**NFT Price Prediction Challenge**

1. **Data Preparation Process**

Two processed datasets have been created using the codes “Data Processing 1-NFT.ipynb” and “Data Processing 2-NFT.ipynb” to give outputs **“data\_processed1.csv”** and **“data\_processed2.csv”** respectively

**Steps Used in Data Processing:**

1. **Merge Datasets:** All the given datasets are combined using a left join on "collection\_id"
2. **Handling Missing Values:** For openrarity\_rank and openrarity\_max\_rank, the unique values are less than 10% of the total population and can be treated as discrete/categorical. Therefore they were converted to categorical and introduced a bin for missing. openrarity\_score can be filled with zero as the scores are not available.
3. **Drop Redundant Features:** Features with only one unique value have been dropped. 'platform\_fees', 'has\_website', 'has\_own\_twitter'
4. **Exploratory Data Analysis**: Most of the features are discrete, which leaves the opportunity to treat them as categorical based on performance.
   1. **Date Time features**: 'last\_sale\_date', 'creation\_date' were converted to DateTime
   2. **New Date features**: 'sale\_year' and 'sale\_month' were created. Also, the gap between creation and sale dates was computed.
   3. **Date features to categorical**: I have observed that these sales and creations had happened only on a few dates constituting <1% of the total population. Hence converted the date variables to categorical and all dates have multiple records.
   4. **Discrete features**: All numerical features with less than 100 unique records(0.02%). However, this method has not improved performance and has not been used in the final solutions.
   5. **Distribution of Numerical Features**: No particular distribution was initially shown but a log transformation has improved slightly. Log Transformation of NFT price has shown a chi-square distribution as expected. Log transformation has been the key to this problem and has improved the model performance significantly.
   6. **Outliers**: There were a lot of outliers however I have not used any distance-based algorithms. Moreover, Tree based algorithms Like Gradient Boosting & XGBoost perform well in most cases.
   7. **Categorical Features**: Across all the categories for all the variables, the median of NFT price was different indicating that there is a correlation between the category and the prices associated with it.
5. **Feature Engineering:**
   1. **Numerical features:** Alog transformation with a correction to include zeros has been adopted
   2. **Categorical features:** This is where I have gathered all my experiments and applied a bunch of methods. Finally used the below methods
      1. **One Hot Encoding**: This is applied on 'openrarity\_rank'& 'openrarity\_max\_rank'. Initially, rare categories were clubbed, reducing 15556 categories to 2 and 11 to 9 respectively. Had this massive reduction not happened, I would not have used one hot for these features since it would not be parsimonious.
      2. **Count-Based Label Encoding**: The method of ranking the distribution of population across the categories. Is one of the best methods for features for both high and low numbers of features. A variant of this would be just a replacement of the category with its population count. (Please note the original method has created processed output1 “data\_processed1.csv” and a log transformation of this has created “data\_processed2.csv”)
   3. **New features**: I have tried creating a few features listed below but none of them have improved performance and therefore not been used in the final submissions.
      1. Total likes: avg\_likes\*n\_tweets\_in\_range
      2. Total Retweets: avg\_retweets\*n\_tweets\_in\_range
      3. Total Replies: avg\_replies\*n\_tweets\_in\_range
      4. Collection Count: collection count across NFT ID
      5. NFT count: NFT distribution across each collection
      6. total Collection Count: Population distribution of the collection
      7. total NFT count: Population Distribution of NFTs
6. **Scaling**: Scaled using a standardization approach

**B. Modeling Approach:**

The technique used to arrive at the final solution is **Panel Regression Approach.** The idea is if the entire dataset shows no particular patterns, the data can be broken down into sets and fit an algorithm that performs well in each set. Ideally, this should be done on a feature that has multiple rows for a particular nft\_id. However, I have decided to make clusters to create subsets and then fit algorithms on each subset. Clusters were decided using the Elbow Visualizer Method. 25 subsets of data have been created using “data\_processed1.csv” and “data\_processed2.csv”

Functions to apply a panel regression on any algorithm have been created( *panel*,*panel\_tune*,*panel\_pre\_tune*).

