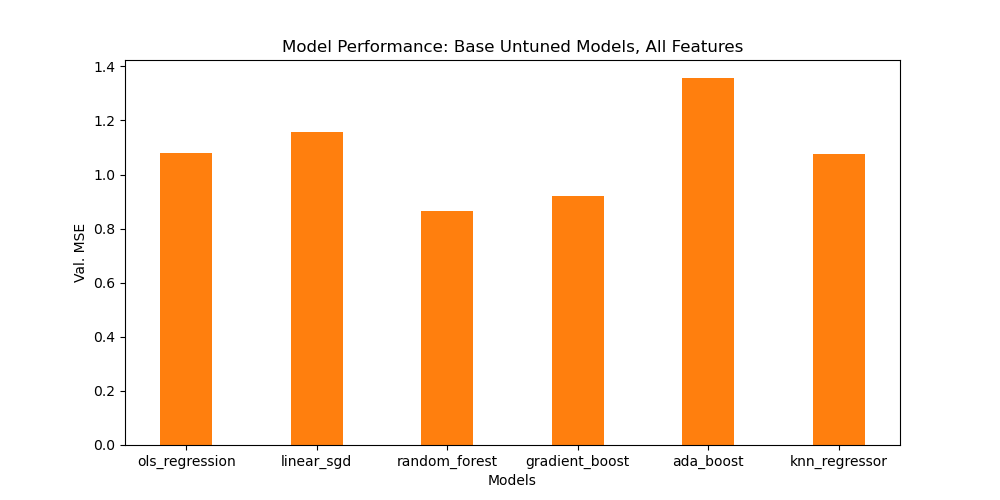
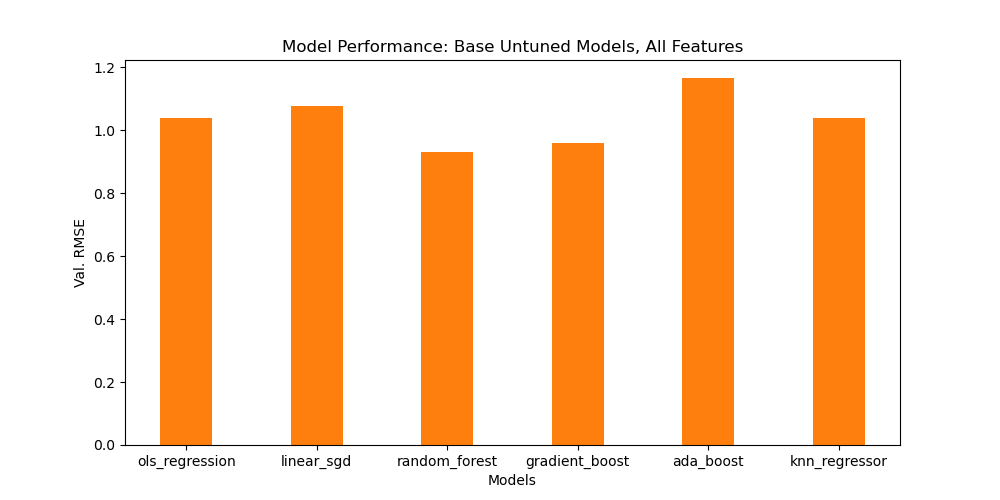
Log:

* 11/4/2021
  + Inspected dataset: looks like we will have high cardinality with many categorical data. Did research into what we can do to reduce the cardinality and reduce dimensionality since just doing OHE will face sparse dataset and have Curse of Dimensionality and may easily overfit our models.
  + Options:
    - Group and cluster our categorical features that have many classes
    - Continue with OHE and use strong L1/L2 regularization
    - Catboost
    - MCA/FAMD
    - Frequency thresholds
    - Grouping / clustering
  + Thinking about doing MCA/FAMD or clustering
* 11/10/21
  + Started pipeline for models with models.py and models\_helper.py
  + Did feature selection with team on: <https://docs.google.com/spreadsheets/d/1qrFCjBWOn3emAx8xtMGC3bpg0CQsx3LMaK1O56ReDO0/edit?usp=sharing>
  + Will use Random Forest to check for Feature importance
    - Then do linear regression with most important features as baseline
    - Then do RandomForest & GradBoost & AdaBoost then tune/cross validate and use best model
* 11/12/21 & 11/13/21
  + Helped out on pre-processing, specifically, worked on encoding of categorical features and feature engineering/extraction.
  + Worked on function to fit and transform actors, director, and writer frequencies on occurrence from training dataset. This is a proxy for quantifying the popularity of cast, writer, or director. If more than one person, we take the average of the frequencies. We also weighted the frequencies by the order of importance that the person played in the movie from the ‘title\_principals.csv’ dataset.
    - The weight can be calculated as linear model where order of 1 is most important with an assigned weighting of 10/10.
    - Order of 2 is assigned weighting of 9/10.
    - Order of 10 is assigned weighting of 1/10.
    - We can calculate the function based on input ‘order’ to output the desired weighting.
    - This is performed in the solve\_linear\_transformation() function in preprocessing\_utils.py where:
      * order \* m + b = weight multiplier, where m and b's are slope and intercept of our linear transformation
      * ie: 1m + b = 10/10, 2m + b = 9/10, 3m + b = 8/10, ...., 10m + b = 1/10
  + Next up is working on the encodings for
    - Genres: genre1, genre2, genre3 will be binary encoded
    - Production company: also frequency based but without order of importance weighting
    - Title: n words, ratio long words, ratio of vowels
    - Description: n words, ratio long words, ratio of vowels, ratio of punctuation, ratio of capital letters, ratio of capital letters after first word
* 11/14/21
  + Finished categorical encoding for:
    - Genres: genre1, genre2, genre3 are binary encoded
      * binary\_encoder\_fit()
      * binary\_encoder\_transform()
      * binary\_encoder()
    - Production company: also frequency based but without order of importance weighting
      * fit\_production\_company\_frequency()
      * transform\_production\_company\_frequency()
    - Title & Description: n words, ratio long words, ratio of vowels, ratio of interesting characters, ratio of capital letters
      * n\_words()
      * ratio\_long\_words()
      * ratio\_vowels()
      * ratio\_interesting\_characters()
      * ratio\_capital\_letters
  + From preprocessing\_utils.py, I wrote approximately 205 lines of actual code (excluding comments). About 36 lines of those I had to use Google to find solutions or code templates that I repurposed for my needs.
  + Next up is to work on actual modeling.
* 11/15/21
  + Started working on modeling part:
    - Created architecture and pipeline to handle modeling part and connect with rest of project modules
    - Created basic random forest regressor to check most important features
  + Changed `genre` encoding from having separate `genre1`, `genre2`, and `genre3` binary encoded labels to just using `genre` where each label is given for each unique tuple of `(genre1, genre2, genre3)` where contents inside tuple are sorted to take different ordering of same 3 genres in different observations into account.
* 11/17/21
  + Have base models trained and validation scores calculated
  + Used LinearRegression(), SGDRegressor(), RandomForestRegressor(), GradientBoostingRegressor(), AdaBoostingRegressor(), KNeighborsRegressor()
  + Here were their MSE and RMSE validation scores:





* + For GridSearch CV, looks like RandomForest and GradientBoosting have the most promise. KNN might be worth a try too to keep.