Fall 2024-2025

EEE485 – Statistical Learning and Data Analytics

Term Project First Report

Predicting the Performance of Monte-Carlo Tree Search (MCTS) Variants Using Machine Learning

11.20.2024

Student Name	Student ID	Section
Emre Akgül	22102310	Section 01

Introduction

Monte Carlo Tree Search (MCTS) is a popular algorithm for creating intelligent agents in board games. Over the past 20 years, numerous variations of MCTS have been introduced by researchers. However, identifying the most effective variants for different game types remains a significant challenge [1]. Currently the way to measure the success of MCTS variants in each game is to simulate it. This is a very expensive process. My goal in the term project is to implement a good predictor to be an alternative to this expensive process. Through the project I aim to improve the understanding of MCTS variants and predict which MCTS variant will perform well in which task.

Problem Description

The problem is given the features of MCTS agent 1, MCTS agent 2 and a game, predicting the success of agent 1 against agent 2 on the given game. Each game is played by some number of times between agents and the success (utility) of agent 1 is calculated by the formula: (n_games_won - n_games_lost) / n_games. All of the games are two-player, sequential, zero-sum board games with perfect information. Root-mean-square-error (RMSE) between predicted and ground-truth utility of agent 1 will be used for measuring the performance of predictors.

Dataset Description

The dataset for the problem is taken from the UM - Game-Playing Strength of MCTS Variants contest currently active on Kaggle [1]. Dataset has 813 features and 4 different target variables. Target variables include utility_agent_1, num_wins_agent_1, num_draws_agent_1, num_losses_agent_1. The main aim is predicting the utility of agent 1 but other target variables can be used for obtaining more information as well. Since there are a lot of features I won't go in detail in this section. I will overview the most important features later. There are over 500 binary categorical feature in the dataset together with other possibly categorical features.

Machine Learning Methods I will Implement:

Since goal is to predict utility_agent_1 a continuous numerical value between -1 and 1 the task is a regression task. To estimate this numerical value, I will implement Linear regression, random forest regressor and gradient boosting regressor algorithms.

Reasons for Selecting Linear Regression:

- Simple to implement interpretable algorithm that model linear relationships between features and the target.
- Provides a good baseline.
- Computationally efficient.
- Lasso Regression is useful for feature elimination.
- As our professor said if something does not work on linear regression to some extent, it is waste of time.

Reasons for Selecting Random Forest Regressor:

- Can capture nonlinear relationships between features and target values.
- Flexibility for bias-variance tradeoff. Easy not to overfit.
- Works good with numerical and categorical features.
- Works good with high dimensional data.

Reasons for Selecting Gradient Boosting:

- High accuracy, for tabular data in practice it is even better than the neural networks.
- Due to its high model complexity, it can capture complex relationships between features and target variables.
- Highly customizable with a range of parameters.

- Works good with numerical and categorical features.
- Works good with high dimensional data.

Limitations of Linear Regression:

- Can not capture nonlinear relationships.

Limitations of Random Forest Regressor:

- Computationally expensive. Given our big dataset it would take long to train and evaluate algorithms.

Limitations of Gradient Boosting:

- Computationally it is even more expensive than Random Forest Regressor.
- Prone to overfitting.

Expected Challenges:

- The biggest challenge for this project is the high dimensionality of the data. Various feature selection methods will be implemented for overcoming this challenge. In addition, dimensionality reduction techniques like PCA will be implemented. Also, I selected random forest regressor and gradient boosting for dealing with this high dimensionality problem.
- Another expected challenge for this project is handling large numbers of categorical features. Deciding between encoding the features for getting better results is a non-trivial mission. One hot encoding will be used for these features.
- High cardinality features also create a threat. There are five object columns which are GameRuleset, agent1, agent2, EnglishRules, LudRules. Feature engineering these columns for getting meaningful features is a hard but rewarding process.
- Correlation between features might be another threat, especially to the Linear Regression methods. Since the dataset has a lot of features, most likely there are some features with high correlations. Effectively eliminating them is a problem.
- Computational constraints are another problem. Dataset includes a quarter of a million data points with approximately 800 features. Working on this dataset is computationally hard.

Contribution of Each Group Member:

This is an individual project all the work will be done by Emre Akgül.

Simulation Setup:

All implementations will be made in Python programming language. The NumPy library will be used for numerical operations and the Pandas library will be used for data reading and interpretation. Additionally, matplotlib will be used for visualization. I’ve got an intel core i7 11th generation CPU and OS is ubuntu 22.04, python version 3.11. I won’t use GPU.

Work Done

1. **Train-Test Split**
   After reading the data, I divided it into two: train and test. For testing, I reserved 20 percent of the data so that it would not be viewed until the final stage of the project. I started working with the train set.
2. **Exploratory Data Analysis: Understanding Assignment and Features**
   Defined objective and evaluation metric. Made an overview of the dataset. Here are the key insights obtained from exploratory data analysis part one.
   - There are 186588 instances of (agent_1, agent_2, game) triple described by 813 different features.

- Behaviour, StateRepetition, Duration, Complexity, BoardCoverage, GameOutcome, StateEvaluation, Clarity, Decisiveness, Drama, MoveEvaluation, StateEvaluationDifference, BoardSitesOccupied, BranchingFactor, DecisionFactor, MoveDistance, PieceNumber, ScoreDifference columns are NaN for all 186588 instance. There are no other missing value in the data. Dropping those columns would be enough for handling missing data. There are 18 features with whole missing values. They will be discarded.
- There are 607 integer columns, 201 float columns and 5 object columns. Some of integer and float columns can be behaved as categorical or ordinal features. There are 382 features that are binary. GameRulesetName, agent1, agent2, EnglishRules, LudRules are objects. They will be inspected more on high cardinality features section
- GameRulesetName have 1377 number of unique values.
  There is no obvious relation between the different values of GameRulesetName. Can be discarded.
- agent1 and agent2 both have 72 unique values which are same.
  They all have the same structure: Both are 4 tuple of different features.
  4 different columns for each agent will be obtained by applying feature engineering to this feature.
  After feature engineering we get 8 different columns: p1_selection, p1_exploration, p1_playout, p1_bounds, p2_selection, p2_exploration, p2_playout, p2_bounds with 4,3, 3, 2, 4, 3, 3, 2 unique values in order. Then we can drop agent columns.
- GameRulesetName have 1328 number of unique values.
  It is long strings of the rules of the games. Can be discarded.
- LudRules have 1373 number of unique values.
  It is a structured description of rules. Can be discarded.
- There are 198 features that have constant value in total. They will be discarded.
- There are 33 duplicated features. One of each will be discarded.
- There are 10 fully correlated - fully uncorrelated features. One of each will be discarded.

  Summary of work done on EDA part 1:
  The objective and evaluation metric are defined. In the codes on Appendix features with high missing values are dropped. Unnecessary high cardinality columns are dropped, from usefull high cardinality columns the information is extracted. Agent columns were initially have 72 unique values, now 4 columns with 4, 3, 3, 2 unique values give all information it gives. Redundant features are dropped. 813 initial features are reduced to the 562 features without loss of information.

3. **Linear Regression Implemented**
   - Implemented a highly customizable linear regression model which takes parameters for fitting method, regularization, gradient descent parameters and loss functions.
   - Fit method is implemented for Ordinary Least Squares (OLS) and Gradient Descent. OLS can calculate the best fit for both linear regression and Ridge regression using closed form solution. It does not support Lasso Regression because there is no closed form solution.
   - Gradient descent iteratively updates weights to minimize the loss function. Three variants for the gradient descent implemented: Batch Gradient Descent, Stochastic Gradient Descent and Mini-Batch Gradient Descent. Gradient descent also supports linear regression, Lasso Regression and Ridge Regression.

4. **Preprocessor Implementation**
    - A preprocessor class is implemented for normalization, standardization and one-hot encoding for the features.
    - The preprocessor can fit the data, transform the data or can do both sequentially.
    - It first one-hot encode the features that have dtype category, then normalize then standardize based on preference.
    - The preprocessor is ideal for preparing datasets for machine learning models by ensuring consistent scaling, encoding, and transformation, especially in workflows with mixed numerical and categorical data.
    - It helps prevent data leakage. Using this preprocessor, it is easier to fit transform the training data and just transform the test data.

5. **Utilization Functions Implementation**
    - Train_test_split function implemented for easier split.
    - RMSE implemented for repeated calculations for validation.
    - K-Fold cross validation function is implemented.

6. **Linear Regression Baseline**
    - Re-applied initial preprocessing which all the unnecessary features are dropped without loss of information.
    - Then applied one-hot encoding and normalization to the dataset using preprocessor implementation.
    - Trained model on OLS, Lasso and Ridge Regressions. Obtained baseline results for these models.
    - Performance is evaluated using root means squared error.
    - In summary this baseline will be used later for checking the improvements that are done in regression task.

7. **Exploratory Data Analysis: Correlation Check with Features and Target**
    - Correleation between features and all the target variables are checked.
    - From correlation check with utility_agent1 AdvantageP1 feature with 0.44 correlation shown as most important feature. The feature with second highest absolute correlation is PiecesPlacedOutsideBoard with 0.09 correlation.
    - From correlation check with num_wins_agent1 it is observed that AdvantageP1 and Completion have high correlation with winning rate of the agent 1. DurationMoves, DurationTurns, Timeouts, Drawishness have all lower than -0.24 correlation due to their lead to the draw in the game.

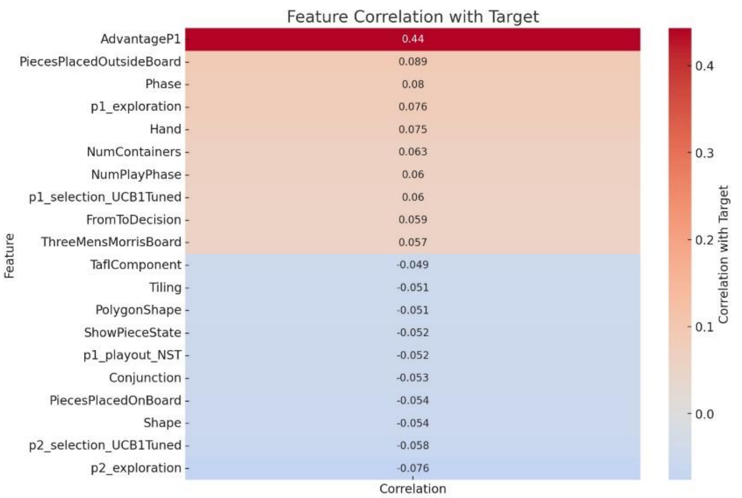

Figure 1: Highest correlated 20 features

    - While checking correlations with num_draws_agent1 it is observed that features are highly correlated with this target variable. There are 6 features that have higher absolute correlation than 0.5.
    - Timeouts, Drawishness have high negative correlation with num_losses_agent1 due to their correlation with draws. AdvantageP1 also have the highest negative

correlation with num losses of the agent1. It also have high correlation with Completion like num_wins_agent_1.

Insights from the Exploratory Data Analysis Part 2:

- AdvantageP1 consistently stands out as the most critical feature across utility, win, and loss predictions, it shows its role in determining game outcomes as expected.
- Features like DurationTurns, Drawishness, and Completion strongly affect the chances of drawing or losses, often pointing to longer or more complex games.
- High negative correlation of Completion with both wins and losses suggest it led to unbalanced chaotic games.
- High number of correlated features to the number of draws target variable suggest there might be ways to use this relation for feature engineering other columns.
- Another models can be trained for predicting the number of drawings and this can be used as a feature in the main prediction algorithm.

	Feature	Correlation
390	Drawishness	0.677952
391	Timeouts	0.614710
378	DurationTurns	0.555967
377	DurationMoves	0.532691
382	GameTreeComplexity	0.486117
392	OutcomeUniformity	0.462496

Figure 2: High correlations on features and number of draws.

8. **Exploratory Data Analysis: Mutual Information Check with Features and Target**
    - MovesPerSecond, PlayoutsPerSecond, DurationActions, DurationMoves, GameTreeComplexity have high mutual information for each target variable.
    - This suggests that these features can be important for models to use for predicting.

9. **Feature Eliminator Implementations**
    - I mentioned that high data dimensionality would be a problem.
    - Implemented a feature eliminator interface for providing general structure for the Feature Elimination method implementations.
    - Implemented variance threshold eliminator that eliminates the features that have less variance than given threshold. It assumes less variance means less information which is not always true.
    - Implemented correlation eliminator that can both eliminate the features to the given number of features or eliminate based on given correlation ranking threshold. I used the correlation ranking formula we learned in the class.
    - Implemented Lasso Regression Eliminator that eliminates the features that have less weight than given threshold, using given lasso regression parameter.

10. **Feature Elimination**
    - The baseline model uses 562 features and its RMSE is 0.5191 on validation set.

- Variance Thresholding with 0.01 variance threshold obtained 0.5316 on validation set by using 336 features.
- Variance Thresholding with 0.1 variance threshold obtained 0.6005 on validation set by using 135 features.
- Correlation Thresholding with 0.01 correlation threshold obtained 0.5340 on validation set by using 269 features.
- Correlation Thresholding with 0.03 correlation threshold obtained 0.5460 on validation set by using 76 features.
- Lasso Eliminator with regularization coefficient 1 and 0.1 threshold obtained 0.5359 RMSE by using 346.
- From these results it is seen that correlation thresholding with 0.03 is the most effective method for now. It balances the bias and variance quite well by using less features than others and not having significant increase in the validation set loss.

Time Takes Algorithms to Train:

- Currently linear regression with OLS can train on the data in matters of seconds. It takes approximately 2 seconds to train.
- Linear regression with gradient descents train time vary a lot based on given learning rate, gradient descents type and number of epochs.
- Since I don't yet implement Random Forest and Gradient Boosting, I do not have data for their training time.

Discussion on The Performance of Considered Methods

1. Summary of Methods

Linear Regression (Implemented for Project):

- A baseline model that assumes a linear relationship between features and the target variable.
- It is as fast as it gets. Runs in two seconds.
- Achieved a competitive result, but its simplicity makes it less suited for capturing complex non-linear relationships in the data.

Random Forest (Using Sklearn for Competition):

- An ensemble method leveraging decision trees and bagging.
- Captures non-linear relationships and interactions between features well.

Gradient Boosting (Using Sklearn for Competition):

- Same as random forest but have higher capacity and have more parameters to tune for bias-variance tradeoff.

2. Performance Metrics

Linear Regression:

- Best result so far is 0.51 RMSE on the fixed dataset.

Random Forest:

- Best result so far without parameter tuning is 0.46 RMSE on the fixed dataset.

Gradient Boosting:

- Using the ensemble method of XGBoost, LightGBM and CatBoost obtained 0.33 RMSE on the same dataset. Without parameter optimization I get 0.44 RMSE on XGBoost.

3. Key Observations

Gradient Boosting which is the method with the highest capacity obtained the best results due to complex relationships in the dataset. I am predicting similar results after implementing the random forest regressor and gradient boosting regressor. In the end I will perform one last cross validation and measure performance of the proposed models on the test set to see which perform best.