Three approaches(functions) of panel regression are:

1. Single Algorithm Based Panel Regression(function is *panel*): A single algorithm, ensemble method is used to predict the test data in all the subsets. Some subsets might not have any train/test data, these are skipped without training and prediction. The subsets with both train and test data available have been fitted with a model. The test subsets that are left over are combined at the end and fit with the rest of the data(including test data subsets that were fit earlier)
   1. Parameters: data(Full dataset), algorithm, variable(used as a panel, “cluster” in this case), error\_func, pred\_features(last\_sale\_price), test(Test data Set)
2. Tuned Panel Regression(function is *panel\_tuneI)*: Instead of fitting an algorithm with the same hyperparameters, RandomizedSearchCV is passed as an additional parameter in the function which tunes the hyperparameters according to each subset and fits the algorithm based on tuned hyperparameters. This method would take an immense amount of time.
   1. Parameters: data(Full dataset), algorithm, variable(used as a panel, “cluster” in this case), error\_func, pred\_features(last\_sale\_price), test(Test data Set), hyp(Hyper Parameter Tuning Method, eg: RandomSearchCV)
3. Pre-Tuned Panel Regression(function is *panel\_pre\_tune*): The algorithms with their hyperparameters obtained from the above method are stored and used to fit all the subsets. This will reduce the time.
   1. Parameters: data(Full dataset), Tuned(Dictionary of Pre-tuned Algoritms to fit each subset), variable(used as a panel, “cluster” in this case), error\_func, pred\_features(last\_sale\_price), test(Test data Set)
4. **Models on Processed Data 1**

The code “**NFT Price Prediction 1.ipynb”** is used with data input **“data\_processed1.csv”** to produce three primary solutions named **“solution1.csv”**, **“solution2.csv”**, & **“solution3.csv”**

**Steps Used in Modeling:**

1. **Basic OLS fit:** Train and test data is split using train\_test\_split and fit an OLS just to get an understanding of the importance of features through p-value, AIC/BIC. This will help to drop/transform features or try another encoding technique
2. **Algorithms:** Two algorithms are used to get the three primary solutions.
   1. **Gradient Boosting Method**: Main hyperparameters are max\_depth, min\_samples\_split, n\_estimators, learning\_rate
   2. **XGBoost:** n\_estimators, max\_depth, booster, learning\_rate,min\_child\_weight, base-score.
3. **Solution 1**: Function “panel” is used with the algorithm *GradientBoostingRegressor(max\_depth=5, min\_samples\_split=200, n\_estimators=50,random\_state=42)* using “cluster” as the panel
4. **Solution 2**: Function “pane\_pre\_tunel” is used with the GBM algorithm with multiple sets of hyperparameters.(All dictionary keys are cluster numbers with both train and test data sets)

pre\_tuned\_gbm={1:GradientBoostingRegressor(max\_depth=4, min\_samples\_split=150, n\_estimators=40,

random\_state=42),

3:GradientBoostingRegressor(max\_depth=4, min\_samples\_split=250, n\_estimators=50,

random\_state=42),

6:GradientBoostingRegressor(max\_depth=6, min\_samples\_split=250, n\_estimators=50,

random\_state=42),

7:GradientBoostingRegressor(max\_depth=5, min\_samples\_split=250, n\_estimators=50,

random\_state=42),

8: GradientBoostingRegressor(max\_depth=4, min\_samples\_split=100, n\_estimators=40,

random\_state=42),

11:GradientBoostingRegressor(max\_depth=4, min\_samples\_split=100, n\_estimators=40,

random\_state=42),

16: GradientBoostingRegressor(max\_depth=4, min\_samples\_split=250, n\_estimators=50,

random\_state=42),

24:GradientBoostingRegressor(max\_depth=6, min\_samples\_split=250, n\_estimators=50,

random\_state=42)}

1. **Solution 3**: Function “pane\_pre\_tunel” is used with the XGBoost algorithm with multiple sets of hyperparameters. (All dictionary keys are cluster numbers with both train and test data sets)

pre\_tuned\_xgb={1:XGBRegressor(gpu\_id=1, interaction\_constraints='', max\_depth=2,

min\_child\_weight=4, monotone\_constraints='(1,-1)', n\_jobs=4,

num\_parallel\_tree=1, tree\_method='exact', validate\_parameters=1,

verbosity=3),

3:XGBRegressor(base\_score=0.25, booster='gblinear', gpu\_id=1,

interaction\_constraints='', learning\_rate=0.2,

monotone\_constraints='(1,-1)', n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3),

6:XGBRegressor(base\_score=1, booster='gblinear', gpu\_id=1,

interaction\_constraints='', learning\_rate=0.2, max\_depth=5,

min\_child\_weight=3, monotone\_constraints='(1,-1)',

n\_estimators=900, n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3),

7:XGBRegressor(base\_score=0.25, booster='gblinear', gpu\_id=1,

interaction\_constraints='', learning\_rate=0.2,

monotone\_constraints='(1,-1)', n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3),

