plt.plot(mae)
- ##Set the tick names to names of combinations
- #ax.set_xticks(range(len(comb)))
- #ax.set_xticklabels(comb,rotation='vertical')
- ##Plot the accuracy for all combinations
- #plt.show()
-
- #High computation time
- #Best estimated performance is 1741 for depth=5
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
- Support Vector Machine (SVM, SVR)
    - Example 1
        -
        -

-
-
-
- # Getting the number of rows and columns
- r, c = <myDataFrameName1>.shape
-
- # Creating an array which has indexes of columns
- i_cols = []
-
- for i in range(0, c-1):
- i_cols.append(i)
-
- # Y is the target column, X has the rest
- X = <myDataFrameName1>[:, 0:(c-1)]
- y = <myDataFrameName1>[:, (c-1)]
-
- # Validation chunk size
- val_size = 0.1
-
- # Using a common seed in all experiments so that same chunk is used for validation
- seed = 0
-
- from sklearn.model_selection import train_test_split
- X_train, X_test, y_train, y_test = train_test_split(
- X, y, test_size=val_size, random_state=seed)
-
- del X
- del y
-
- # All features
- X_all = []
-
- # List of combinations
- comb = []
-
- # Dictionary to store the Mean Absolute Error for all algorithms
- mae = []
-
- #Scoring parameter

```
-   from sklearn.metrics import
    mean_absolute_error
-
-   #Add this version of X to the list
-   n = “All”
-
-   X_all.append([n, i_cols])
-
-
-   print(X_all)     # A list of all the columns
    along with dummy vars
-
-
-
-
-    # SVM (Non-linear Algo)
-
-
-   #Import the library
-   from sklearn.svm import SVR
-
-   #Add the C value to the below list if you
    want to run the algo
-   c_list = np.array([])
-
-   for C in c_list:
-      #Set the base model
-      model = SVR(C=C)
-
-      algo = "SVM"
-
-      #Accuracy of the model using all
    features
-      for name,i_cols_list in X_all:
-
    model.fit(X_train[:,i_cols_list],Y_train)
-         result =
    mean_absolute_error(np.expm1(y_test),
    np.expm1(model.predict(X_test[:,i_cols_
    list])))
-         mae.append(result)
-         print(name + " %s" % result)
-
-      comb.append(algo + " %s" % C )
-
-
-
```

-
- ##Set figure size
- #plt.rc("figure", figsize=(25, 10))
-
- ##Plot the MAE of all combinations
- #fig, ax = plt.subplots()
- #plt.plot(mae)
- ##Set the tick names to names of combinations
- #ax.set_xticks(range(len(comb)))
-
  #ax.set_xticklabels(comb,rotation='vertical')
- ##Plot the accuracy for all combinations
- #plt.show()
-
- #very very high computation time, not running
-