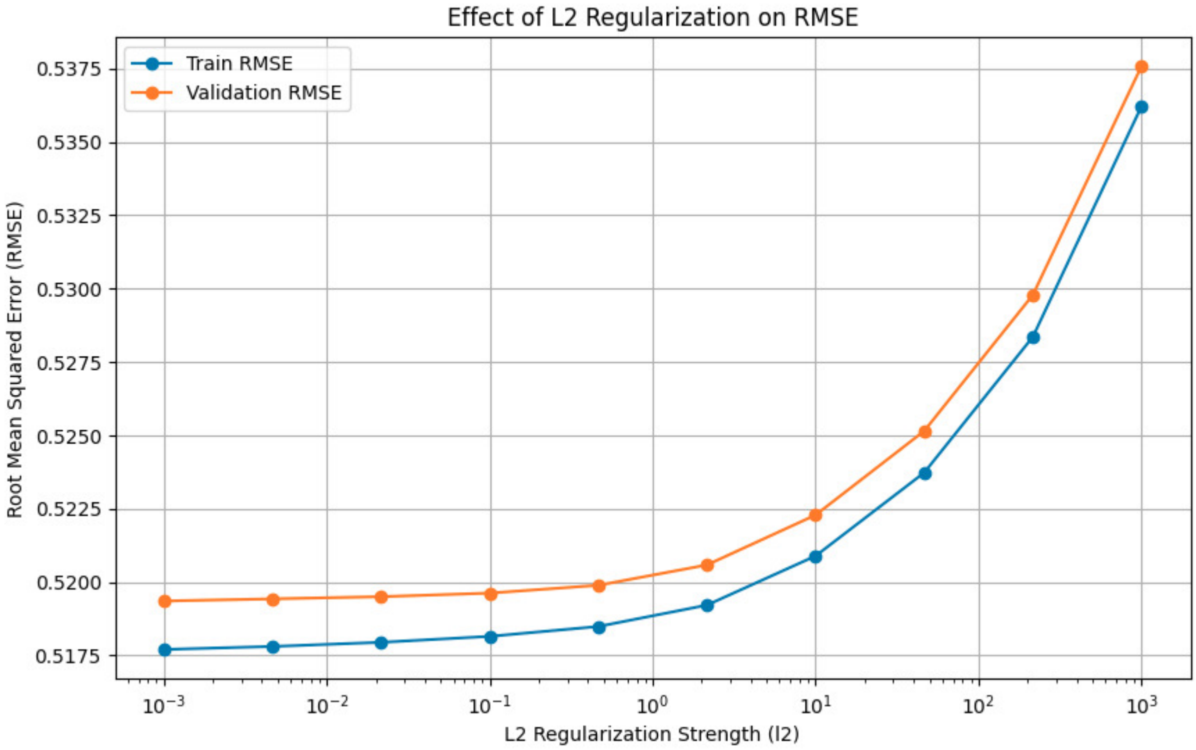

Figure 3: RMSE change of Ridge Regression with change of regularization parameter

References

1- **Kaggle. (2024).** ***UM - Game-Playing Strength of MCTS Variants*****. Retrieved from** **https://www.kaggle.com/competitions/um-game-playing-strength-of-mcts-variants/overview**

Appendix

Appendix A:

Splitting Data to Test: I won’t touch test created here until final report.

```
import pandas as pd
In [ ]:
# Load whole data
file_path = '../Data/data.csv'
data = pd.read_csv(file_path, index_col='Id')

# Split the data into training and testing sets (80% train, 20% test)
test_size = int(0.2 * len(data))
data_shuffled = data.sample(frac=1, random_state=42)

train_data = data_shuffled.iloc[:-test_size]  # First 80% for training
test_data = data_shuffled.iloc[-test_size:]  # Last 20% for testing

train_data.to_csv('../Data/train.csv')
test_data.to_csv('../Data/test.csv')
```

Appendix B:

Exploratory Data Analysis Part 1: Understanding Assignment and Features

1- Objective:

Predict the performance of one Monte-Carlo Tree Search (MCTS) variant against another in a given game.

The target column is the utility_agent1 column, which ranges from -1 (all loss for agent1) to 1 (all wins for agent1).

Draw is also possible. Beside utility_agent1, (num_wins_agent1, num_draws_agent1, num_losses_agent1) also given in the training dataset.

They can be usefull for trying different techniques.

2- Evaluation Metric:

Root Mean Square Error (RMSE)

3- Dataset Overview

3.1- Shape of Data

Each row is defined by 2 agent and a game.

There are 186588 instance of (agent_1, agent_2, game) triple described by 813 different features.

3.2- Missing Data

Behaviour, StateRepetition, Duration, Complexity, BoardCoverage, GameOutcome, StateEvaluation, Clarity, Decisiveness, Drama, MoveEvaluation, StateEvaluationDifference, BoardSitesOccupied, BranchingFactor, DecisionFactor, MoveDistance, PieceNumber, ScoreDifference columns are NaN for all 186588 instance. There are no other missing value in the data. Dropping those columns would be enough for handling missing data.

There are 18 features with whole missing values. They will be discarded.

3.3- Data Types

There are 607 integer columns, 201 float columns and 5 object columns.

Some of integer and float columns can be behaved as categorical or ordinal features. There are

382 features that are binary.
GameRulesetName, agent1, agent2, EnglishRules, LudRules are objects. They will be inspected more on high cardinality features section.

3.4- High Cardinality Features

3.4.1- GameRulesetName
GameRulesetName have 1377 number of unique values.
There is no obvious relation between the different values of GameRulesetName. Can be discarded.

3.4.2- agent1
agent1 and agent2 both have 72 unique values which are same.
They all have the same structure: MCTS--<EXPLORATION_CONST>--<SCORE_BOUNDS>
4 different columns for each agent will be obtained by applying feature engineering to this feature.
After feature engineering we get 8 different columns: p1_selection, p1_exploration, p1_playout, p1_bounds, p2_selection, p2_exploration, p2_playout, p2_bounds with 4,3, 3, 2, 4, 3, 3, 2 unique values in order. Then we can drop agent columns.

3.4.3- agent2
Same with agent1

3.4.4- EnglishRules
GameRulesetName have 1328 number of unique values.
It is long strings of the rules of the games. Can be discarded.

3.4.5- LudRules
LudRules have 1373 number of unique values.
It is a structured description of rules. Can be discarded.

4- Redundant Features

4.1- Constant Features
There are 198 features that have constant value in total. They will be discarded.

4.2- Duplicated Features
There are 33 duplicated features. One of each will be discarded.

4.3- Fully Correlated - Fully Uncorrelated Features
There are 10 fully correlated - fully uncorrelated features. One of each will be discarded.

Summary of What We Have Done
The objective and evaluation metric are defined. In the codes on Appendix features with high missing values are dropped. Unnecessary high cardinality columns are dropped, from usefull high cardinality columns the information is extracted. Agent columns were initially have 72 unique values, now 4 columns with 4, 3, 3, 2 unique values give all information it gives. Redundant features are dropped. 813 initial features are reduced to the 562 features without loss of information.
Code Part:
3- Dataset Overview
In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
# Read the data
train = pd.read_csv('../Data/train.csv', index_col='Id')
```
3.1- Shape of Data

In [3]:
```
# Dimensions
train.shape
```
Out[3]:
```
(186588, 813)
```
3.2- Missing Values

In [4]:
```
# Check number of missing values by column
missing_values_count = train.isnull().sum()
missing_values_count[missing_values_count > 0]
```
Out[4]:
```
Behaviour                    186588
StateRepetition              186588
Duration                     186588
Complexity                   186588
BoardCoverage                186588
GameOutcome                  186588
StateEvaluation              186588
Clarity                      186588
Decisiveness                 186588
Drama                        186588
MoveEvaluation               186588
StateEvaluationDifference    186588
BoardSitesOccupied           186588
BranchingFactor              186588
DecisionFactor               186588
MoveDistance                 186588
PieceNumber                  186588
ScoreDifference              186588
dtype: int64
```
In [5]:
```
# Missing value feature names
missing_features = missing_values_count[missing_values_count > 0].index
len(missing_features)
```
Out[5]:
```
18
```
In [6]:
```
# Drop missing value features
train.drop(missing_features, axis=1, inplace=True)
```
3.3- Data Types

In [7]:
```
# data types of features
train.dtypes.value_counts()
```
Out[7]:
```
int64      607
float64    183
object       5
Name: count, dtype: int64
```
In [8]:
```
# number of features that have only 2 different values
binary_features = train.nunique()[train.nunique() == 2]
```

```
len(binary_features)
```
Out[8]:
```
382
```
In [9]:
```
# features that are object
object_features = train.select_dtypes(include=['object']).columns
object_features
```
Out[9]:
```
Index(['GameRulesetName', 'agent1', 'agent2', 'EnglishRules', 'LudRules'], dtype='object')
```
3.4- High Cardinality Features

3.4.1- GameRulesetName

In [10]:
```
train["GameRulesetName"].nunique()
```
Out[10]:
```
1377
```
In [11]:
```
train["GameRulesetName"].value_counts()
```
Out[11]:
```
GameRulesetName
Pathway                                      181
Tule_Paid                                    174
Greater_Even_Loss                            173
Sweep_Burrow                                 173
Ludus_Latrunculorum8x8_Seega_Rules_Suggested     172
                                            ...
Pancha_Keliya                                 55
Bheri_Bakhri                                  54
58_HolesTab_Parallel_Connections_D6_Suggested     53
58_HolesTab_Unmarked_Suggested                46
Faraday                                        3
Name: count, Length: 1377, dtype: int64
```
3.4.2- agent1

In [12]:
```
train["agent1"].nunique()
```
Out[12]:
```
72
```
In [13]:
```
train["agent1"].value_counts()
```
Out[13]:
```
agent1
MCTS-UCB1Tuned-0.1-NST-false                     2875
MCTS-UCB1GRAVE-0.1-Random200-false               2871
MCTS-UCB1Tuned-0.1-MAST-false                    2859
MCTS-UCB1GRAVE-0.1-MAST-false                    2848
MCTS-UCB1-0.6-MAST-false                         2839
                                                 ...
MCTS-ProgressiveHistory-0.6-NST-true             2350
MCTS-ProgressiveHistory-1.41421356237-NST-true    2347
MCTS-UCB1-1.41421356237-Random200-true           2337
MCTS-UCB1Tuned-1.41421356237-MAST-true           2322
MCTS-ProgressiveHistory-0.1-NST-true             2290
Name: count, Length: 72, dtype: int64
```
3.4.2- agent2

In [14]:
```
train["agent2"].nunique()
```

Out[14]:
72
In [15]:
```
train["agent2"].value_counts()
```
Out[15]:
```
agent2
MCTS-UCB1GRAVE-0.1-MAST-false            2913
MCTS-UCB1Tuned-0.1-MAST-false            2892
MCTS-UCB1GRAVE-0.1-Random200-false       2873
MCTS-UCB1Tuned-0.1-NST-false             2847
MCTS-UCB1GRAVE-1.41421356237-NST-false   2840
                                         ...
MCTS-UCB1GRAVE-0.6-MAST-true             2360
MCTS-ProgressiveHistory-0.6-NST-true     2356
MCTS-UCB1GRAVE-0.6-NST-true              2337
MCTS-UCB1GRAVE-0.1-NST-true              2335
MCTS-ProgressiveHistory-0.1-NST-true     2300
Name: count, Length: 72, dtype: int64
```
In [16]:
```
# Getting the selection, exploration, playout and bounds from the agent columns

# Function to extract features based on the pattern provided
def extract_features(agent_column):
    selection = agent_column.str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)', expand=True)[0]
    exploration = agent_column.str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)',
expand=True)[1].astype(float)
    playout = agent_column.str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)', expand=True)[2]
    bounds = agent_column.str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)', expand=True)[3]
    return selection, exploration, playout, bounds

# Applying the function to extract features for agent1 and agent2
train['p1_selection'], train['p1_exploration'], train['p1_playout'], train['p1_bounds'] =
extract_features(train['agent1'])
train['p2_selection'], train['p2_exploration'], train['p2_playout'], train['p2_bounds'] =
extract_features(train['agent2'])