8: XGBRegressor(base\_score=1, booster='gblinear', gpu\_id=1,

interaction\_constraints='', learning\_rate=0.2, max\_depth=5,

min\_child\_weight=3, monotone\_constraints='(1,-1)',

n\_estimators=900, n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3),

11:XGBRegressor(booster='gblinear', gpu\_id=1, interaction\_constraints='',

max\_depth=8, min\_child\_weight=4, monotone\_constraints='(1,-1)',

n\_estimators=1100, n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3) ,

16: XGBRegressor(base\_score=1, booster='gblinear', gpu\_id=1,

interaction\_constraints='', learning\_rate=0.2, max\_depth=5,

min\_child\_weight=3, monotone\_constraints='(1,-1)',

n\_estimators=900, n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3),

24:XGBRegressor(base\_score=0.25, booster='gblinear', gpu\_id=1,

interaction\_constraints='', learning\_rate=0.2,

monotone\_constraints='(1,-1)', n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3)}

1. **Models on Processed Data 2:**

The code “**NFT Price Prediction 2.ipynb”** is used with data input **“data\_processed1.csv”** to produce three primary solutions named **“solution4.csv”** & **“solution5.csv”**

**Steps Used in Modeling:**

1. **Basic OLS fit:** Train and test data is split using train\_test\_split and fit an OLS just to get an understanding of the importance of features through p-value, AIC/BIC. This will help to drop/transform features or try another encoding technique
2. **Algorithms:** Two algorithms are used to get the three primary solutions.
   1. **Extra Trees Method:** Main hyperparameters are max\_depth, min\_samples\_leaf, min\_samples\_split. Please note that Extra Trees Predicts different solutions each time when it is fit on the same data set. Hence every run will produce different results however the score as per the problem statement is more than 99% between any two solutions
   2. **Gradient Boosting Method**: Main hyperparameters are max\_depth, min\_samples\_split, n\_estimators, learning\_rate
   3. **XGBoost:** n\_estimators, max\_depth, booster, learning\_rate,min\_child\_weight, base-score.
   4. **Light GBM**: Main hyperparameters are num\_leaves, max\_depth, learning\_rate, min\_data\_in\_leaf, bagging\_fraction, feature\_fraction, lambda\_l1, n\_estimators
3. **Solution 4:** The function “panel” is used with three algorithms and the results are created using a simple average.
   1. **Algorithm 1**: GradientBoostingRegressor(max\_depth=5, min\_samples\_split=200, n\_estimators=50, random\_state=42)
   2. **Algorithm 2**: LGBMRegressor(bagging\_fraction=0.9, bagging\_freq=5, bagging\_seed=7, feature\_fraction=0.75, feature\_fraction\_seed=7, lambda\_l1=2, learning\_rate=0.2, max\_bin=200, max\_depth=4, min\_data\_in\_leaf=10, n\_estimators=80, num\_leaves=40, objective='regression',verbose=-1)
   3. **Algorithm 3:** XGBRegressor(gpu\_id=1, interaction\_constraints='', max\_depth=5,min\_child\_weight=3, monotone\_constraints='(1,-1)', n\_jobs=4,num\_parallel\_tree=1, tree\_method='exact', validate\_parameters=1,verbosity=3)
4. **Solution 5:** The function “panel\_pre\_tune” is used with four algorithms and the results are created by stacking them using a simple mean.(All dictionary keys are cluster numbers with both train and test data sets).

Solution 5, can give different results each time because of Extra Trees with above 99% of score9as per problem statement) between any solutions. This has been demonstrated in the code. Please be assured that the dataset and the algorithms(including Hyper Parameters).

* 1. **Extra Trees:**

pre\_tuned\_etr={0: ExtraTreesRegressor(max\_depth=12, min\_samples\_leaf=20, min\_samples\_split=25,

n\_jobs=4),

1:ExtraTreesRegressor(max\_depth=5, min\_samples\_leaf=80, min\_samples\_split=25,

n\_estimators=55, n\_jobs=4),

2:ExtraTreesRegressor(max\_depth=4, min\_samples\_leaf=40, min\_samples\_split=25,

n\_estimators=55, n\_jobs=4),

6:ExtraTreesRegressor(max\_depth=4, min\_samples\_leaf=40, min\_samples\_split=25,

n\_estimators=40, n\_jobs=4),

7:ExtraTreesRegressor(max\_depth=8, min\_samples\_leaf=80, min\_samples\_split=25,

n\_jobs=4),

10:ExtraTreesRegressor(max\_depth=4, min\_samples\_leaf=20, min\_samples\_split=25,

n\_estimators=40, n\_jobs=4),

16: ExtraTreesRegressor(max\_depth=2, min\_samples\_leaf=10, min\_samples\_split=25,