- Bagged Decision Trees (Bagging)
  - Example 1
    -
    -
    -
    -
    -
    - # Getting the number of rows and columns
    - r, c = <myDataFrameName1>.shape
    -
    - # Creating an array which has indexes of columns
    - i_cols = []
    -
    - for i in range(0, c-1):
    - i_cols.append(i)
    -
    - # Y is the target column, X has the rest
    - X = <myDataFrameName1>[:, 0:(c-1)]
    - y = <myDataFrameName1>[:, (c-1)]
    -
    - # Validation chunk size
    - val_size = 0.1
    -
    - # Using a common seed in all experiments so that same chunk is used for validation

```
-   seed = 0
-
-   from sklearn.model_selection import
    train_test_split
-   X_train, X_test, y_train, y_test =
    train_test_split(
-       X, y, test_size=val_size,
    random_state=seed)
-
-   del X
-   del y
-
-   # All features
-   X_all = []
-
-   # List of combinations
-   comb = []
-
-   # Dictionary to store the Mean Absolute
    Error for all algorithms
-   mae = []
-
-   #Scoring parameter
-   from sklearn.metrics import
    mean_absolute_error
-
-   #Add this version of X to the list
-   n = “All”
-
-   X_all.append([n, i_cols])
-
-
-   print(X_all)     # A list of all the columns
    along with dummy vars
-
-
-
-
-    # Bagged Decision Trees (Bagging)
-
-
-
-   #Evaluation of various combinations of
    Bagged Decision Trees
-
-
-
```

```
-    #Import the library
-    from sklearn.ensemble import
     BaggingRegressor
-    #from sklearn.tree import
     DecisionTreeRegressor
-
-    #Add the n_estimators value to the
     below list if you want to run the algo
-    n_list = np.array([])
-
-    for n_estimators in n_list:
-       #Setting the base model
-       model =
     BaggingRegressor(n_jobs=-1,n_estimat
     ors=n_estimators)
-
-       algo = "Bag"
-
-       #Accuracy of the model using all
     features
-       for name,i_cols_list in X_all:
-
     model.fit(X_train[:,i_cols_list],Y_train)
-           result =
     mean_absolute_error(np.expm1(y_test),
     np.expm1(model.predict(X_test[:,i_cols_
     list])))
-           mae.append(result)
-           print(name + " %s" % result)
-
-       comb.append(algo + " %s" %
     n_estimators )
-
-
-
-    ##Set figure size
-    #plt.rc("figure", figsize=(25, 10))
-
-    ##Plot the MAE of all combinations
-    #fig, ax = plt.subplots()
-    #plt.plot(mae)
-    ##Set the tick names to names of
     combinations
-    #ax.set_xticks(range(len(comb)))
-
     #ax.set_xticklabels(comb,rotation='vertic
     al')
```

- ##Plot the accuracy for all combinations
- #plt.show()
-
- #very high computation time, not running
-

- Random Forest (Bagging) # note, we use bagging to find the sweet spot between a simple and complex model (see bias vs variance tradeoff)
  - Example 1
    -
    -
    -
    -
    - # Getting the number of rows and columns
    - r, c = <myDataFrameName1>.shape
    -
    - # Creating an array which has indexes of columns
    - i_cols = []
    -
    - for i in range(0, c-1):
    - i_cols.append(i)
    -
    - # Y is the target column, X has the rest
    - X = <myDataFrameName1>[:, 0:(c-1)]
    - y = <myDataFrameName1>[:, (c-1)]
    -
    - # Validation chunk size
    - val_size = 0.1
    -
    - # Using a common seed in all experiments so that same chunk is used for validation
    - seed = 0
    -
    - from sklearn.model_selection import train_test_split
    - X_train, X_test, y_train, y_test = train_test_split(
    - X, y, test_size=val_size, random_state=seed)
    -
    - del X
    - del y
    -

```
-   # All features
-   X_all = []
-
-   # List of combinations
-   comb = []
-
-   # Dictionary to store the Mean Absolute
    Error for all algorithms
-   mae = []
-
-   #Scoring parameter
-   from sklearn.metrics import
    mean_absolute_error
-
-   #Add this version of X to the list
-   n = “All”
-
-   X_all.append([n, i_cols])
-
-
-   print(X_all)     # A list of all the columns
    along with dummy vars
-
-
-
-   # Random Forest (Bagging)
-
-
-
-   # Evaluation of various combinations of
    RandomForest
-
-   #Import the library
-   from sklearn.ensemble import
    RandomForestRegressor
-
-   #Add the n_estimators value to the
    below list if you want to run the algo
-   n_list = np.array([])
-
-   for n_estimators in n_list:
-      #Set the base model
-      model =
    RandomForestRegressor(n_jobs=-1,n_e
    stimators=n_estimators,random_state=s
    eed)
-
```