train = train.drop(["agent1", "agent2"], axis=1)
```
In [17]:
```
print(train["p1_selection"].nunique(), train["p1_exploration"].nunique(),
train["p1_playout"].nunique(), train["p1_bounds"].nunique())
print(train["p2_selection"].nunique(), train["p2_exploration"].nunique(),
train["p2_playout"].nunique(), train["p2_bounds"].nunique())
```
4 3 3 2
4 3 3 2

3.4.4- EnglishRules

In [18]:
```
train["EnglishRules"].nunique()
```
Out[18]:
1328
In [19]:
```
train["EnglishRules"].value_counts()
```
Out[19]:
EnglishRules

Hares start
first.
3133
First, the giant takes place on all empty sites. The Giant piece can step to an empty site, the dwarves can step only forward to the top of the board. The giant wins if it reaches the bottom sites and the dwarves win if they block the giant to
move.
560
Hare starts
first.
464
First, the giant takes place on all empty sites. The Giant piece can step to an empty site, the dwarves can step only forward to the top of the board. The Giant can capture in hopping in all directions, the dwarves can capture on forwards. The giant wins if the dwarves have no moves and the dwarves win if they capture the
giant.
414
Five pieces per player, which begin on the first five spaces in each track. Four sticks, each with a white side and a yellow side. Throws equal the number of white sides which fall up; when only yellow sides are up the throw equals 6. A throw of 1, 4, or 6 grants another throw to the player. Players perform all of their throws first, and then move pieces according to the values of the throws without subdividing the value of a single throw. Players cannot move their pieces until the throw a 1. Pieces cannot move past one another in the home row. Each piece in the home row must individually be unlocked with a throw of 1 before it can move When a player's piece lands in a space occupied by an opponent's piece, the opponent's piece is removed from the board. When a piece lands on a hole with a line connecting it to another hole, the piece moves forward along that line to the hole on the opposite end it. When a piece reaches the opponent's starting row, it cannot move if there are other pieces belonging to the player outside of this row. A piece resting on a marked space cannot be captured.
362

...
Caerter Crossover 1 with a six-sided
die.
55
3x8 board. Eight pieces per player, which start in the spaces of the outer rows of the board. Four cowrie shells used as dice, the number of mouths face up being the value of the throw. A throw of 1 grants the player another throw. A player must throw 1 for the first move of each of their pieces. Players may only play with one piece out of the home row at a time and cannot move the next of their pieces until the piece being played has been captured. Throws of 1 must be used to move a piece in the home row, if possible. Pieces move from left to right in the player's home row, then from right to left in the central row, left to right in the opponent's home row, and right to left in the central row. When a piece lands on a space occupied by an opponent's piece, the opponent's piece is captured. The player who captures all of the opponent's pieces
wins.
54
Tab on Parallel Connections boards with
D6.
53
Five pieces per player, which begin on the first five spaces in each track. Four sticks, each with a white side and a yellow side. Throws equal the number of white sides which fall up; when only yellow sides are up the throw equals 6. A throw of 1, 4, or 6 grants another throw to the player. Players perform all of their throws first, and then move pieces according to the values of the throws without subdividing the value of a single throw. Players cannot move

their pieces until the throw a 1. Pieces cannot move past one another in the home row. Each piece in the home row must individually be unlocked with a throw of 1 before it can move When a player's piece lands in a space occupied by an opponent's piece, the opponent's piece is removed from the board. When a piece reaches the opponent's starting row, it cannot move if there are other pieces belonging to the player outside of this
row.
46
Goal: End the game with the highest scoring group. A group scores one point for each stone it contains. Definitions: A group, as in Go, is every stone that can be reached from a selected stone through a series of adjacent stones of the same color. Play: Start with a pie offer of 1 to 3 stones of any color combination. (Player 1 does this by making 3 placements, or passes; after which Player 2 may choose to play, or to have the pieces exchanged with the opposite colors) Turns alternate. On a turn, a player places a series of stones, one at a time, (as described below) until no more placements are possible, and then passes. Order of placement matters. Placements are made to empty cells that: -- 1) have more neighbors that are oppositely charged than similarly charged, or -- 2) have 3 or more oppositely charged neighbors. The game ends when neither player can play. The largest group for each player is then scored. In case of a tie the last to place a stone loses. Variants: Exception for Surplus Charge Immediately after placing to a cell with 4 or more oppositely charged neighbors, the player MUST, if possible, place the next stone on an otherwise unplayable empty cell with an equal number of both types of charge around it.   3
Name: count, Length: 1328, dtype: int64

3.4.5- LudRules

In [20]:
train["LudRules"].nunique()
Out[20]:
1373
In [21]:
train["LudRules"].value_counts()
Out[21]:
LudRules
(game "Ludus Coriovalli" (players 2) (equipment { (board (add (merge { (rectangle 1 2) (shift 1 0 (rectangle 1 3)) (shift 3 0 (rectangle 1 2)) (rectangle 2 1) (shift 4 0 (rectangle 2 1)) (shift 4 1.5 (rectangle 2 1)) (shift 0 1.5 (rectangle 2 1)) (shift 0 2.5 (rectangle 1 2)) (shift 1 2.5 (rectangle 1 3)) (shift 3 2.5 (rectangle 1 2)) }) edges:{ {9 5} {5 1} {9 11} {12 2} {13 7} {6 3} {6 7} }) use:Vertex) (piece "Dog" P2 (move Step (to if:(is Empty (to))))) (piece "Hare" P1 (move Step (to if:(is Empty (to))))) }) (rules (start { (place "Hare1" (sites {"C1"})) (place "Dog2" (sites {"A4" "C4" "E4"})) }) (play (forEach Piece)) (end { (if (no Moves P1) (result P2 Win)) (if (or (>= (count Moves) (- (value MoveLimit) 10)) (>= (count Turns) (- (value TurnLimit) 5))) (result P1 Win)) })))
)
292
(game "Shono" (players 2) (equipment { (mancalaBoard 2 6 (track "Track" "1,E,N,W" loop:True)) (regions P1 (sites Bottom)) (regions P2 (sites Top)) (map { (pair P1 FirstSite) (pair P2 LastSite) }) (piece "Seed" Shared) }) (rules (start (set Count 5 to:(sites Track))) (play (move Select (from (sites Mover) if:(< 0 (count at:(from)))) (then (sow if:(and (is In (to) (sites Next)) (or (= (count at:(to)) 2) (= (count at:(to)) 3))) apply:(fromTo (from (to)) (to (mapEntry (mover))) count:(count at:(to))) includeSelf:False backtracking:True)))) (end (if (no Moves Mover) (byScore { (score P1 (+ (count at:(mapEntry P1)) (count in:(sites P1)))) (score P2 (+ (count at:(mapEntry P2)) (count in:(sites P2)))) })))))
)
288
(game "Ludus Coriovalli" (players 2) (equipment { (board (add (merge { (scale 2 1 (rectangle 1 3)) (rectangle 2 1) (shift 4 0 (rectangle 2 1)) (shift 4 1.5 (rectangle 2 1)) (shift 0 1.5 (rectangle 2 1)) (scale 2 1 (shift 0 2.5 (rectangle 1 3))) }) edges:{ {3 7} {5 4} {9 1} {3 1} {1

4} {5 9} {9 7} }) use:Vertex) (piece "Dog" P2 (move Step (to if:(is Empty (to))))) (piece "Hare" P1 (move Step (to if:(is Empty (to))))) }) (rules (start { (place "Hare1" (sites {"B1"})) (place "Dog2" (sites {"A4" "B4" "C4"})) }) (play (forEach Piece)) (end { (if (no Moves P1) (result P2 Win)) (if (or (>= (count Moves) (- (value MoveLimit) 10)) (>= (count Turns) (- (value TurnLimit) 5))) (result P1 Win)) }))
)
280
(game "Pathway" (players 2) (equipment { (board (square 6)) (piece "Disc" Each) }) (rules (play (move Add (to (sites Empty) if:(or (all Sites (sites Around (to) Orthogonal) if:(is Empty (site))) (= 1 (count Sites in:(sites Around (to) Own Orthogonal))))))) (end (if (no Moves Next) (result Next Win))))
)
181
(game "Tule Paid" (players 2) (equipment { (board (concentric Square rings:3 joinCorners:True) use:Vertex) (hand Each) (piece "Marker" Each (move Step (to if:(is Empty (to))) (then (if (is Line 3) (moveAgain))))) }) (rules (start (place "Marker" "Hand" count:12)) phases:{ (phase "Placement" (play (if (is Prev Mover) (move Remove (sites Occupied by:Enemy container:"Board")) (move (from (handSite Mover)) (to (sites Empty)) (then (if (is Line 3) (moveAgain)))))) (nextPhase Mover (all Sites (sites Hand Mover) if:(= 0 (count Cell at:(site)))) "Movement")) (phase "Movement" (play (if (is Prev Mover) (move Remove (sites Occupied by:Enemy container:"Board")) (forEach Piece)))) } (end (if (no Pieces Next) (result Next Loss))))
)
174

...
(game "Pancha Keliya" (players 2) (equipment { (board (rotate 90 (merge { (shift 2.79 10.44 (rotate 135 (rectangle 5 1))) (shift 6.32 11.15 (rotate 45 (rectangle 5 1))) (shift 9 11 (graph vertices:{ {0 0} {-0.75 0.55} {-0.04 1.24} {1 0} } edges:{ { 0 1} { 1 2 } { 2 3 } { 3 0 } })) (shift 9 5 (rectangle 6 1)) (shift 5 5 (rectangle 1 5)) (rectangle 1 9) (shift 4 0 (rectangle 6 1)) })) { (track "Track1" "23,N4,W,N,W,11,7,SW,SE,End" P1 directed:True) (track "Track2" "31,S4,W,N,W,11,7,SW,SE,End" P2 directed:True) }) (piece "Marker" Each (if (= (trackSite Move steps:(count Pips)) End) (move Remove (from)) (if (!= (trackSite Move steps:(count Pips)) Off) (if (or (is Empty (trackSite Move steps:(count Pips))) (and (not (is Friend (who at:(trackSite Move steps:(count Pips))))) (not (is In (trackSite Move steps:(count Pips)) (sites "Protect"))))) (move (from) (to (trackSite Move steps:(count Pips)) (apply if:(is Enemy (who at:(to))) (fromTo (from (to)) (to (mapEntry "Entry" Next)))))))))) (dice d:2 from:0 num:6) (hand Each) (regions "Protect" (sites {27 19 12 10 1})) (regions "SpecialDiceValues" (sites {1 5 6})) (map "Entry" { (pair P1 23) (pair P2 31) }) }) (rules (start (place "Marker" "Hand" count:3)) (play (do (roll) next:(if (and (is In (count Pips) (sites "SpecialDiceValues")) (not (all Sites (sites Hand Mover) if:(= 0 (count Cell at:(site)))))) (or (move (from (handSite Mover)) (to (mapEntry "Entry" Mover) if:(not (is Enemy (who at:(to)))))) (forEach Piece) (then (moveAgain))) (forEach Piece)))) (end (if (no Pieces Mover) (result Mover Win))))
)
55
(game "Bheri Bakhri" (players 2) (equipment { (board (rectangle 3 8) { (track "Track1" "0,E,N1,W,N1,E,S1,W" loop:True P1) (track "Track2" "23,W,S1,E,S1,W,N1,E" loop:True P2) }) (dice d:2 from:0 num:4) (piece "Marker" Each (if (and { (if (!= 0 (state at:(from))) True (= 1 (count Pips))) }) (if (and (not (is Friend (who at:(trackSite Move steps:(count Pips))))) (if (not (is In (from) (sites Mover "Home"))) True (if (is In (trackSite Move steps:(count Pips)) (sites Mover "Home")) True (= (count Pieces Mover in:(difference (sites Board) (sites Mover "Home"))) 0))))) (move (from) (to (trackSite Move steps:(count Pips)) (apply if:(is Enemy (who at:(to))) (remove (to)))) (then (if (and (not (!= 0 (state at:(last To)))) (= 1 (count Pips))) (set State at:(last To) 1))))))) (regions "Home" P1 (sites Bottom))

(regions "Home" P2 (sites Top)) }) (rules (start { (place "Marker1" (sites Bottom)) (place "Marker2" (sites Top)) }) (play (do (roll) next:(if (= 1 (count Pips)) (priority { (forEach Piece (if (and { (if (!= 0 (state at:(from))) True (= 1 (count Pips))) (is In (from) (sites Mover "Home")) }) (if (and (not (is Friend (who at:(trackSite Move steps:(count Pips))))) (if (not (is In (from) (sites Mover "Home"))) True (if (is In (trackSite Move steps:(count Pips)) (sites Mover "Home")) True (= (count Pieces Mover in:(difference (sites Board) (sites Mover "Home"))) 0))))) (move (from) (to (trackSite Move steps:(count Pips)) (apply if:(is Enemy (who at:(to))) (remove (to)))) (then (if (and (not (!= 0 (state at:(last To)))) (= 1 (count Pips))) (set State at:(last To) 1)))))))) (forEach Piece) }) (forEach Piece)) (then (if (= 1 (count Pips)) (moveAgain))))) (end (if (no Pieces Next) (result Next Loss))))
)
54
(game "58 Holes" (players 2) (equipment { (board (graph vertices:{ {9 27} {9 24} {9 21} {9 18} {9 15} {9 12} {9 9} {9 6} {9 3} {9 0} {3 0} {3 2} {3 4} {3 6} {3 8} {3 10} {3 12} {3 14} {3 16} {3 18} {3 20} {3 22} {3 24} {3 26} {3 28} {4 30} {6 31} {8 32} {10 33} {15 27} {15 24} {15 21} {15 18} {15 15} {15 12} {15 9} {15 6} {15 3} {15 0} {21 0} {21 2} {21 4} {21 6} {21 8} {21 10} {21 12} {21 14} {21 16} {21 18} {21 20} {21 22} {21 24} {21 26} {21 28} {20 30} {18 31} {16 32} {14 33} {12 33} } edges:{ {0 1} {1 2} {2 3} {3 4} {4 5} {5 6} {6 7} {7 8} {8 9} {9 10} {10 11} {11 12} {12 13} {13 14} {14 15} {15 16} {16 17} {17 18} {18 19} {19 20} {20 21} {21 22} {22 23} {23 24} {24 25} {25 26} {26 27} {27 28} {28 58} {29 30} {30 31} {31 32} {32 33} {33 34} {34 35} {35 36} {36 37} {37 38} {38 39} {39 40} {40 41} {41 42} {42 43} {43 44} {44 45} {45 46} {46 47} {47 48} {48 49} {49 50} {50 51} {51 52} {52 53} {53 54} {54 55} {55 56} {56 57} {57 58} }) { (track "Track1" { 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 } loop:True P1) (track "Track2" { 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 } loop:True P2) } use:Vertex) (dice d:6 num:1) (regions "Protection" {5 7 9 34 36 38 14 43 19 48 24 53 58}) (piece "Marker" Each (if (if (is In (from) (sites Start (piece (id "Marker" Next)))) (all Sites (sites Occupied by:Mover) if:(is In (site) (sites Start (piece (id "Marker" Next))))) True) (if (= 0 (state at:(from))) (forEach Site (sites (values Remembered "Throws")) (if (or (is Empty (trackSite Move steps:(site))) (and (is Enemy (who at:(trackSite Move steps:(site)))) (not (is In (trackSite Move steps:(site)) (sites "Protection"))))) (move (from) (to (trackSite Move steps:(site)) (apply (forget Value "Throws" (site)))) (then (if (!= (last To) (if (is Mover P1) (mapEntry "ConnectionP1" (last To)) (mapEntry "ConnectionP2" (last To)))) (if (is Empty (if (is Mover P1) (mapEntry "ConnectionP1" (last To)) (mapEntry "ConnectionP2" (last To)))) (move (from (last To)) (to (if (is Mover P1) (mapEntry "ConnectionP1" (last To)) (mapEntry "ConnectionP2" (last To)))))))))))) (forEach Site (sites (values Remembered "Throws")) (if (= 1 (site)) (if (or (is Empty (trackSite Move steps:(site))) (and (is Enemy (who at:(trackSite Move steps:(site)))) (not (is In (trackSite Move steps:(site)) (sites "Protection"))))) (move (from) (to (trackSite Move steps:(site)) (apply (forget Value "Throws" (site)))) (then (and (set State at:(last To) 0) (if (!= (last To) (if (is Mover P1) (mapEntry "ConnectionP1" (last To)) (mapEntry "ConnectionP2" (last To)))) (if (is Empty (if (is Mover P1) (mapEntry "ConnectionP1" (last To)) (mapEntry "ConnectionP2" (last To)))) (move (from (last To)) (to (if (is Mover P1) (mapEntry "ConnectionP1" (last To)) (mapEntry "ConnectionP2" (last To)))))))))))))))))) (map "ConnectionP1" { (pair 5 19) (pair 7 9) (pair 48 34) (pair 38 36) }) (map "ConnectionP2" { (pair 19 5) (pair 9 7) (pair 34 48) (pair 36 38) }) (map "Throw" { (pair 0 6) (pair 1 1) (pair 2 2) (pair 3 3) (pair 4 4) }) }) (rules (start { (place "Marker1" (sites {0 1 2 3 4}) state:1) (place "Marker2" (sites {29 30 31 32 33}) state:1) }) phases:{ (phase "GetMoves" (play (do (roll) next:(move Pass (then (remember Value "Throws" (mapEntry "Throw" (count Pips))))) (then (moveAgain)))) (nextPhase (not (is In (mapEntry "Throw" (count Pips)) (sites {1 4 6}))) "Movement")) (phase "Movement" (play (if (can Move (forEach Piece)) (forEach Piece (then (if (!= 0 (count Sites in:(sites (values Remembered "Throws")))) (moveAgain)))) (move Pass (then (forget Value "Throws" All))))) (nextPhase (not (!= 0 (count Sites in:(sites

(values Remembered "Throws"))))) "GetMoves")) } (end (if (no Pieces Next) (result Next Loss))))) 53
(game "58 Holes" (players 2) (equipment { (board (graph vertices:{ {7 26} {8 23} {8.3 21} {8.3 19} {8.5 17} {8.4 14.5} {8.5 12} {8.6 9.5} {8.6 7} {8.6 4.5} {8.8 2} {8.9 1} {8 0} {7 0.5} {6 2} {5 4.6} {5.1 7} {5 8.5} {5.1 10} {5.1 12.5} {5 15} {5.2 17} {5.4 19} {5.6 21} {5 24} {4 27} {5 29} {6 30} {8 31} {17 26} {16 23} {15.7 21} {15.7 19} {15.5 17} {15.6 14.5} {15.5 12} {15.4 9.5} {15.4 7} {15.4 4.5} {15.2 2} {15.1 1} {16 0} {17 0.5} {18 2} {19 4.6} {18.9 7} {19 8.5} {18.9 10} {18.9 12.5} {19.5 16} {19.6 18.5} {19.6 21} {19.8 23.5} {20 27} {19 30} {18 31} {16 31.5} {14 32} {12 31} } edges:{ {0 1} {1 2} {2 3} {3 4} {4 5} {5 6} {6 7} {7 8} {8 9} {9 10} {10 11} {11 12} {12 13} {13 14} {14 15} {15 16} {16 17} {17 18} {18 19} {19 20} {20 21} {21 22} {22 23} {23 24} {24 25} {25 26} {26 27} {27 28} {28 58} {29 30} {30 31} {31 32} {32 33} {33 34} {34 35} {35 36} {36 37} {37 38} {38 39} {39 40} {40 41} {41 42} {42 43} {43 44} {44 45} {45 46} {46 47} {47 48} {48 49} {49 50} {50 51} {51 52} {52 53} {53 54} {54 55} {55 56} {56 57} {57 58} }) { (track "Track1" { 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 } loop:True P1) (track "Track2" { 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 } loop:True P2) } use:Vertex) (dice d:2 from:0 num:4) (piece "Marker" Each (if (if (is In (from) (sites Start (piece (id "Marker" Next)))) (all Sites (sites Occupied by:Mover) if:(is In (site) (sites Start (piece (id "Marker" Next))))) True) (if (= 0 (state at:(from))) (forEach Site (sites (values Remembered "Throws")) (if (or (is Empty (trackSite Move steps:(site))) (is Enemy (who at:(trackSite Move steps:(site))))) (move (from) (to (trackSite Move steps:(site)) (apply (forget Value "Throws" (site)))))))) (forEach Site (sites (values Remembered "Throws")) (if (= 1 (site)) (if (or (is Empty (trackSite Move steps:(site))) (is Enemy (who at:(trackSite Move steps:(site))))) (move (from) (to (trackSite Move steps:(site)) (apply (forget Value "Throws" (site)))) (then (set State at:(last To) 0))))))))) (map "Throw" { (pair 0 6) (pair 1 1) (pair 2 2) (pair 3 3) (pair 4 4) }) }) (rules (start { (place "Marker1" (sites {0 1 2 3 4}) state:1) (place "Marker2" (sites {29 30 31 32 33}) state:1) }) phases:{ (phase "GetMoves" (play (do (roll) next:(move Pass (then (remember Value "Throws" (mapEntry "Throw" (count Pips))))) (then (moveAgain)))) (nextPhase (not (is In (mapEntry "Throw" (count Pips)) (sites {1 4 6}))) "Movement")) (phase "Movement" (play (if (can Move (forEach Piece)) (forEach Piece (then (if (!= 0 (count Sites in:(sites (values Remembered "Throws")))) (moveAgain)))) (move Pass (then (forget Value "Throws" All))))) (nextPhase (not (!= 0 (count Sites in:(sites (values Remembered "Throws"))))) "GetMoves")) } (end (if (no Pieces Next) (result Next Loss))))
)
46
(game "Faraday" (players 2) (equipment { (board (tri Hexagon 6) use:Vertex) (piece "Ball" P1) (piece "Ball" P2) }) (rules (meta (no Repeat Positional)) (start (set Score Each 0)) phases:{ (phase "Pie" (play (if (is Mover P1) (or { (move Add (piece (mover)) (to (sites Empty))) (move Add (piece (next)) (to (sites Empty))) (move Pass) } (then (if (< 0 (counter)) (set NextPlayer (player (next))) (moveAgain)))) (or (move Propose "Accept Pie Offer and Move" (then (set NextPlayer (player (mover))))) (move Propose "Swap Pieces" (then (do (forEach Site (sites Occupied by:Mover) (remember Value (site))) next:(forEach Site (sites Occupied by:Next) (and (remove (site)) (add (piece (mover)) (to (site))))) (then (forEach Value (values Remembered) (and (remove (value)) (add (piece (next)) (to (value)))) (then (and (forget Value All) (set NextPlayer (player (next)))))))))))) (nextPhase (or (is Proposed "Swap Pieces") (is Proposed "Accept Pie Offer and Move")) "Placement")) (phase "Placement" (play (move Add (piece (mover)) (to (sites Empty) if:(or (<= 3 (count Pieces Next in:(sites Around (to) Orthogonal))) (< 0 (- (count Pieces Next in:(sites Around (to) Orthogonal)) (count Pieces Mover in:(sites Around (to) Orthogonal)))))) (then (and { (set Score Mover (max (sizes Group Orthogonal Mover))) (set Var "Last2Move" (mover)) (if (not (no Moves Mover)) (moveAgain)) }))))) (end (if (all Passed) { (if (!= (score Mover) (score Next)) (byScore)) (if (and (= (score Mover) (score Next)) (= (var "Last2Move") (mover)))

(result Mover Loss)) (if (and (= (score Mover) (score Next)) (!= (var "Last2Move") (mover))) (result Mover Loss)) } (byScore)))) })
)
3
Name: count, Length: 1373, dtype: int64
In [22]:
```
# End of high cardinality features
# Drop GameRulesetName, EnglishRules, LudRules
train.drop(["GameRulesetName", "EnglishRules", "LudRules"], axis=1, inplace=True)
```
4- Redundant Features
4.1- Constant Features
In [23]:
```
# Columns that have constant value
constant_columns = train.columns[train.nunique() == 1]
len(constant_columns)
```
Out[23]:
198
In [24]:
```
# Drop columns that have constant value
train.drop(columns=constant_columns, inplace=True)
```
4.2- Features with Same Values
In [25]:
```
# Features with same values
duplicated_columns = []
for i in range(len(train.columns)):
    v = train.iloc[:, i].values
    for j in range(i + 1, len(train.columns)):
        if np.array_equal(v, train.iloc[:, j].values):
            duplicated_columns.append(train.columns[i])
            break