n\_jobs=4),

24: ExtraTreesRegressor(max\_depth=12, min\_samples\_leaf=20, min\_samples\_split=25,

n\_jobs=4)}

* 1. **Gradient Boosting Method:**

pre\_tuned\_gbm={0:GradientBoostingRegressor(max\_depth=4, min\_samples\_split=100, n\_estimators=40,

random\_state=42),

1:GradientBoostingRegressor(max\_depth=6, min\_samples\_split=125, n\_estimators=60,

random\_state=42),

2:GradientBoostingRegressor(max\_depth=4, min\_samples\_split=250, n\_estimators=50,

random\_state=42),

6: GradientBoostingRegressor(max\_depth=5, min\_samples\_split=150, n\_estimators=50,

random\_state=42),

7:GradientBoostingRegressor(max\_depth=5, min\_samples\_split=125, n\_estimators=55,

random\_state=42),

10: GradientBoostingRegressor(max\_depth=4, min\_samples\_split=100, n\_estimators=40,

random\_state=42),

16: GradientBoostingRegressor(max\_depth=4, min\_samples\_split=250, n\_estimators=50,

random\_state=42),

24: GradientBoostingRegressor(max\_depth=6, min\_samples\_split=100, n\_estimators=80,

random\_state=42)}

* 1. **XGBoost:**

pre\_tuned\_xgb={0:XGBRegressor(base\_score=0.75, booster='gblinear', gpu\_id=1,

interaction\_constraints='', min\_child\_weight=4,

monotone\_constraints='(1,-1)', n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3),

1:XGBRegressor(base\_score=0.25, gpu\_id=1, interaction\_constraints='',

min\_child\_weight=3, monotone\_constraints='(1,-1)', n\_jobs=4,

num\_parallel\_tree=1, tree\_method='exact', validate\_parameters=1,

verbosity=3),

2:XGBRegressor(base\_score=0.25, gpu\_id=1, interaction\_constraints='',

min\_child\_weight=3, monotone\_constraints='(1,-1)', n\_jobs=4,

num\_parallel\_tree=1, tree\_method='exact', validate\_parameters=1,

verbosity=3),

6:XGBRegressor(base\_score=0.75, booster='gblinear', gpu\_id=1,

interaction\_constraints='', min\_child\_weight=4,

monotone\_constraints='(1,-1)', n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3),

7:XGBRegressor(base\_score=0.75, booster='gblinear', gpu\_id=1,

interaction\_constraints='', min\_child\_weight=4,

monotone\_constraints='(1,-1)', n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3),

10: XGBRegressor(base\_score=1, booster='gblinear', gpu\_id=1,

interaction\_constraints='', learning\_rate=0.2, max\_depth=10,

min\_child\_weight=3, monotone\_constraints='(1,-1)',

n\_estimators=1500, n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3),

16:XGBRegressor(base\_score=0.25, booster='gblinear', gpu\_id=1,

interaction\_constraints='', learning\_rate=0.2, max\_depth=10,

min\_child\_weight=2, monotone\_constraints='(1,-1)',

n\_estimators=1500, n\_jobs=4, num\_parallel\_tree=1,

tree\_method='exact', validate\_parameters=1, verbosity=3) ,

24: XGBRegressor(booster='gblinear', gpu\_id=1, interaction\_constraints='',

learning\_rate=0.05, min\_child\_weight=4,

monotone\_constraints='(1,-1)', n\_estimators=1100, n\_jobs=4,

num\_parallel\_tree=1, tree\_method='exact', validate\_parameters=1,

verbosity=3)}

* 1. **Light GBM:**

pre\_tuned\_lgbm={0:LGBMRegressor(bagging\_fraction=0.8, bagging\_freq=5, bagging\_seed=7,