- algo = "RF"
-
- #Accuracy of the model using all features
- for name,i_cols_list in X_all:
- model.fit(X_train[:,i_cols_list],Y_train)
- result = mean_absolute_error(np.expm1(y_test), np.expm1(model.predict(X_testl[:,i_cols_list])))
- mae.append(result)
- print(name + " %s" % result)
-
- comb.append(algo + " %s" % n_estimators)
-
-
- # since we know the outcome, we can skip the algorithm and append the result
- if (len(n_list)==0):
- mae.append(1213)
- comb.append("RF" + " %s" % 50)
-
- ##Set figure size
- #plt.rc("figure", figsize=(25, 10))
-
- ##Plot the MAE of all combinations
- #fig, ax = plt.subplots()
- #plt.plot(mae)
- ##Set the tick names to names of combinations
- #ax.set_xticks(range(len(comb)))
-
#ax.set_xticklabels(comb,rotation='vertical')
- ##Plot the accuracy for all combinations
- #plt.show()
-
- #Best estimated performance is 1213 when the number of estimators is 50
-
-

- Extra Trees (Bagging) # note, we use bagging to find the sweet spot between a simple and complex model (see bias vs variance tradeoff)
  - Example 1

```
-
-
-   # Getting the number of rows and
    columns
-   r, c = <myDataFrameName1>.shape
-
-   # Creating an array which has indexes
    of columns
-   i_cols = []
-
-   for i in range(0, c-1):
-       i_cols.append(i)
-
-   # Y is the target column, X has the rest
-   X = <myDataFrameName1>[:, 0:(c-1)]
-   y = <myDataFrameName1>[:, (c-1)]
-
-   # Validation chunk size
-   val_size = 0.1
-
-   # Using a common seed in all
    experiments so that same chunk is used
    for validation
-   seed = 0
-
-   from sklearn.model_selection import
    train_test_split
-   X_train, X_test, y_train, y_test =
    train_test_split(
-       X, y, test_size=val_size,
    random_state=seed)
-
-   del X
-   del y
-
-   # All features
-   X_all = []
-
-   # List of combinations
-   comb = []
-
-   # Dictionary to store the Mean Absolute
    Error for all algorithms
-   mae = []
-
-   #Scoring parameter
-   from sklearn.metrics import
```

```
        mean_absolute_error
-
-       #Add this version of X to the list
-       n = “All”
-
-       X_all.append([n, i_cols])
-
-
-       print(X_all)     # A list of all the columns
        along with dummy vars
-
-
-
-       # Extra Trees (Bagging)
-
-
-
-       #Evaluation of various combinations of
        ExtraTrees
-
-       #Import the library
-       from sklearn.ensemble import
        ExtraTreesRegressor
-
-
-       #Add the n_estimators value to the
        below list if you want to run the algo
-       n_list = np.array([])
-
-       for n_estimators in n_list:
-          #Set the base model
-          model =
        ExtraTreesRegressor(n_jobs=-1,n_estim
        ators=n_estimators,random_state=seed
        )
-
-          algo = "ET"
-
-          #Accuracy of the model using all
        features
-          for name,i_cols_list in X_all:
-
        model.fit(X_train[:,i_cols_list],Y_train)
-             result =
        mean_absolute_error(np.expm1(y_test),
        np.expm1(model.predict(X_test[:,i_cols_
        list])))
```

- mae.append(result)
- print(name + " %s" % result)
-
- comb.append(algo + " %s" % n_estimators)
-
-
-
- # since we know the outcome, we can skip the algorithm and append the result
- if (len(n_list)==0):
- mae.append(1254)
- comb.append("ET" + " %s" % 100)
-
-
-
- ##Set figure size
- #plt.rc("figure", figsize=(25, 10))
-
- ##Plot the MAE of all combinations
- #fig, ax = plt.subplots()
- #plt.plot(mae)
- ##Set the tick names to names of combinations
- #ax.set_xticks(range(len(comb)))
-
#ax.set_xticklabels(comb,rotation='vertical')
- ##Plot the accuracy for all combinations
- #plt.show()
-
- #Best estimated performance is 1254 for 100 estimators
-