len(duplicated_columns)
```
Out[25]:
33
In [26]:
```
# Drop duplicated
train.drop(columns=duplicated_columns, inplace=True)
```
4.3- Fully Correlated - Fully Uncorrelated Features
In [ ]:
```
# Fully correlated or fully uncorrelated features
# I already implemented correlation in CorrelationEliminator.py please let me use .corr() method
numerical_features = train.select_dtypes(include=['number'])
correlation_matrix = numerical_features.corr().abs()
upper = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape), k=1).astype(bool))
fully_corr_features = [column for column in upper.columns if any(upper[column] >= 1)]
len(fully_corr_features)
```
Out[ ]:
5
In [28]:
```
# Drop fully correlated features
train.drop(columns=fully_corr_features, inplace=True)
```
End of the Understanding Assignment and Dataset
In [33]:

train.shape
Out[33]:
(186588, 562)
In [38]:
filtered_column_names = list(train.columns)
In [37]:
 # Filtered features in one line
["Stochastic", "AsymmetricPiecesType", "Team", "Shape", "SquareShape", "HexShape", "TriangleShape", "DiamondShape", "RectangleShape", "StarShape", "RegularShape", "PolygonShape", "Tiling", "SquareTiling", "HexTiling", "TriangleTiling", "SemiRegularTiling", "MorrisTiling", "CircleTiling", "ConcentricTiling", "SpiralTiling", "AlquerqueTiling", "MancalaStores", "MancalaTwoRows", "MancalaThreeRows", "MancalaFourRows", "MancalaSixRows", "MancalaCircular", "AlquerqueBoard", "AlquerqueBoardWithOneTriangle", "AlquerqueBoardWithTwoTriangles", "AlquerqueBoardWithFourTriangles", "AlquerqueBoardWithEightTriangles", "ThreeMensMorrisBoard", "ThreeMensMorrisBoardWithTwoTriangles", "NineMensMorrisBoard", "StarBoard", "CrossBoard", "KintsBoard", "PachisiBoard", "FortyStonesWithFourGapsBoard", "Track", "TrackLoop", "TrackOwned", "Region", "Boardless", "Vertex", "Cell", "Edge", "NumPlayableSitesOnBoard", "NumColumns", "NumRows", "NumCorners", "NumDirections", "NumOrthogonalDirections", "NumDiagonalDirections", "NumAdjacentDirections", "NumOffDiagonalDirections", "NumInnerSites", "NumLayers", "NumEdges", "NumCells", "NumVertices", "NumPerimeterSites", "NumTopSites", "NumBottomSites", "NumRightSites", "NumLeftSites", "NumCentreSites", "NumConvexCorners", "NumConcaveCorners", "NumPhasesBoard", "Hand", "NumContainers", "NumPlayableSites", "Piece", "PieceValue", "PieceDirection", "DiceD2", "DiceD4", "DiceD6", "LargePiece", "Tile", "NumComponentsType", "NumComponentsTypePerPlayer", "NumDice", "Meta", "SwapOption", "Repetition", "TurnKo", "PositionalSuperko", "Start", "PiecesPlacedOnBoard", "PiecesPlacedOutsideBoard", "InitialRandomPlacement", "InitialScore", "InitialCost", "NumStartComponentsBoard", "NumStartComponentsHand", "NumStartComponents", "Moves", "MovesDecision", "NoSiteMoves", "VoteDecision", "SwapPlayersDecision", "SwapPlayersDecisionFrequency", "PassDecision", "PassDecisionFrequency", "ProposeDecision", "ProposeDecisionFrequency", "SingleSiteMoves", "AddDecision", "AddDecisionFrequency", "PromotionDecision", "PromotionDecisionFrequency", "RemoveDecision", "RemoveDecisionFrequency", "RotationDecision", "TwoSitesMoves", "StepDecision", "StepDecisionFrequency", "StepDecisionToEmpty", "StepDecisionToEmptyFrequency", "StepDecisionToFriend", "StepDecisionToFriendFrequency", "StepDecisionToEnemy", "StepDecisionToEnemyFrequency", "SlideDecision", "SlideDecisionFrequency", "SlideDecisionToEmpty", "SlideDecisionToEmptyFrequency", "SlideDecisionToEnemy", "SlideDecisionToEnemyFrequency", "SlideDecisionToFriend", "SlideDecisionToFriendFrequency", "LeapDecision", "LeapDecisionFrequency", "LeapDecisionToEmpty", "LeapDecisionToEmptyFrequency", "LeapDecisionToEnemy", "LeapDecisionToEnemyFrequency", "HopDecision", "HopDecisionFrequency", "HopDecisionMoreThanOne", "HopDecisionMoreThanOneFrequency", "HopDecisionEnemyToEmpty", "HopDecisionEnemyToEmptyFrequency", "HopDecisionFriendToEmpty", "HopDecisionFriendToEmptyFrequency", "HopDecisionFriendToFriendFrequency", "HopDecisionEnemyToEnemy", "HopDecisionEnemyToEnemyFrequency", "HopDecisionFriendToEnemy", "HopDecisionFriendToEnemyFrequency", "FromToDecision", "FromToDecisionFrequency", "FromToDecisionWithinBoardFrequency", "FromToDecisionBetweenContainersFrequency", "FromToDecisionEmpty", "FromToDecisionEmptyFrequency", "FromToDecisionEnemy", "FromToDecisionEnemyFrequency", "FromToDecisionFriend", "FromToDecisionFriendFrequency", "SwapPiecesDecision", "SwapPiecesDecisionFrequency", "ShootDecision", "MovesNonDecision", "MovesEffects",

"VoteEffect", "SwapPlayersEffect", "PassEffect", "Roll", "RollFrequency", "ProposeEffect", "ProposeEffectFrequency", "AddEffect", "AddEffectFrequency", "SowFrequency", "SowWithEffect", "SowCapture", "SowCaptureFrequency", "SowRemove", "SowRemoveFrequency", "SowBacktracking", "SowBacktrackingFrequency", "SowSkip", "SowOriginFirst", "SowCW", "SowCCW", "PromotionEffect", "PromotionEffectFrequency", "RemoveEffect", "RemoveEffectFrequency", "PushEffect", "PushEffectFrequency", "Flip", "FlipFrequency", "SetMove", "SetNextPlayer", "SetNextPlayerFrequency", "MoveAgain", "MoveAgainFrequency", "SetValue", "SetValueFrequency", "SetCount", "SetCountFrequency", "SetRotation", "StepEffect", "SlideEffect", "LeapEffect", "HopEffect", "FromToEffect", "MovesOperators", "Priority", "ByDieMove", "MaxMovesInTurn", "MaxDistance", "Capture", "ReplacementCapture", "ReplacementCaptureFrequency", "HopCapture", "HopCaptureFrequency", "HopCaptureMoreThanOne", "HopCaptureMoreThanOneFrequency", "DirectionCapture", "DirectionCaptureFrequency", "EncloseCapture", "EncloseCaptureFrequency", "CustodialCapture", "CustodialCaptureFrequency", "InterveneCapture", "InterveneCaptureFrequency", "SurroundCapture", "SurroundCaptureFrequency", "CaptureSequence", "CaptureSequenceFrequency", "Conditions", "SpaceConditions", "Line", "Connection", "Group", "Contains", "Pattern", "Fill", "Distance", "MoveConditions", "NoMoves", "NoMovesMover", "NoMovesNext", "CanMove", "CanNotMove", "PieceConditions", "NoPiece", "NoPieceMover", "NoPieceNext", "NoTargetPiece", "Threat", "IsEmpty", "IsEnemy", "IsFriend", "IsPieceAt", "LineOfSight", "CountPiecesComparison", "CountPiecesMoverComparison", "CountPiecesNextComparison", "ProgressCheck", "Directions", "AbsoluteDirections", "AllDirections", "AdjacentDirection", "OrthogonalDirection", "DiagonalDirection", "RotationalDirection", "SameLayerDirection", "RelativeDirections", "ForwardDirection", "BackwardDirection", "ForwardsDirection", "BackwardsDirection", "LeftwardDirection", "LeftwardsDirection", "ForwardRightDirection", "BackwardRightDirection", "SameDirection", "OppositeDirection", "Phase", "NumPlayPhase", "Scoring", "PieceCount", "SumDice", "SpaceEnd", "LineEnd", "LineEndFrequency", "LineWin", "LineWinFrequency", "LineLoss", "LineLossFrequency", "LineDraw", "ConnectionEnd", "ConnectionEndFrequency", "ConnectionWin", "ConnectionWinFrequency", "ConnectionLoss", "ConnectionLossFrequency", "GroupEnd", "GroupEndFrequency", "GroupWin", "GroupWinFrequency", "GroupLoss", "GroupDraw", "LoopEnd", "LoopWin", "LoopWinFrequency", "PatternWin", "PatternWinFrequency", "PathExtentLoss", "TerritoryWin", "TerritoryWinFrequency", "CaptureEnd", "Checkmate", "CheckmateFrequency", "CheckmateWin", "CheckmateWinFrequency", "NoTargetPieceEnd", "NoTargetPieceEndFrequency", "NoTargetPieceWin", "NoTargetPieceWinFrequency", "EliminatePiecesEnd", "EliminatePiecesEndFrequency", "EliminatePiecesWin", "EliminatePiecesWinFrequency", "EliminatePiecesLoss", "EliminatePiecesLossFrequency", "EliminatePiecesDraw", "EliminatePiecesDrawFrequency", "RaceEnd", "NoOwnPiecesEnd", "NoOwnPiecesEndFrequency", "NoOwnPiecesWin", "NoOwnPiecesWinFrequency", "NoOwnPiecesLoss", "NoOwnPiecesLossFrequency", "FillEnd", "FillEndFrequency", "FillWin", "FillWinFrequency", "ReachEnd", "ReachEndFrequency", "ReachWin", "ReachWinFrequency", "ReachLoss", "ReachLossFrequency", "ReachDraw", "ReachDrawFrequency", "ScoringEnd", "ScoringEndFrequency", "ScoringWin", "ScoringWinFrequency", "ScoringLoss", "ScoringLossFrequency", "ScoringDraw", "NoMovesEnd", "NoMovesEndFrequency", "NoMovesWin", "NoMovesWinFrequency", "NoMovesLoss", "NoMovesLossFrequency", "NoMovesDraw", "NoMovesDrawFrequency", "NoProgressEnd", "NoProgressDraw", "NoProgressDrawFrequency", "Draw", "DrawFrequency", "Misere", "DurationActions", "DurationMoves", "DurationTurns", "DurationTurnsStdDev", "DurationTurnsNotTimeouts", "DecisionMoves", "GameTreeComplexity", "StateTreeComplexity", "BoardCoverageDefault", "BoardCoverageFull", "BoardCoverageUsed", "AdvantageP1", "Balance", "Completion", "Drawishness", "Timeouts", "OutcomeUniformity", "BoardSitesOccupiedAverage", "BoardSitesOccupiedMedian", "BoardSitesOccupiedMaximum", "BoardSitesOccupiedVariance", "BoardSitesOccupiedChangeAverage", "BoardSitesOccupiedChangeSign", "BoardSitesOccupiedChangeLineBestFit", "BoardSitesOccupiedChangeNumTimes", "BoardSitesOccupiedMaxIncrease", "BoardSitesOccupiedMaxDecrease", "BranchingFactorAverage", "BranchingFactorMedian", "BranchingFactorMaximum", "BranchingFactorVariance", "BranchingFactorChangeAverage", "BranchingFactorChangeSign", "BranchingFactorChangeLineBestFit", "BranchingFactorChangeNumTimesn", "BranchingFactorChangeMaxIncrease", "BranchingFactorChangeMaxDecrease", "DecisionFactorAverage", "DecisionFactorMedian", "DecisionFactorMaximum", "DecisionFactorVariance", "DecisionFactorChangeAverage", "DecisionFactorChangeSign", "DecisionFactorChangeLineBestFit", "DecisionFactorChangeNumTimes", "DecisionFactorMaxIncrease", "DecisionFactorMaxDecrease", "MoveDistanceAverage", "MoveDistanceMedian", "MoveDistanceMaximum", "MoveDistanceVariance", "MoveDistanceChangeAverage", "MoveDistanceChangeSign", "MoveDistanceChangeLineBestFit", "MoveDistanceChangeNumTimes", "MoveDistanceMaxIncrease", "MoveDistanceMaxDecrease", "PieceNumberAverage", "PieceNumberMedian", "PieceNumberMaximum", "PieceNumberVariance", "PieceNumberChangeAverage", "PieceNumberChangeSign", "PieceNumberChangeLineBestFit", "PieceNumberChangeNumTimes", "PieceNumberMaxIncrease", "PieceNumberMaxDecrease", "ScoreDifferenceAverage", "ScoreDifferenceMedian", "ScoreDifferenceMaximum", "ScoreDifferenceVariance", "ScoreDifferenceChangeAverage", "ScoreDifferenceChangeSign", "ScoreDifferenceChangeLineBestFit", "ScoreDifferenceMaxIncrease", "ScoreDifferenceMaxDecrease", "Math", "Arithmetic", "Operations", "Addition", "Subtraction", "Multiplication", "Division", "Modulo", "Absolute", "Exponentiation", "Minimum", "Maximum", "Comparison", "Equal", "NotEqual", "LesserThan", "LesserThanOrEqual", "GreaterThan", "GreaterThanOrEqual", "Parity", "Even", "Odd", "Logic", "Conjunction", "Disjunction", "Negation", "Set", "Union", "Intersection", "Complement", "Algorithmics", "ConditionalStatement", "ControlFlowStatement", "Visual", "Style", "BoardStyle", "GraphStyle", "ChessStyle", "GoStyle", "MancalaStyle", "PenAndPaperStyle", "ShibumiStyle", "BackgammonStyle", "JanggiStyle", "XiangqiStyle", "ShogiStyle", "TableStyle", "SurakartaStyle", "TaflStyle", "NoBoard", "ComponentStyle", "AnimalComponent", "ChessComponent", "KingComponent", "QueenComponent", "KnightComponent", "RookComponent", "BishopComponent", "PawnComponent", "FairyChessComponent", "PloyComponent", "ShogiComponent", "XiangqiComponent", "StrategoComponent", "JanggiComponent", "CheckersComponent", "BallComponent", "TaflComponent", "DiscComponent", "MarkerComponent", "StackType", "Stack", "Symbols", "ShowPieceValue", "ShowPieceState", "Implementation", "State", "StackState", "PieceState", "SiteState", "SetSiteState", "VisitedSites", "Variable", "SetVar", "RememberValues", "ForgetValues", "SetPending", "InternalCounter", "SetInternalCounter", "PlayerValue", "Efficiency", "CopyContext", "Then", "ForEachPiece", "DoLudeme", "Trigger", "PlayoutsPerSecond", "MovesPerSecond", "num_wins_agent1", "num_draws_agent1", "num_losses_agent1", "utility_agent1", "p1_selection", "p1_exploration", "p1_playout", "p1_bounds", "p2_selection", "p2_exploration", "p2_playout", "p2_bounds"]

Appendix C:

Correlation Check With Features and Target
1- Initial Preprocessing
In [1]:
```
import numpy as np
import pandas as pd