feature\_fraction=0.5, feature\_fraction\_seed=7, lambda\_l1=10,

learning\_rate=0.2, max\_bin=200, max\_depth=12, min\_data\_in\_leaf=50,

n\_estimators=60, num\_leaves=10, objective='regression',

verbose=-1),

1:LGBMRegressor(bagging\_fraction=0.7, bagging\_freq=5, bagging\_seed=7,

feature\_fraction=0.2, feature\_fraction\_seed=7, lambda\_l1=2,

learning\_rate=0.05, max\_bin=200, max\_depth=8, min\_data\_in\_leaf=40,

n\_estimators=80, num\_leaves=40, objective='regression',

verbose=-1),

2:LGBMRegressor(bagging\_fraction=0.9, bagging\_freq=5, bagging\_seed=7,

feature\_fraction=0.5, feature\_fraction\_seed=7, lambda\_l1=10,

max\_bin=200, max\_depth=10, min\_data\_in\_leaf=20, num\_leaves=3,

objective='regression', verbose=-1),

6:LGBMRegressor(bagging\_fraction=0.8, bagging\_freq=5, bagging\_seed=7,

feature\_fraction=0.2, feature\_fraction\_seed=7, lambda\_l1=0,

learning\_rate=0.02, max\_bin=200, max\_depth=10,

min\_data\_in\_leaf=10, n\_estimators=40, num\_leaves=40,

objective='regression', verbose=-1),

7:LGBMRegressor(bagging\_fraction=0.8, bagging\_freq=5, bagging\_seed=7,

feature\_fraction=0.5, feature\_fraction\_seed=7, lambda\_l1=0,

learning\_rate=0.03, max\_bin=200, max\_depth=10,

min\_data\_in\_leaf=40, n\_estimators=50, num\_leaves=5,

objective='regression', verbose=-1),

10:LGBMRegressor(bagging\_fraction=0.9, bagging\_freq=5, bagging\_seed=7,

feature\_fraction=0.5, feature\_fraction\_seed=7, lambda\_l1=2,

learning\_rate=0.05, max\_bin=200, max\_depth=2, min\_data\_in\_leaf=10,

n\_estimators=30, num\_leaves=40, objective='regression',

verbose=-1),

16: LGBMRegressor(bagging\_fraction=0.9, bagging\_freq=5, bagging\_seed=7,

feature\_fraction=0.75, feature\_fraction\_seed=7, lambda\_l1=2,

learning\_rate=0.01, max\_bin=200, max\_depth=2, min\_data\_in\_leaf=30,

n\_estimators=80, num\_leaves=10, objective='regression',

verbose=-1),

24: LGBMRegressor(bagging\_fraction=0.7, bagging\_freq=5, bagging\_seed=7,

feature\_fraction=0.75, feature\_fraction\_seed=7, lambda\_l1=0,

learning\_rate=0.02, max\_bin=200, max\_depth=12,

min\_data\_in\_leaf=10, n\_estimators=200, num\_leaves=30,

objective='regression', verbose=-1)}

**B. Stacking:** Two final solutions are made from the 5 primary solutions and these are selected as the final submissions. The code used is “Stacking NFT.ipynb” with inputs “solution1.csv”, “solution2.csv”, “solution3.csv”, “solution4.csv”, & “solution5.csv”

1. **Final Submission 1(**filename=”result35\_22\_32\_34.csv”)**:**

**Simple Average of “solution1.csv”, “solution3.csv”, “solution5.csv”**

1. **Final Submission 2(**filename=”result38\_M.csv”)**:**

**Simple Average of “solution1.csv”“solution3.csv”, “solution5.csv”, Final Submission 1, GM, SM1, SM2, “solution5.csv”, “solution2.csv”**

* 1. **Final Submission 1:** Simple Average of “solution1.csv”, “solution3.csv”, “solution5.csv”
  2. **GM:** Geometric Mean of “solution1.csv”, “solution3.csv”, “solution5.csv”
  3. **SM1:** Simple Average of “solution1.csv”, “solution3.csv”, “solution5.csv”, **GM, Final Submission 1**
  4. **SM2:** Simple Average of “solution1.csv”, “solution2.csv”, “solution4.csv”,

**C. How to Arrive at the final submissions:**

Run the codes in the below order:

1. **Data Processing 1-NFT.ipynb** 
   1. Inputs:nfts\_train.csv, nfts\_predict.csv, collections.csv, collections\_twitter\_stats.csv
   2. output is “data\_processed1.csv”
2. **Data Processing 2-NFT.ipynb**
   1. Inputs:nfts\_train.csv, nfts\_predict.csv, collections.csv, collections\_twitter\_stats.csv
   2. output: “data\_processed2.csv”
3. **NFT Price Prediction 1.ipynb:**
   1. Inputs: nfts\_predict.csv,data\_processed1.csv
   2. Output: “solution1.csv”,“solution2.csv”,“solution3.csv”
4. **NFT Price Prediction 2.ipynb:**
   1. Inputs: nfts\_predict.csv,data\_processed2.csv
   2. Output: “solution4.csv”,“solution5.csv”
5. **Stacking NFT.ipynb:**
   1. Inputs:Output: “solution1.csv”, “solution2.csv”, “solution3.csv”, “solution4.csv”, & “solution5.csv”
   2. Outputs: result35\_22\_32\_34.csv, result38\_M.csv

**C. Environment details:** Google Colab was used to run all the processes(Python 3 Google Compute Engine backend)

RAM: 12.7GB

Disk Space: 107.7GB

Hardware accelerator: None