- Adaboost # note, we use bagging to find the sweet spot between a simple and complex model (see bias vs variance tradeoff)
  - Example 1
    -
    -
    - # Getting the number of rows and columns
    - r, c = <myDataFrameName1>.shape
    -
    - # Creating an array which has indexes of columns
    - i_cols = []

```
-
-       for i in range(0, c-1):
-           i_cols.append(i)
-
-       # Y is the target column, X has the rest
-       X = <myDataFrameName1>[:, 0:(c-1)]
-       y = <myDataFrameName1>[:, (c-1)]
-
-       # Validation chunk size
-       val_size = 0.1
-
-       # Using a common seed in all
        experiments so that same chunk is used
        for validation
-       seed = 0
-
-       from sklearn.model_selection import
        train_test_split
-       X_train, X_test, y_train, y_test =
        train_test_split(
-           X, y, test_size=val_size,
        random_state=seed)
-
-       del X
-       del y
-
-       # All features
-       X_all = []
-
-       # List of combinations
-       comb = []
-
-       # Dictionary to store the Mean Absolute
        Error for all algorithms
-       mae = []
-
-       #Scoring parameter
-       from sklearn.metrics import
        mean_absolute_error
-
-       #Add this version of X to the list
-       n = “All”
-
-       X_all.append([n, i_cols])
-
-
-       print(X_all)     # A list of all the columns
```

```
        along with dummy vars
-
-
-       #Evaluation of various combinations of
        AdaBoost
-
-
-       #Import the library
-       from sklearn.ensemble import
        AdaBoostRegressor
-
-       #Add the n_estimators value to the
        below list if you want to run the algo
-       n_list = np.array([])
-
-       for n_estimators in n_list:
-           #Set the base model
-           model =
        AdaBoostRegressor(n_estimators=n_es
        timators,random_state=seed)
-
-           algo = "Ada"
-
-           #Accuracy of the model using all
        features
-           for name,i_cols_list in X_all:
-
        model.fit(X_train[:,i_cols_list],Y_train)
-               result =
        mean_absolute_error(np.expm1(y_test),
        np.expm1(model.predict(X_test[:,i_cols_
        list])))
-               mae.append(result)
-               print(name + " %s" % result)
-
-           comb.append(algo + " %s" %
        n_estimators )
-
-
-       # since we know the outcome, we can
        skip the algorithm and append the result
-       if (len(n_list)==0):
-           mae.append(1678)
-           comb.append("Ada" + " %s" % 100 )
-
-       ##Set figure size
-       #plt.rc("figure", figsize=(25, 10))
```

-
- ##Plot the MAE of all combinations
- #fig, ax = plt.subplots()
- #plt.plot(mae)
- ##Set the tick names to names of combinations
- #ax.set_xticks(range(len(comb)))
- #ax.set_xticklabels(comb,rotation='vertical')
- ##Plot the accuracy for all combinations
- #plt.show()
-
- #Best estimated performance is 1678 with n=100
-

- Stochastic Gradient Boosting (Boosting) # note, we use bagging to find the sweet spot between a simple and complex model (see bias vs variance tradeoff)
  - Example 1
    -
    -
    - # Getting the number of rows and columns
    - r, c = <myDataFrameName1>.shape
    -
    - # Creating an array which has indexes of columns
    - i_cols = []
    -
    - for i in range(0, c-1):
    - i_cols.append(i)
    -
    - # Y is the target column, X has the rest
    - X = <myDataFrameName1>[:, 0:(c-1)]
    - y = <myDataFrameName1>[:, (c-1)]
    -
    - # Validation chunk size
    - val_size = 0.1
    -
    - # Using a common seed in all experiments so that same chunk is used for validation
    - seed = 0
    -
    - from sklearn.model_selection import train_test_split

```
- X_train, X_test, y_train, y_test =
  train_test_split(
-     X, y, test_size=val_size,
  random_state=seed)
-
- del X
- del y
-
- # All features
- X_all = []
-
- # List of combinations
- comb = []
-
- # Dictionary to store the Mean Absolute
  Error for all algorithms
- mae = []
-
- #Scoring parameter
- from sklearn.metrics import
  mean_absolute_error
-
- #Add this version of X to the list
- n = “All”
-
- X_all.append([n, i_cols])
-
-
- print(X_all)     # A list of all the columns
  along with dummy vars
-
- #Evaluation of various combinations of
  SGB
-
-
- #Import the library
- from sklearn.ensemble import
  GradientBoostingRegressor
-
- #Add the n_estimators value to the
  below list if you want to run the algo
- n_list = np.array([])
-
- for n_estimators in n_list:
-    #Set the base model
-    model =
  GradientBoostingRegressor(n_estimator
```