import sys
import os
```

```
sys.path.append(os.path.abspath('../'))

from Models.LinearRegression import LinearRegression
from Utils.Preprocessor import Preprocessor
from Utils.Utils import root_mean_squared_error, train_test_split, initial_preprocessing
```
In [2]:
```
# Read the data
train = pd.read_csv('../Data/train.csv', index_col='Id')
```
In [3]:
```
# Remove unnecessary features based on exploratory data analysis part 1.
train = initial_preprocessing(train)
```
In [4]:
```
X = train.drop(columns=["num_wins_agent1", "num_draws_agent1", "num_losses_agent1", "utility_agent1"], axis=1)
y = train["utility_agent1"]
y1 = train["num_wins_agent1"]
y2 = train["num_draws_agent1"]
y3 = train["num_losses_agent1"]
```
In [5]:
```
# Preprocess the data
preprocessor = Preprocessor(normalize=True, one_hot_encode=True)

X = preprocessor.fit_transform(X)

# Convert back to pandas dataframe
X = pd.DataFrame(X, columns=preprocessor.get_column_names())
```
2- Correlation Check with Target

2.1- utility_agent1

In [7]:
```
def calculate_feature_correlations(X, y):
    # Drop categorical columns
    X = X.drop(columns=X.select_dtypes(include=['category']).columns, axis=1)

    # Store column names / for converting back to DataFrame
    colnames = X.columns

    # Convert to numpy arrays
    X = X.values
    y = y.values


    # corr calculation
    feature_means = np.mean(X, axis=0)
    target_mean = np.mean(y)
    feature_std = np.sqrt(np.sum((X - feature_means) ** 2, axis=0))
    target_std = np.sqrt(np.sum((y - target_mean) ** 2))

    # Reshape y for broadcasting
    y_reshaped = y.reshape(-1, 1)

    numerator = np.sum((X - feature_means) * (y_reshaped - target_mean), axis=0)
    denominator = feature_std * target_std
    denominator[denominator == 0] = 1000000  # Substitute large value for zero denominator to effectively make correlation zero if denominator is zero
    feature_correlations = numerator / denominator
```

```
    # Create a DataFrame with results
    correlation_df = pd.DataFrame({
        "Feature": colnames,
        "Correlation": feature_correlations
    }).sort_values(by="Correlation", ascending=False)

    return correlation_df
```
In [8]:
```
correlation_utility_agent1 = calculate_feature_correlations(X, y)
```
In [9]:
```
correlation_utility_agent1.head(10)
```
Out[9]:

	Feature	Correlation
387	AdvantageP1	0.442847
93	PiecesPlacedOutsideBoard	0.089204
286	Phase	0.079615
550	p1_exploration	0.076133
72	Hand	0.075179
73	NumContainers	0.063279
287	NumPlayPhase	0.060330
554	p1_selection_UCB1Tuned	0.059548
154	FromToDecision	0.059189
33	ThreeMensMorrisBoard	0.056606

In [10]:
```
correlation_utility_agent1.tail(10)
```
Out[10]:

	Feature	Correlation
519	TaflComponent	-0.049401
12	Tiling	-0.050846
11	PolygonShape	-0.051112
526	ShowPieceState	-0.052241
555	p1_playout_NST	-0.052393
475	Conjunction	-0.052703
92	PiecesPlacedOnBoard	-0.054233

	Feature	Correlation
3	Shape	-0.054465
560	p2_selection_UCB1Tuned	-0.058114
551	p2_exploration	-0.076252

2.1- num_wins_agent1

In [11]:

```
correlation_num_wins_agent1 = calculate_feature_correlations(X, y1)
```

In [12]:

```
correlation_num_wins_agent1.head(10)
```

Out[12]:

	Feature	Correlation
387	AdvantageP1	0.389072
389	Completion	0.314977
110	SingleSiteMoves	0.110007
115	RemoveDecision	0.080307
116	RemoveDecisionFrequency	0.072110
111	AddDecision	0.069312
1	AsymmetricPiecesType	0.068292
112	AddDecisionFrequency	0.065406
52	NumCorners	0.062774
550	p1_exploration	0.060513

In [13]:

```
correlation_num_wins_agent1.tail(10)
```

Out[13]:

	Feature	Correlation
388	Balance	-0.124447
99	NumStartComponents	-0.130388
374	DrawFrequency	-0.134934
376	DurationActions	-0.164809
392	OutcomeUniformity	-0.195809
382	GameTreeComplexity	-0.226140

	Feature	Correlation
377	DurationMoves	-0.243908
378	DurationTurns	-0.252856
391	Timeouts	-0.282967
390	Drawishness	-0.312753

2.2- num_draws_agent1

In [14]:

```
correlation_num_draws_agent1 = calculate_feature_correlations(X, y2)
```

In [15]:

```
correlation_num_draws_agent1.head(10)
```

Out[15]:

	Feature	Correlation
390	Drawishness	0.677952
391	Timeouts	0.614710
378	DurationTurns	0.555967
377	DurationMoves	0.532691
382	GameTreeComplexity	0.486117
392	OutcomeUniformity	0.462496
376	DurationActions	0.360513
374	DrawFrequency	0.314832
99	NumStartComponents	0.260798
388	Balance	0.225678

In [16]:

```
correlation_num_draws_agent1.tail(10)
```

Out[16]:

	Feature	Correlation
476	Disjunction	-0.109217
241	Group	-0.109456
14	HexTiling	-0.109519
402	BoardSitesOccupiedMaxDecrease	-0.112180
115	RemoveDecision	-0.122399

	Feature	Correlation
381	DecisionMoves	-0.128625
112	AddDecisionFrequency	-0.147525
111	AddDecision	-0.157628
110	SingleSiteMoves	-0.214452
389	Completion	-0.676826

2.3- num_losses_agent1

In [17]:

```
correlation_num_losses_agent1 = calculate_feature_correlations(X, y3)
```

In [18]:

```
correlation_num_losses_agent1.head(10)
```

Out[18]:

	Feature	Correlation
389	Completion	0.273189
110	SingleSiteMoves	0.078352
193	RemoveEffectFrequency	0.073616
12	Tiling	0.071929
11	PolygonShape	0.071534
3	Shape	0.071314
111	AddDecision	0.066895
241	Group	0.064701
10	RegularShape	0.063672
547	Trigger	0.063236

In [19]:

```
correlation_num_losses_agent1.tail(10)
```

Out[19]:

	Feature	Correlation
286	Phase	-0.117593
374	DrawFrequency	-0.139882
376	DurationActions	-0.145386
382	GameTreeComplexity	-0.197588

	Feature	Correlation
392	OutcomeUniformity	-0.209482
377	DurationMoves	-0.217724
378	DurationTurns	-0.230183
391	Timeouts	-0.250679
390	Drawishness	-0.276366
387	AdvantageP1	-0.395224

Appendix D:

Mutual Information Check With Features and Target

1- Initial Preprocessing

In [1]:

```
import numpy as np
import pandas as pd

import sys
import os
sys.path.append(os.path.abspath('../'))

from Models.LinearRegression import LinearRegression
from Utils.Preprocessor import Preprocessor
from Utils.Utils import root_mean_squared_error, train_test_split, initial_preprocessing
```

In [2]:

```
# Read the data
train = pd.read_csv('../Data/train.csv', index_col='Id')
```

In [3]:

```
# Remove unnecessary features based on exploratory data analysis part 1.
train = initial_preprocessing(train)
```

In [4]:

```
X = train.drop(columns=["num_wins_agent1", "num_draws_agent1", "num_losses_agent1",
"utility_agent1"], axis=1)
y = train["utility_agent1"]
y1 = train["num_wins_agent1"]
y2 = train["num_draws_agent1"]
y3 = train["num_losses_agent1"]
```

In [5]:

```
# Preprocess the data
preprocessor = Preprocessor(normalize=True, one_hot_encode=True)

X = preprocessor.fit_transform(X)

# Convert back to pandas dataframe
X = pd.DataFrame(X, columns=preprocessor.get_column_names())
```

2- Mutual Information Check with Target

2.1- utility_agent1

In [6]:

```
def calculate_mutual_information(X, y):
    # taken the formula from internet dont know if works.
```

```
    from collections import Counter

    def entropy(values):
        """Calculate the entropy of a dataset."""
        total = len(values)
        counts = Counter(values)
        probabilities = np.array([count / total for count in counts.values()])
        return -np.sum(probabilities * np.log2(probabilities + 1e-9))

    def conditional_entropy(feature, target):
        """Calculate the conditional entropy of target given feature."""
        total = len(feature)
        unique_values = np.unique(feature)
        cond_entropy = 0
        for value in unique_values:
            indices = np.where(feature == value)[0]
            subset = target[indices]
            cond_entropy += len(subset) / total * entropy(subset)
        return cond_entropy

    # Drop categorical columns
    X = X.drop(columns=X.select_dtypes(include=['category']).columns, axis=1)

    # Store column names for later data frame creation
    colnames = X.columns

    # Convert to numpy arrays
    X = X.values
    y = y.values if isinstance(y, pd.Series) else y

    # Calculate mutual information for each feature
    mutual_info = []
    for i in range(X.shape[1]):
        feature = X[:, i]
        feature_entropy = entropy(feature)
        target_entropy = entropy(y)
        cond_entropy = conditional_entropy(feature, y)
        mi = target_entropy - cond_entropy
        mutual_info.append(mi)

    # Create a DataFrame with results
    mutual_info_df = pd.DataFrame({
        "Feature": colnames,
        "Mutual Information": mutual_info
    }).sort_values(by="Mutual Information", ascending=False)

    return mutual_info_df
```

In [7]:

```
mutual_info_utility_agent1 = calculate_mutual_information(X, y)
```

In [8]:

```
mutual_info_utility_agent1.head(10)
```

Out[8]:

	Feature	Mutual Information
549	MovesPerSecond	1.283681
548	PlayoutsPerSecond	1.280695
376	DurationActions	1.273614
377	DurationMoves	1.263127
382	GameTreeComplexity	1.262383
378	DurationTurns	1.252940
380	DurationTurnsNotTimeouts	1.246248
379	DurationTurnsStdDev	1.182272
406	BranchingFactorVariance	1.117977
416	DecisionFactorVariance	1.091581

2.1- num_wins_agent1

In [9]:

```
mutual_info_num_wins_agent1 = calculate_mutual_information(X, y1)
```

In [10]:

```
mutual_info_num_wins_agent1.head(10)
```

Out[10]:

	Feature	Mutual Information
549	MovesPerSecond	0.986466
548	PlayoutsPerSecond	0.983382
376	DurationActions	0.978639
377	DurationMoves	0.970353
382	GameTreeComplexity	0.968839
378	DurationTurns	0.961348
380	DurationTurnsNotTimeouts	0.956689
379	DurationTurnsStdDev	0.908221
406	BranchingFactorVariance	0.850977
416	DecisionFactorVariance	0.836046

2.2- num_draws_agent1

In [11]:

```
mutual_info_num_draws_agent1 = calculate_mutual_information(X, y2)
```

In [12]:

mutual_info_num_draws_agent1.head(10)
Out[12]:

	Feature	Mutual Information
549	MovesPerSecond	1.041842
548	PlayoutsPerSecond	1.037821
376	DurationActions	1.036927
382	GameTreeComplexity	1.028330
377	DurationMoves	1.027954
378	DurationTurns	1.019944
380	DurationTurnsNotTimeouts	1.010021
379	DurationTurnsStdDev	0.963085
406	BranchingFactorVariance	0.914733
416	DecisionFactorVariance	0.906778

2.3- num_losses_agent1
In [13]:
mutual_info_num_lossess_agent1 = calculate_mutual_information(X, y3)
In [14]:
mutual_info_num_lossess_agent1.head(10)
Out[14]:

	Feature	Mutual Information
549	MovesPerSecond	0.982406
548	PlayoutsPerSecond	0.979751
376	DurationActions	0.973548
377	DurationMoves	0.967428
382	GameTreeComplexity	0.964320
378	DurationTurns	0.962127
380	DurationTurnsNotTimeouts	0.957681
379	DurationTurnsStdDev	0.912563
406	BranchingFactorVariance	0.843829
416	DecisionFactorVariance	0.826475

Appendix E: Model Interface
from abc import ABC, abstractmethod

```
import numpy as np

class Model(ABC):
    def __init__(self):
        """
        Initialize the Model.
        I hate python interfaces.
        """

    @abstractmethod
    def fit(self, X, y):
        """
        Method for fitting the model to data.
        Must be implemented by all subclasses.
        """
        pass

    @abstractmethod
    def predict(self, X):
        """
        Method for predicting values.
        Must be implemented by all subclasses.
        """
        pass
```

Appendix F: Linear Regression Model

```
import numpy as np
from .Model import Model

class LinearRegression(Model):
    def __init__(self, fit_method='ols', loss_function="rmse", l1=0, l2=0, learning_rate=0.01,
epochs=1000, min_step_size=0.001, gradient_descent='batch', batch_size=32):
        """
        Initialize the LinearRegression model with a specified fitting method.

        Parameters:
        - fit_method: The fitting method to use: "ols" for Ordinary Least Squares, "gd" for
Gradient Descent.
        - learning_rate: Learning rate for Gradient Descent.
        - loss_function: Loss function to use. rmse for Root Mean Squared Error. Only Root
Mean Squared Error is supported for now.
        - l1: L1 regularization parameter.
        - l2: L2 regularization parameter.
        - epochs: Number of epochs for Gradient Descent.
        - min_step_size: Minimum step size for Gradient Descent.
        - gradient_descent: Type of gradient_descent. Possible values: "batch", "stochastic",
"mini-batch".
        - batch_size: Size of batch for mini-bactch gradient descent.

        Notes:
        - You cant use l1 regularization with ols because there is no closed form solution.
        """

        super().__init__()
```

```
        # general parameters
        self.fit_method = fit_method
        self.loss_function = loss_function

        # regularization parameters
        self.l1 = l1
        self.l2 = l2

        # gradient descent parameters
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.min_step_size = min_step_size
        self.gradient_descent = gradient_descent
        self.batch_size = batch_size

        # initialize weights to none
        self.weights = None # W0 will be bias.

    def fit(self, X, y):
        """
        Fit the model to the data based on selected fit method.

        Parameters:
        - X: Input value array for training data. Should be numpy array with shape (n_samples,
n_features).
        - y: Target value array for training data. Should be numpy array with shape (n_samples, ).
        """

        # Add bias terms coefficent to the X for easier bias term handling.
        X = np.c_[np.ones((X.shape[0], 1)), X]

        if self.fit_method == 'ols':
            self._fit_ols(X, y)
        elif self.fit_method == 'gd':
            self._fit_gd(X, y)
        else:
            raise ValueError("fit_method should be either 'ols' or 'gd'")


    def predict(self, X):
        """
        Predict the target values for given inputs.

        Parameters:
        - X: Input value array for prediction. Should be numpy array with shape (n_samples,
n_features).

        Returns:
        - y: Predictions values for input array X. numpy array with shape (n_samples, )
        """

        if self.weights is None:
            raise ValueError("Model has not been fitted yet.")

        # Add bias terms coefficent to the X for prediction.
```

```
        X = np.c_[np.ones((X.shape[0], 1)), X]

        y = self._predict(X)
        return y

    def _calculate_gradient(self, X, y):
        """
        Calculate the gradient for the loss function for given X, y_true and y_pred values.

        Parameters:
        - X: Input value array for training data. Should be numpy array with shape (n_samples,
n_features).
        - y: Target value array for training data. Should be numpy array with shape (n_samples, ).
        """

        y_pred = self._predict(X)

        if self.loss_function == 'rmse':
            loss_gradient = - X.T @ (y - y_pred) / (X.shape[0] * np.sqrt(np.mean((y - y_pred) **
2))) + self.l1 * np.sign(self.weights) + 2 * self.l2 * self.weights - self.l1 *
np.sign(self.weights[0]) - 2 * self.l2 * self.weights[0]
        else:
            raise ValueError("loss_function should be rmse.")

        return loss_gradient

    def _fit_ols(self, X, y):
        """
        Fit the model to the data using ordinary least squares fit method by calculating weights
by given formula.

        Parameters:
        - X: Input value array for training data. Should be numpy array with shape (n_samples,
n_features).
        - y: Target value array for training data. Should be numpy array with shape (n_samples, ).
        """

        self.weights = np.linalg.pinv(X.T @ X + self.l2 * np.identity(X.shape[1])) @ X.T @ y

    def _fit_gd(self, X, y):
        if self.gradient_descent == 'batch':
            self._fit_gd_batch(X, y)
        elif self.gradient_descent == 'stochastic':
            self._fit_gd_stochastic(X, y)
        elif self.gradient_descent == 'mini-batch':
            self._fit_gd_mini_batch(X, y)
        else:
            raise ValueError("Incorrect gradient_descent value. Possible values: batch, stochastic,
mini-batch.")

    def _fit_gd_batch(self, X, y):
        """
        Fit the model to the data using batch gradient descent method by updating weights untill
convergence.
        Batch gradients use all the training data for updating weights at each step.
```

```
        Parameters:
        - X: Input value array for training data. Should be numpy array with shape (n_samples,
n_features).
        - y: Target value array for training data. Should be numpy array with shape (n_samples, ).
        """

        # Initialize weights
        self.weights = np.random.randn(X.shape[1], ) * 0.01
        self.weights[0] = 0 # Thats what they do in NN

        # Gradient Descent Loop
        for _ in range(self.epochs):
            gradient = self._calculate_gradient(X, y)
            self.weights = self.weights - self.learning_rate * gradient

    def _fit_gd_stochastic(self, X, y):
        """
        Fit the model to the data using batch gradient descent method by updating weights untill
convergence.
        Batch gradients use all the training data for updating weights at each step.

        Parameters:
        - X: Input value array for training data. Should be numpy array with shape (n_samples,
n_features).
        - y: Target value array for training data. Should be numpy array with shape (n_samples, ).
        """

        # Initialize weights
        self.weights = np.random.randn(X.shape[1], ) * 0.01
        self.weights[0] = 0 # Thats what they do in NN

        n = X.shape[0]
        current_index = 0
        for epoch in range(self.epochs):
            if epoch % n == 0:
                indices = np.arange(n)
                np.random.shuffle(indices)
                X = X[indices]
                y = y[indices]

            current_X, current_y = X[current_index : current_index + 1], y[current_index]
            current_index = (current_index + 1) % n
            gradient = self._calculate_gradient(current_X, current_y)
            self.weights = self.weights - self.learning_rate * gradient

    def _fit_gd_mini_batch(self, X, y):
        """
        Fit the model to the data using batch gradient descent method by updating weights untill
convergence.
        Batch gradients use all the training data for updating weights at each step.

        Parameters:
        - X: Input value array for training data. Should be numpy array with shape (n_samples,
n_features).
```

```
        - y: Target value array for training data. Should be numpy array with shape (n_samples, ).
        """

        # Initialize weights
        self.weights = np.random.randn(X.shape[1], ) * 0.01
        self.weights[0] = 0 # Thats what they do in NN

        n = X.shape[0]
        current_index = 0

        for epoch in range(self.epochs):
            if epoch % n == 0:
                indices = np.arange(n)
                np.random.shuffle(indices)
                X = X[indices]
                y = y[indices]

            current_X, current_y = X[current_index : min(current_index + self.batch_size, n)],
y[current_index : min(current_index + self.batch_size, n)]
            current_index = min(current_index + self.batch_size, n) % n
            gradient = self._calculate_gradient(current_X, current_y)
            self.weights = self.weights - self.learning_rate * gradient

    def _predict(self, X):
        """
        Helper method for gradient descent. Using self.predict add 1s for the biases.
        """
        return X @ self.weights
```

Appendix G: Decision Tree Regressor

```
import numpy as np
from Models.Model import Model

class DecisionTreeRegressor(Model):
    def __init__(self, max_depth=None, min_samples_split=2):
        """
        Initialize the DecisionTreeRegressor with minimal parameters.

        Parameters:
        max_depth: maximum depth of the tree. Integer or None. None is no bound.

        min_samples_split: int, optional (default=20) # statsquest
            The minimum number of samples required to split an internal node.
        """
        super().__init__()
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.tree = None

    def fit(self, X, y):
        """
        Fit the decision tree to the data.

        Parameters:
```

```
        - X: Input value array for training data. Should be numpy array with shape (n_samples,
n_features).
        - y: Target value array for training data. Should be numpy array with shape (n_samples, ).
        """
        pass

    def predict(self, X):
        """
        Predict the target values for given inputs.

        Parameters:
        - X: Input value array for prediction. Should be numpy array with shape (n_samples,
n_features).

        Returns:
        - y: Predictions values for input array X. numpy array with shape (n_samples, )
        """
        pass
```

Appendix H: Utils

```
import numpy as np
import pandas as pd


def extract_features(agent_column):
    """
    Getting the selection, exploration, playout and bounds from the agent columns
    Function to extract features based on the pattern provided
    """
    selection = agent_column.str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)',
expand=True)[0].astype('category')
    exploration = agent_column.str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)',
expand=True)[1].astype(float)
    playout = agent_column.str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)',
expand=True)[2].astype('category')
    bounds = agent_column.str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)',
expand=True)[3].astype('category')
    return selection, exploration, playout, bounds

def initial_preprocessing(df):
    """
    Apply initial preprocessing.
    Remove features that are unnecessary in df and returns it.
    Details are on ExploratoryDataAnalysis/1-UnderstandingAssignmentAndFeatures.ipynb
    """

    # Applying the function to extract features for agent1 and agent2
    df['p1_selection'], df['p1_exploration'], df['p1_playout'], df['p1_bounds'] =
extract_features(df['agent1'])
    df['p2_selection'], df['p2_exploration'], df['p2_playout'], df['p2_bounds'] =
extract_features(df['agent2'])

    df = df.drop(["agent1", "agent2"], axis=1)