s=n_estimators,random_state=seed)
-
- algo = "SGB"
-
- #Accuracy of the model using all features
- for name,i_cols_list in X_all:
-
model.fit(X_train[:,i_cols_list],Y_train)
- result = mean_absolute_error(np.expm1(y_test), np.expm1(model.predict(X_testl[:,i_cols_list])))
- mae.append(result)
- print(name + " %s" % result)
-
- comb.append(algo + " %s" % n_estimators)
-
- # since we know the outcome, we can skip the algorithm and append the result
- if (len(n_list)==0):
- mae.append(1278)
- comb.append("SGB" + " %s" % 50)
-
- ##Set figure size
- #plt.rc("figure", figsize=(25, 10))
-
- ##Plot the MAE of all combinations
- #fig, ax = plt.subplots()
- #plt.plot(mae)
- ##Set the tick names to names of combinations
- #ax.set_xticks(range(len(comb)))
-
#ax.set_xticklabels(comb,rotation='vertical')
- ##Plot the accuracy for all combinations
- #plt.show()
-
- #Best estimated performance is ?
-

- XGBoost # note, we use bagging to find the sweet spot between a simple and complex model (see bias vs variance tradeoff)
  - Example 1
    -

```
-
-    # Getting the number of rows and
     columns
-    r, c = <myDataFrameName1>.shape
-
-    # Creating an array which has indexes
     of columns
-    i_cols = []
-
-    for i in range(0, c-1):
-        i_cols.append(i)
-
-    # Y is the target column, X has the rest
-    X = <myDataFrameName1>[:, 0:(c-1)]
-    y = <myDataFrameName1>[:, (c-1)]
-
-    # Validation chunk size
-    val_size = 0.1
-
-    # Using a common seed in all
     experiments so that same chunk is used
     for validation
-    seed = 0
-
-    from sklearn.model_selection import
     train_test_split
-    X_train, X_test, y_train, y_test =
     train_test_split(
-        X, y, test_size=val_size,
     random_state=seed)
-
-    del X
-    del y
-
-    # All features
-    X_all = []
-
-    # List of combinations
-    comb = []
-
-    # Dictionary to store the Mean Absolute
     Error for all algorithms
-    mae = []
-
-    #Scoring parameter
-    from sklearn.metrics import
     mean_absolute_error
```

```
-
-       #Add this version of X to the list
-       n = “All”
-
-       X_all.append([n, i_cols])
-
-
-       print(X_all)     # A list of all the columns
        along with dummy vars
-
-
-
-        #XGBoost
-
-       #Evaluation of various combinations of
        XGB
-
-       #Import the library
-       from xgboost import XGBRegressor
-
-       #Add the n_estimators value to the
        below list if you want to run the algo
-       n_list = np.array([])
-
-       for n_estimators in n_list:
-          #Set the base model
-          model =
        XGBRegressor(n_estimators=n_estimat
        ors,seed=seed)
-
-          algo = "XGB"
-
-          #Accuracy of the model using all
        features
-          for name,i_cols_list in X_all:
-
        model.fit(X_train[:,i_cols_list],Y_train)
-             result =
        mean_absolute_error(np.expm1(y_test),
        np.expm1(model.predict(X_test[:,i_cols_
        list])))
-             mae.append(result)
-             print(name + " %s" % result)
-
-          comb.append(algo + " %s" %
        n_estimators )
-
```