    # Keep only columns that give new information.
```

Filter by the reduced columns obtained in the Exploratory Data Analysis - Understanding Assignment and Features

df = df[["Stochastic", "AsymmetricPiecesType", "Team", "Shape", "SquareShape", "HexShape", "TriangleShape", "DiamondShape", "RectangleShape", "StarShape", "RegularShape", "PolygonShape", "Tiling", "SquareTiling", "HexTiling", "TriangleTiling", "SemiRegularTiling", "MorrisTiling", "CircleTiling", "ConcentricTiling", "SpiralTiling", "AlquerqueTiling", "MancalaStores", "MancalaTwoRows", "MancalaThreeRows", "MancalaFourRows", "MancalaSixRows", "MancalaCircular", "AlquerqueBoard", "AlquerqueBoardWithOneTriangle", "AlquerqueBoardWithTwoTriangles", "AlquerqueBoardWithFourTriangles", "AlquerqueBoardWithEightTriangles", "ThreeMensMorrisBoard", "ThreeMensMorrisBoardWithTwoTriangles", "NineMensMorrisBoard", "StarBoard", "CrossBoard", "KintsBoard", "PachisiBoard", "FortyStonesWithFourGapsBoard", "Track", "TrackLoop", "TrackOwned", "Region", "Boardless", "Vertex", "Cell", "Edge", "NumPlayableSitesOnBoard", "NumColumns", "NumRows", "NumCorners", "NumDirections", "NumOrthogonalDirections", "NumDiagonalDirections", "NumAdjacentDirections", "NumOffDiagonalDirections", "NumInnerSites", "NumLayers", "NumEdges", "NumCells", "NumVertices", "NumPerimeterSites", "NumTopSites", "NumBottomSites", "NumRightSites", "NumLeftSites", "NumCentreSites", "NumConvexCorners", "NumConcaveCorners", "NumPhasesBoard", "Hand", "NumContainers", "NumPlayableSites", "Piece", "PieceValue", "PieceDirection", "DiceD2", "DiceD4", "DiceD6", "LargePiece", "Tile", "NumComponentsType", "NumComponentsTypePerPlayer", "NumDice", "Meta", "SwapOption", "Repetition", "TurnKo", "PositionalSuperko", "Start", "PiecesPlacedOnBoard", "PiecesPlacedOutsideBoard", "InitialRandomPlacement", "InitialScore", "InitialCost", "NumStartComponentsBoard", "NumStartComponentsHand", "NumStartComponents", "Moves", "MovesDecision", "NoSiteMoves", "VoteDecision", "SwapPlayersDecision", "SwapPlayersDecisionFrequency", "PassDecision", "PassDecisionFrequency", "ProposeDecision", "ProposeDecisionFrequency", "SingleSiteMoves", "AddDecision", "AddDecisionFrequency", "PromotionDecision", "PromotionDecisionFrequency", "RemoveDecision", "RemoveDecisionFrequency", "RotationDecision", "TwoSitesMoves", "StepDecision", "StepDecisionFrequency", "StepDecisionToEmpty", "StepDecisionToEmptyFrequency", "StepDecisionToFriend", "StepDecisionToFriendFrequency", "StepDecisionToEnemy", "StepDecisionToEnemyFrequency", "SlideDecision", "SlideDecisionFrequency", "SlideDecisionToEmpty", "SlideDecisionToEmptyFrequency", "SlideDecisionToEnemy", "SlideDecisionToEnemyFrequency", "SlideDecisionToFriend", "SlideDecisionToFriendFrequency", "LeapDecision", "LeapDecisionFrequency", "LeapDecisionToEmpty", "LeapDecisionToEmptyFrequency", "LeapDecisionToEnemy", "LeapDecisionToEnemyFrequency", "HopDecision", "HopDecisionFrequency", "HopDecisionMoreThanOne", "HopDecisionMoreThanOneFrequency", "HopDecisionEnemyToEmpty", "HopDecisionEnemyToEmptyFrequency", "HopDecisionFriendToEmpty", "HopDecisionFriendToEmptyFrequency", "HopDecisionFriendToFriendFrequency", "HopDecisionEnemyToEnemy", "HopDecisionEnemyToEnemyFrequency", "HopDecisionFriendToEnemy", "HopDecisionFriendToEnemyFrequency", "FromToDecision", "FromToDecisionFrequency", "FromToDecisionWithinBoardFrequency", "FromToDecisionBetweenContainersFrequency", "FromToDecisionEmpty", "FromToDecisionEmptyFrequency", "FromToDecisionEnemy", "FromToDecisionEnemyFrequency", "FromToDecisionFriend", "FromToDecisionFriendFrequency", "SwapPiecesDecision", "SwapPiecesDecisionFrequency", "ShootDecision", "MovesNonDecision", "MovesEffects", "VoteEffect", "SwapPlayersEffect", "PassEffect", "Roll", "RollFrequency", "ProposeEffect", "ProposeEffectFrequency", "AddEffect", "AddEffectFrequency", "SowFrequency", "SowWithEffect", "SowCapture", "SowCaptureFrequency", "SowRemove", "SowRemoveFrequency", "SowBacktracking", "SowBacktrackingFrequency", "SowSkip", "SowOriginFirst", "SowCW", "SowCCW", "PromotionEffect", "PromotionEffectFrequency", "RemoveEffect", "RemoveEffectFrequency", "PushEffect", "PushEffectFrequency", "Flip", "FlipFrequency", "SetMove", "SetNextPlayer", "SetNextPlayerFrequency", "MoveAgain", "MoveAgainFrequency", "SetValue", "SetValueFrequency", "SetCount", "SetCountFrequency", "SetRotation", "StepEffect", "SlideEffect", "LeapEffect", "HopEffect", "FromToEffect", "MovesOperators", "Priority", "ByDieMove", "MaxMovesInTurn", "MaxDistance", "Capture", "ReplacementCapture", "ReplacementCaptureFrequency", "HopCapture", "HopCaptureFrequency", "HopCaptureMoreThanOne", "HopCaptureMoreThanOneFrequency", "DirectionCapture", "DirectionCaptureFrequency", "EncloseCapture", "EncloseCaptureFrequency", "CustodialCapture", "CustodialCaptureFrequency", "InterveneCapture", "InterveneCaptureFrequency", "SurroundCapture", "SurroundCaptureFrequency", "CaptureSequence", "CaptureSequenceFrequency", "Conditions", "SpaceConditions", "Line", "Connection", "Group", "Contains", "Pattern", "Fill", "Distance", "MoveConditions", "NoMoves", "NoMovesMover", "NoMovesNext", "CanMove", "CanNotMove", "PieceConditions", "NoPiece", "NoPieceMover", "NoPieceNext", "NoTargetPiece", "Threat", "IsEmpty", "IsEnemy", "IsFriend", "IsPieceAt", "LineOfSight", "CountPiecesComparison", "CountPiecesMoverComparison", "CountPiecesNextComparison", "ProgressCheck", "Directions", "AbsoluteDirections", "AllDirections", "AdjacentDirection", "OrthogonalDirection", "DiagonalDirection", "RotationalDirection", "SameLayerDirection", "RelativeDirections", "ForwardDirection", "BackwardDirection", "ForwardsDirection", "BackwardsDirection", "LeftwardDirection", "LeftwardsDirection", "ForwardRightDirection", "BackwardRightDirection", "SameDirection", "OppositeDirection", "Phase", "NumPlayPhase", "Scoring", "PieceCount", "SumDice", "SpaceEnd", "LineEnd", "LineEndFrequency", "LineWin", "LineWinFrequency", "LineLoss", "LineLossFrequency", "LineDraw", "ConnectionEnd", "ConnectionEndFrequency", "ConnectionWin", "ConnectionWinFrequency", "ConnectionLoss", "ConnectionLossFrequency", "GroupEnd", "GroupEndFrequency", "GroupWin", "GroupWinFrequency", "GroupLoss", "GroupDraw", "LoopEnd", "LoopWin", "LoopWinFrequency", "PatternWin", "PatternWinFrequency", "PathExtentLoss", "TerritoryWin", "TerritoryWinFrequency", "CaptureEnd", "Checkmate", "CheckmateFrequency", "CheckmateWin", "CheckmateWinFrequency", "NoTargetPieceEnd", "NoTargetPieceEndFrequency", "NoTargetPieceWin", "NoTargetPieceWinFrequency", "EliminatePiecesEnd", "EliminatePiecesEndFrequency", "EliminatePiecesWin", "EliminatePiecesWinFrequency", "EliminatePiecesLoss", "EliminatePiecesLossFrequency", "EliminatePiecesDraw", "EliminatePiecesDrawFrequency", "RaceEnd", "NoOwnPiecesEnd", "NoOwnPiecesEndFrequency", "NoOwnPiecesWin", "NoOwnPiecesWinFrequency", "NoOwnPiecesLoss", "NoOwnPiecesLossFrequency", "FillEnd", "FillEndFrequency", "FillWin", "FillWinFrequency", "ReachEnd", "ReachEndFrequency", "ReachWin", "ReachWinFrequency", "ReachLoss", "ReachLossFrequency", "ReachDraw", "ReachDrawFrequency", "ScoringEnd", "ScoringEndFrequency", "ScoringWin", "ScoringWinFrequency", "ScoringLoss", "ScoringLossFrequency", "ScoringDraw", "NoMovesEnd", "NoMovesEndFrequency", "NoMovesWin", "NoMovesWinFrequency", "NoMovesLoss", "NoMovesLossFrequency", "NoMovesDraw", "NoMovesDrawFrequency", "NoProgressEnd", "NoProgressDraw", "NoProgressDrawFrequency", "Draw", "DrawFrequency", "Misere", "DurationActions", "DurationMoves", "DurationTurns", "DurationTurnsStdDev", "DurationTurnsNotTimeouts", "DecisionMoves", "GameTreeComplexity", "StateTreeComplexity", "BoardCoverageDefault", "BoardCoverageFull", "BoardCoverageUsed", "AdvantageP1", "Balance", "Completion", "Drawishness", "Timeouts", "OutcomeUniformity", "BoardSitesOccupiedAverage", "BoardSitesOccupiedMedian", "BoardSitesOccupiedMaximum", "BoardSitesOccupiedVariance", "BoardSitesOccupiedChangeAverage", "BoardSitesOccupiedChangeSign", "BoardSitesOccupiedChangeLineBestFit", "BoardSitesOccupiedChangeNumTimes", "BoardSitesOccupiedMaxIncrease", "BoardSitesOccupiedMaxDecrease", "BranchingFactorAverage", "BranchingFactorMedian", "BranchingFactorMaximum", "BranchingFactorVariance", "BranchingFactorChangeAverage",

```
"BranchingFactorChangeSign", "BranchingFactorChangeLineBestFit",
"BranchingFactorChangeNumTimesn", "BranchingFactorChangeMaxIncrease",
"BranchingFactorChangeMaxDecrease", "DecisionFactorAverage", "DecisionFactorMedian",
"DecisionFactorMaximum", "DecisionFactorVariance", "DecisionFactorChangeAverage",
"DecisionFactorChangeSign", "DecisionFactorChangeLineBestFit",
"DecisionFactorChangeNumTimes", "DecisionFactorMaxIncrease",
"DecisionFactorMaxDecrease", "MoveDistanceAverage", "MoveDistanceMedian",
"MoveDistanceMaximum", "MoveDistanceVariance", "MoveDistanceChangeAverage",
"MoveDistanceChangeSign", "MoveDistanceChangeLineBestFit",
"MoveDistanceChangeNumTimes", "MoveDistanceMaxIncrease",
"MoveDistanceMaxDecrease", "PieceNumberAverage", "PieceNumberMedian",
"PieceNumberMaximum", "PieceNumberVariance", "PieceNumberChangeAverage",
"PieceNumberChangeSign", "PieceNumberChangeLineBestFit",
"PieceNumberChangeNumTimes", "PieceNumberMaxIncrease",
"PieceNumberMaxDecrease", "ScoreDifferenceAverage", "ScoreDifferenceMedian",
"ScoreDifferenceMaximum", "ScoreDifferenceVariance", "ScoreDifferenceChangeAverage",
"ScoreDifferenceChangeSign", "ScoreDifferenceChangeLineBestFit",
"ScoreDifferenceMaxIncrease", "ScoreDifferenceMaxDecrease", "Math", "Arithmetic",
"Operations", "Addition", "Subtraction", "Multiplication", "Division", "Modulo", "Absolute",
"Exponentiation", "Minimum", "Maximum", "Comparison", "Equal", "NotEqual",
"LesserThan", "LesserThanOrEqual", "GreaterThan", "GreaterThanOrEqual", "Parity",
"Even", "Odd", "Logic", "Conjunction", "Disjunction", "Negation", "Set", "Union",
"Intersection", "Complement", "Algorithmics", "ConditionalStatement",
"ControlFlowStatement", "Visual", "Style", "BoardStyle", "GraphStyle", "ChessStyle",
"GoStyle", "MancalaStyle", "PenAndPaperStyle", "ShibumiStyle", "BackgammonStyle",
"JanggiStyle", "XiangqiStyle", "ShogiStyle", "TableStyle", "SurakartaStyle", "TaflStyle",
"NoBoard", "ComponentStyle", "AnimalComponent", "ChessComponent",
"KingComponent", "QueenComponent", "KnightComponent", "RookComponent",
"BishopComponent", "PawnComponent", "FairyChessComponent", "PloyComponent",
"ShogiComponent", "XiangqiComponent", "StrategoComponent", "JanggiComponent",
"CheckersComponent", "BallComponent", "TaflComponent", "DiscComponent",
"MarkerComponent", "StackType", "Stack", "Symbols", "ShowPieceValue",
"ShowPieceState", "Implementation", "State", "StackState", "PieceState", "SiteState",
"SetSiteState", "VisitedSites", "Variable", "SetVar", "RememberValues", "ForgetValues",
"SetPending", "InternalCounter", "SetInternalCounter", "PlayerValue", "Efficiency",
"CopyContext", "Then", "ForEachPiece", "DoLudeme", "Trigger", "PlayoutsPerSecond",
"MovesPerSecond", "num_wins_agent1", "num_draws_agent1", "num_losses_agent1",
"utility_agent1", "p1_selection", "p1_exploration", "p1_playout", "p1_bounds",
"p2_selection", "p2_exploration", "p2_playout", "p2_bounds"]]

    return df

def train_test_split(X, y, test_size=0.2, random_state=None):
    """
    Split the X and y into training and testing.
    """

    # Shuffle the data
    shuffled_indices = X.sample(frac=1, random_state=random_state).index
    X_shuffled = X.loc[shuffled_indices]
    y_shuffled = y.loc[shuffled_indices]

    # Calculate
    test_size_count = int(test_size * len(X))
```

```
    # Split data
    X_train = X_shuffled.iloc[:-test_size_count]
    X_test = X_shuffled.iloc[-test_size_count:]
    y_train = y_shuffled.iloc[:-test_size_count]
    y_test = y_shuffled.iloc[-test_size_count:]

    return X_train, X_test, y_train, y_test

def mean_squared_error(y_true, y_pred):
    """
    Calculate mean squared error.
    """
    return np.mean((y_true - y_pred) ** 2)

def root_mean_squared_error(y_true, y_pred):
    """
    Calculate root mean squared error.
    """
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

Appendix I: Preprocessor

```
import numpy as np
import pandas as pd

class Preprocessor:
    def __init__(self, normalize=False, standardize=False, one_hot_encode=False):
        """
        Initialize the Preprocessor. Takes pandas dataframe, normalizes and standardizes it.
Return numpy array.

        Parameters:
        - normalize: Normalize if true.
        - standardize: Standardize if true.
        - one_hot_endoce: One hot encode if true.
        """
        self.normalize = normalize
        self.standardize = standardize
        self.one_hot_encode = one_hot_encode

        self.feature_min = None
        self.feature_max = None
        self.feature_mean = None
        self.feature_std = None

        self.categorical_columns = None
        self.categories_per_column = {}

    def fit(self, df):
        """
        Fit the preprocessor to the data.
        Extracts categories and categories_per_column for one-hot encoding.
        Extracts feature_min, feature_max for normalization.
        Extracts feature_mean, feature_std for standardization

        Parameters:
```

```
        - df: A pandas DataFrame to normalize or standardize.
        """

        if self.one_hot_encode:
            self.categorical_columns = df.select_dtypes(include='category').columns.tolist()
            for col in self.categorical_columns:
                categories = df[col].cat.categories.tolist()
                self.categories_per_column[col] = categories

            # Perform one-hot encoding temporarily to add new columns for
normalization/standardization
            one_hot_encoded_df = df.copy()
            for col in self.categorical_columns:
                categories = self.categories_per_column[col]
                one_hot_encoded_df[col] = pd.Categorical(one_hot_encoded_df[col],
categories=categories)
                for category in categories[1:]:  # Dropping the first one for one-hot encoding
                    new_col = f"{col}_{category}"
                    one_hot_encoded_df[new_col] = (one_hot_encoded_df[col] ==
category).astype(int)
                one_hot_encoded_df.drop(columns=col, inplace=True)

            df_num = one_hot_encoded_df
        else:
            df_num = df

        if self.normalize:
            self.feature_min = df_num.min(axis=0)
            self.feature_max = df_num.max(axis=0)

        if self.standardize:
            if self.normalize:
                normalized_df = (df_num - self.feature_min) / (self.feature_max - self.feature_min)
                self.feature_mean = normalized_df.mean(axis=0)
                self.feature_std = normalized_df.std(axis=0)
            else:
                self.feature_mean = df_num.mean(axis=0)
                self.feature_std = df_num.std(axis=0)

    def transform(self, df):
        """
        Transform the data.

        Parameters:
        - df: A pandas DataFrame to be transformed.

        Returns:
        - A NumPy ndarray with the transformed data.
        """

        df_transformed = df.copy()

        if self.one_hot_encode:
            if self.categorical_columns is None or self.categories_per_column is None:
                raise ValueError("Not fitted yet.")
```

```
        for col in self.categorical_columns:
            categories = self.categories_per_column[col]
            df_transformed[col] = pd.Categorical(df_transformed[col], categories=categories)
            for category in categories[1:]: # Dropping the first one.
                new_col = f"{col}_{category}"
                df_transformed[new_col] = (df_transformed[col] == category).astype(int)
            df_transformed.drop(columns=col, inplace=True)

        df_num = df_transformed
        self.column_names = df_num.columns

        if self.normalize:
            if self.feature_min is None or self.feature_max is None:
                raise ValueError("Not fitted yet.")
            normalized = (df_num - self.feature_min) / (self.feature_max - self.feature_min)
        else:
            normalized = df_num

        if self.standardize:
            if self.feature_mean is None or self.feature_std is None:
                raise ValueError("Not fitted yet.")
            standardized = (normalized - self.feature_mean) / self.feature_std
            return standardized.to_numpy()

        return normalized.to_numpy()

    def fit_transform(self, df):
        """
        Fit and transform the data.

        Parameters:
        - df: A pandas DataFrame to be fitted and transformed.

        Returns:
        - A NumPy ndarray with the transformed data.
        """
        self.fit(df)
        return self.transform(df)

    def get_column_names(self):
        return self.column_names
```