- # since we know the outcome, we can skip the algorithm and append the result
- if (len(n_list)==0):
- mae.append(1169)
- comb.append("XGB" + " %s" % 1000)
-
- ##Set figure size
- #plt.rc("figure", figsize=(25, 10))
-
- ##Plot the MAE of all combinations
- #fig, ax = plt.subplots()
- #plt.plot(mae)
- ##Set the tick names to names of combinations
- #ax.set_xticks(range(len(comb)))
-
#ax.set_xticklabels(comb,rotation='vertical')
- ##Plot the accuracy for all combinations
- #plt.show()
-
- #Best estimated performance is 1169 with n=1000
-

- Multi-layer Perceptrons (Deep Learning)
  - Example 1
    -
    -
    - # Getting the number of rows and columns
    - r, c = <myDataFrameName1>.shape
    -
    - # Creating an array which has indexes of columns
    - i_cols = []
    -
    - for i in range(0, c-1):
    - i_cols.append(i)
    -
    - # Y is the target column, X has the rest
    - X = <myDataFrameName1>[:, 0:(c-1)]
    - y = <myDataFrameName1>[:, (c-1)]
    -
    - # Validation chunk size
    - val_size = 0.1
    -

```
# Using a common seed in all experiments so that same chunk is used for validation
seed = 0

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=val_size, random_state=seed)

del X
del y

# All features
X_all = []

# List of combinations
comb = []

# Dictionary to store the Mean Absolute Error for all algorithms
mae = []

#Scoring parameter
from sklearn.metrics import mean_absolute_error

#Add this version of X to the list
n = “All”

X_all.append([n, i_cols])


print(X_all)     # A list of all the columns along with dummy vars


#MLP (Deep Learning)



#Evaluation of various combinations of multi-layer perceptrons

#Import libraries for deep learning
```

```
- from keras.wrappers.scikit_learn import KerasRegressor
- from keras.models import Sequential
- from keras.layers import Dense
-
- # define baseline model
- def baseline(v):
-     # create model
-     model = Sequential()
-     model.add(Dense(v*(c-1), input_dim=v*(c-1), init='normal', activation='relu'))
-     model.add(Dense(1, init='normal'))
-     # Compile model
-     model.compile(loss='mean_absolute_error', optimizer='adam')
-     return model
-
- # define smaller model
- def smaller(v):
-     # create model
-     model = Sequential()
-     model.add(Dense(v*(c-1)/2, input_dim=v*(c-1), init='normal', activation='relu'))
-     model.add(Dense(1, init='normal', activation='relu'))
-     # Compile model
-     model.compile(loss='mean_absolute_error', optimizer='adam')
-     return model
-
- # define deeper model
- def deeper(v):
-  # create model
-  model = Sequential()
-  model.add(Dense(v*(c-1), input_dim=v*(c-1), init='normal', activation='relu'))
-  model.add(Dense(v*(c-1)/2, init='normal', activation='relu'))
-  model.add(Dense(1, init='normal', activation='relu'))
-  # Compile model
-
```

```
        model.compile(loss='mean_absolute_err
        or', optimizer='adam')
-        return model
-
-    # Optimize using dropout and decay
-    from keras.optimizers import SGD
-    from keras.layers import Dropout
-    from keras.constraints import maxnorm
-
-    def dropout(v):
-        #create model
-        model = Sequential()
-        model.add(Dense(v*(c-1),
        input_dim=v*(c-1), init='normal',
        activation='relu',W_constraint=maxnorm
        (3)))
-        model.add(Dropout(0.2))
-        model.add(Dense(v*(c-1)/2,
        init='normal', activation='relu',
        W_constraint=maxnorm(3)))
-        model.add(Dropout(0.2))
-        model.add(Dense(1, init='normal',
        activation='relu'))
-        # Compile model
-        sgd =
        SGD(lr=0.1,momentum=0.9,decay=0.0,
        nesterov=False)
-
        model.compile(loss='mean_absolute_err
        or', optimizer=sgd)
-        return model
-
-    # define decay model
-    def decay(v):
-        # create model
-        model = Sequential()
-        model.add(Dense(v*(c-1),
        input_dim=v*(c-1), init='normal',
        activation='relu'))
-        model.add(Dense(1, init='normal',
        activation='relu'))
-        # Compile model
-        sgd =
        SGD(lr=0.1,momentum=0.8,decay=0.01
        ,nesterov=False)
-
        model.compile(loss='mean_absolute_err
```

```
or', optimizer=sgd)
-       return model
-
-
-   est_list = []
-   #uncomment the below if you want to
    run the algo
-   #est_list = [('MLP',baseline),
    ('smaller',smaller),('deeper',deeper),
    ('dropout',dropout),('decay',decay)]
-
-   for name, est in est_list:
-
-       algo = name
-
-       #Accuracy of the model using all
    features
-       for m,i_cols_list in X_all:
-           model =
    KerasRegressor(build_fn=est, v=1,
    nb_epoch=10, verbose=0)
-
    model.fit(X_train[:,i_cols_list],Y_train)
-           result =
    mean_absolute_error(np.expm1(y_test),
    np.expm1(model.predict(X_test[:,i_cols_
    list])))
-           mae.append(result)
-           print(name + " %s" % result)
-
-       comb.append(algo )
-
-
-   # since we know the outcome, we can
    skip the algorithm and append the result
-   if (len(est_list)==0):
-       mae.append(1168)
-       comb.append("MLP" + " baseline" )
-
-
-
-   print("mae--> %s" % mae)
-   print("comb--> %s" % comb)
-   ##Set figure size
-   plt.rc("figure", figsize=(25, 10))
-
-   #Plot the MAE of all combinations
```

- fig, ax = plt.subplots()
- plt.plot(mae)
- #Set the tick names to names of combinations
- ax.set_xticks(range(len(comb)))
- ax.set_xticklabels(comb,rotation='vertical')
- #Plot the accuracy for all combinations
- plt.show()
-
- #Best estimated performance is MLP=1168
-
-
-

- Classification
- Model Selection
  - Example 1
    -
    - “””
    - Since XGBRegressor is showing the best performance, we can select it as our best model. Therefore, we now need to finalize the model with all of the available data.
    - “””
    -
    -
    - # note, X_train and X_test are both coming from the training set CSV. axis=0 is stacking rows on top of one another.
    - X = np.concatenate((X_train,X_test), axis=0)
    - del X_train
    - del X_test
    - Y = np.concatenate((y_train,y_test),axis=0)
    - del y_train
    - del y_test
    -
    - print("I am here 0 - debug")
    -
    -
    - n_estimators = 1000
    -
    - #Best model definition
    - best_model = XGBRegressor(n_estimators=n_estimators,seed=seed)

```
-       print("I am here 0.0 - debug")
-       best_model.fit(X,Y)
-       print("I am here 0.1 - debug")
-       del X
-       del Y
-       #Read test dataset
-       dataset_test = pd.read_csv("test.csv")
-       print("I am here 0.2 - debug")
-       #Drop unnecessary columns
-       ID = dataset_test['id']
-       dataset_test.drop('id',axis=1,inplace=True)
-
-       #One hot encode all categorical attributes
-       cats = []
-       print("I am here 1 - debug")
-       for i in range(0, split):
-           # label encoding
-           label_encoder = LabelEncoder()
-           label_encoder.fit(labels[i])
-           feature =
        label_encoder.transform(dataset_test.iloc[:,i])
-           feature = feature.reshape(dataset_test.shape[0], 1)
-           #One hot encoding
-           onehot_encoder =
        OneHotEncoder(sparse=False,n_values=len(labels[i])
        )
-           feature = onehot_encoder.fit_transform(feature)
-           cats.append(feature)
-
-       print("I am here 2 - debug")
-       # Making a 2D array from a list of 1D arrays
-       encoded_cats = np.column_stack(cats)
-       del cats
-
-       # Concatenating encoded attributes with continous
        attributes
-       X_test = np.concatenate((encoded_cats,
        dataset_test.iloc[:,split:].values), axis=1)
-       print("I am here 3 - debug")
-       del encoded_cats
-       del dataset_test
-
-       # Making predictions using the best model now
-       predictions = np.expm1(best_model.predict(X_test))
-
-
-
```

-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-