Appendix J: Linear Regression Baseline Model

Linear Regression Baseline
Preprocess Data
In [1]:

```
import numpy as np
import pandas as pd

import sys
import os
sys.path.append(os.path.abspath('../'))

from Models.LinearRegression import LinearRegression
```

```
from Utils.Preprocessor import Preprocessor
from Utils.Utils import root_mean_squared_error, train_test_split, initial_preprocessing
```
In [2]:
```
# Read the data
train = pd.read_csv('../Data/train.csv', index_col='Id')
```
In [ ]:
```
# Remove unnecessary features based on exploratory data analysis part 1.
train = initial_preprocessing(train)
```
1- Linear Regression

In [4]:
```
X = train.drop(columns=["num_wins_agent1", "num_draws_agent1", "num_losses_agent1",
"utility_agent1"], axis=1)
y = train["utility_agent1"]
```
In [5]:
```
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)
```
In [6]:
```
preprocessor = Preprocessor(normalize=True, standardize=False, one_hot_encode=True)

X_train_p = preprocessor.fit_transform(X_train)
X_valid_p = preprocessor.transform(X_valid)
```
In [7]:
```
lr_model = LinearRegression(fit_method="ols", loss_function="rmse")
#lr_model = LinearRegression(fit_method="gd", loss_function="rmse", learning_rate=0.01,
epochs=10, min_step_size=0.001, gradient_descent='batch')

lr_model.fit(X_train_p, y_train)

train_pred = lr_model.predict(X_train_p)
test_pred = lr_model.predict(X_valid_p)

print("Linear Regression: ")
print("Train mean squared error: ", root_mean_squared_error(y_train, train_pred))
print("Validation mean squared error: ", root_mean_squared_error(y_valid, test_pred))
```
```
Linear Regression:
Train mean squared error:  0.5175135945021986
Validation mean squared error:  0.51911678407925
```
2- Lasso Regression

3- Ridge Regression

Appendix K: Feature Eliminator Interface
```
from abc import ABC, abstractmethod
import numpy as np

class FeatureEliminator(ABC):
    """
    FeatureEliminator interface.

    Does feature elimination based on supervised or unsupervised methods.
    """

    def __init__(self, X, y):
        """
        Initialize the FeatureEliminator.
        X is feature array.
```

```
        y is

        Parameterers:
        - X : Feature array. numpy ndarray.
        - y : Target array. numpy ndarray.
        """
        self.X = X
        self.y = y

        self.feature_indices = None
        self.feature_mask = None

        self.select_features()

    @abstractmethod
    def select_features(self):
        """
        Select features based on some supervised or unsupervised method.
        Extracts feature indices and feature_maskç
        """
        pass

    def get_feature_indices(self):
        """
        Get indices of features to keep.

        Return numpy ndarray consist of indices to keep.
        """
        return self.feature_indices

    def get_feature_mask(self):
        """
        Get feature mask. Feature mask consist of True, False values.
        True for keeping the index, False for dropping.

        Return numpy ndarray mask.
        """
        return self.feature_mask
```

Appendix L: Variance Eliminator

```
import numpy as np

from .FeatureEliminator import FeatureEliminator
class VarianceEliminator(FeatureEliminator):
    """
    Eliminates features with variance below a specified threshold.
    """

    def __init__(self, X, y, threshold=0.01):
        """
        Initialize the VarianceEliminator with a variance threshold.

        Parameters:
        - X : Feature array. numpy ndarray.
        - y : Target array. numpy ndarray.
```

```
        - threshold : Minimum variance required to retain a feature.
        """
        self.threshold = threshold
        super().__init__(X, y)

    def select_features(self):
        """
        Select features with variance above the specified threshold.
        """
        feature_means = np.mean(self.X, axis=0)
        feature_variances = np.sum((self.X - feature_means) ** 2, axis=0) / self.X.shape[0]

        self.feature_mask = feature_variances > self.threshold
        self.feature_indices = [i for i, is_selected in enumerate(self.feature_mask) if is_selected]
```

Appendix M: Correlation Eliminator

```
import numpy as np
from .FeatureEliminator import FeatureEliminator

class CorrelationEliminator(FeatureEliminator):
    """
    Keeps features that have an absolute correlation higher than a specified threshold
    with the target variable.
    """

    def __init__(self, X, y, correlation_threshold=0.1, num_features=None):
        """
        Initialize the CorrelationEliminator with a correlation threshold.

        Parameters:
        - X : Feature array. numpy ndarray.
        - y : Target array. numpy ndarray.
        - correlation_threshold : Minimum absolute correlation with the target required to keep a
feature.
        - num_features : Number of top features to keep.
        """
        self.correlation_threshold = correlation_threshold
        self.num_features = num_features
        super().__init__(X, y)

    def select_features(self):
        """
        Select features based on their absolute correlation with the target variable.
        """
        feature_means = np.mean(self.X, axis=0)
        target_mean = np.mean(self.y)

        feature_std = np.sqrt(np.sum((self.X - feature_means) ** 2, axis=0))
        target_std = np.sqrt(np.sum((self.y - target_mean) ** 2))

        # Ensure `self.y` is a NumPy array and reshape it
        y_reshaped = np.array(self.y).reshape(-1, 1)
        numerator = np.sum((self.X - feature_means) * (y_reshaped - target_mean), axis=0)

        denominator = feature_std * target_std
```

```
        denominator[denominator == 0] = 1e6  # Avoid division by zero

        # Compute the correlation coefficients
        feature_correlations = numerator / denominator

        if self.num_features is not None:
            # Select top `num_features` based on absolute correlation
            sorted_indices = np.argsort(-np.abs(feature_correlations))
            self.feature_indices = sorted_indices[:self.num_features]
            self.feature_mask = np.zeros(self.X.shape[1], dtype=bool)
            self.feature_mask[self.feature_indices] = True
        elif self.correlation_threshold is not None:
            # Select features with absolute correlation above the threshold
            self.feature_mask = np.abs(feature_correlations) > self.correlation_threshold
            self.feature_indices = [i for i, keep in enumerate(self.feature_mask) if keep]
```

Appendix N: Lasso Eliminator

```
import numpy as np
from .FeatureEliminator import FeatureEliminator
from Models.LinearRegression import LinearRegression

class LassoEliminator(FeatureEliminator):
    """
    Feature eliminator using Lasso (L1 regularization) regression.
    Features with coefficients close to zero are eliminated.
    """

    def __init__(self, X, y, l1=0.1, threshold=1e-5):
        """
        Initialize the LassoFeatureEliminator.

        Parameters:
        - X : Feature array. numpy ndarray.
        - y : Target array. numpy ndarray.
        - l1 : L1 regularization strength. Controls sparsity.
        - threshold : Threshold below which feature coefficients are considered zero.
        """
        self.l1 = l1
        self.threshold = threshold
        super().__init__(X, y)

    def select_features(self):
        """
        Select features based on Lasso regression coefficients.
        Features with coefficients close to zero are eliminated.
        """
        # Fit a Lasso regression model
        lasso_model = LinearRegression(fit_method='gd', l1=self.l1, loss_function="rmse", epochs=10000, learning_rate=0.1, gradient_descent='mini-batch', batch_size=32)  # high iteration low learning rate my favorite
        lasso_model.fit(self.X, self.y)

        # Extract coefficients
        coefficients = lasso_model.weights[1:] # 1: beacuse first one is bias
        print(coefficients)
```

```
        # Identify features with coefficients above the threshold
        self.feature_mask = np.abs(coefficients) > self.threshold
        self.feature_indices = [i for i, keep in enumerate(self.feature_mask) if keep]
```

Appendix O: Feature Elimination

Feature Elimination

1- Initial Preprocessing

In [1]:

```
import numpy as np
import pandas as pd

import sys
import os
sys.path.append(os.path.abspath('../'))

from Models.LinearRegression import LinearRegression
from Utils.Preprocessor import Preprocessor
from Utils.Utils import root_mean_squared_error, train_test_split, initial_preprocessing
from Utils.FeatureEliminators.VarianceEliminator import VarianceEliminator
from Utils.FeatureEliminators.CorrelationEliminator import CorrelationEliminator
from Utils.FeatureEliminators.LassoEliminator import LassoEliminator
```

In [2]:

```
# Read the data
train = pd.read_csv('../Data/train.csv', index_col='Id')
```

In [3]:

```
# Remove unnecessary features based on exploratory data analysis part 1.
train = initial_preprocessing(train)
```

In [4]:

```
X = train.drop(columns=["num_wins_agent1", "num_draws_agent1", "num_losses_agent1",
"utility_agent1"], axis=1)
y = train["utility_agent1"]
```

In [5]:

```
# Split the data into training and testing sets
X_train, X_valid, y_train, y_valid= train_test_split(X, y, test_size=0.2, random_state=42)
```

In [6]:

```
# Preprocess the data
preprocessor = Preprocessor(normalize=True, one_hot_encode=True)

X_train = preprocessor.fit_transform(X_train)
X_train = pd.DataFrame(X_train, columns=preprocessor.get_column_names())

X_valid = preprocessor.transform(X_valid)
X_valid = pd.DataFrame(X_valid, columns=preprocessor.get_column_names())

y_train.reset_index(drop=True, inplace=True)
y_valid.reset_index(drop=True, inplace=True)
```

Reminder: Linear Regression Baseline

Linear Regression:
Train mean squared error: 0.5175135945021986
Validation mean squared error: 0.51911678407925

Method 1: Variance Thresholding

Method 1 version 1

In [7]:

```
variance_eliminator = VarianceEliminator(X_train, y_train, threshold=0.01)

selected_features = variance_eliminator.get_feature_indices()
variance_1_mask = variance_eliminator.get_feature_mask()

X_train_var_1 = X_train.iloc[:, selected_features]
X_test_var_1 = X_valid.iloc[:, selected_features]
```
In [8]:
```
# to numpy array
X_train_var_1 = X_train_var_1.to_numpy()
X_test_var_1 = X_test_var_1.to_numpy()

lr_model = LinearRegression(fit_method="ols", loss_function="rmse")

lr_model.fit(X_train_var_1, y_train)

train_pred = lr_model.predict(X_train_var_1)
test_pred = lr_model.predict(X_test_var_1)

print("Linear Regression: ")
print("Number of used featues: ", len(selected_features))
print("Train mean squared error: ", root_mean_squared_error(y_train, train_pred))
print("Validation mean squared error: ", root_mean_squared_error(y_valid, test_pred))
```
```
Linear Regression:
Number of used featues:  336
Train mean squared error:  0.5306881182466742
Validation mean squared error:  0.5315929002975792
```
Method 1 version 2

In [9]:
```
variance_eliminator = VarianceEliminator(X_train, y_train, threshold=0.1)

variance_2_selected_features = variance_eliminator.get_feature_indices()
variance_2_mask = variance_eliminator.get_feature_mask()

X_train_var_2 = X_train.iloc[:, variance_2_selected_features]
X_test_var_2 = X_valid.iloc[:, variance_2_selected_features]
```
In [10]:
```
# to numpy array
X_train_var_2 = X_train_var_2.to_numpy()
X_test_var_2 = X_test_var_2.to_numpy()

lr_model = LinearRegression(fit_method="ols", loss_function="rmse")

lr_model.fit(X_train_var_2, y_train)

train_pred = lr_model.predict(X_train_var_2)
test_pred = lr_model.predict(X_test_var_2)

print("Linear Regression: ")
print("Number of used featues: ", len(variance_2_selected_features))
print("Train mean squared error: ", root_mean_squared_error(y_train, train_pred))
print("Validation mean squared error: ", root_mean_squared_error(y_valid, test_pred))
```
```
Linear Regression:
Number of used featues:  135
Train mean squared error:  0.5995919481377527
```

Validation mean squared error: 0.6004933618410249

Method 2: Correlation Thresholding

Method 2 Version 1

In [11]:

```
correlation_eliminator_1 = CorrelationEliminator(X_train, y_train,
correlation_threshold=0.01)

corr_1_selected_features = correlation_eliminator_1.get_feature_indices()
corr_1_mask = variance_eliminator.get_feature_mask()

X_train_corr_1 = X_train.iloc[:, corr_1_selected_features]
X_test_corr_1 = X_valid.iloc[:, corr_1_selected_features]
```

In [12]:

```
# to numpy array
X_train_corr_1 = X_train_corr_1.to_numpy()
X_test_corr_1 = X_test_corr_1.to_numpy()

lr_model = LinearRegression(fit_method="ols", loss_function="rmse")

lr_model.fit(X_train_corr_1, y_train)

train_pred = lr_model.predict(X_train_corr_1)
test_pred = lr_model.predict(X_test_corr_1)

print("Linear Regression: ")
print("Number of used featues: ", len(corr_1_selected_features))
print("Train mean squared error: ", root_mean_squared_error(y_train, train_pred))
print("Validation mean squared error: ", root_mean_squared_error(y_valid, test_pred))
```

Linear Regression:
Number of used featues: 269
Train mean squared error: 0.5324470589299449
Validation mean squared error: 0.5340079095469382

Method 2 Version 2

In [13]:

```
correlation_eliminator_2 = CorrelationEliminator(X_train, y_train,
correlation_threshold=0.03)

corr_2_selected_features = correlation_eliminator_2.get_feature_indices()
corr_2_mask = variance_eliminator.get_feature_mask()

X_train_corr_2 = X_train.iloc[:, corr_2_selected_features]
X_test_corr_2 = X_valid.iloc[:, corr_2_selected_features]
```

In [14]:

```
# to numpy array
X_train_corr_2 = X_train_corr_2.to_numpy()
X_test_corr_2 = X_test_corr_2.to_numpy()

lr_model = LinearRegression(fit_method="ols", loss_function="rmse")

lr_model.fit(X_train_corr_2, y_train)

train_pred = lr_model.predict(X_train_corr_2)
test_pred = lr_model.predict(X_test_corr_2)

print("Linear Regression: ")
```

```
print("Number of used featues: ", len(corr_2_selected_features))
print("Train mean squared error: ", root_mean_squared_error(y_train, train_pred))
print("Validation mean squared error: ", root_mean_squared_error(y_valid, test_pred))
```

Linear Regression:
Number of used featues: 76
Train mean squared error: 0.5435803901515012
Validation mean squared error: 0.5459862384342256

Method 3: Lasso Eliminator

In [27]:

```
lasso_eliminator = LassoEliminator(X_train, y_train, l1=1, threshold=1e-1)

lasso_selected_features = lasso_eliminator.get_feature_indices()
lasso_mask = variance_eliminator.get_feature_mask()

X_train_lasso = X_train.iloc[:, lasso_selected_features]
X_test_lasso = X_valid.iloc[:, lasso_selected_features]

# to numpy array
X_train_lasso = X_train_lasso.to_numpy()
X_test_lasso = X_test_lasso.to_numpy()

lr_model = LinearRegression(fit_method="ols", loss_function="rmse")

lr_model.fit(X_train_lasso, y_train)

train_pred = lr_model.predict(X_train_lasso)
test_pred = lr_model.predict(X_test_lasso)

print("Linear Regression: ")
print("Number of used featues: ", len(lasso_selected_features))
print("Train mean squared error: ", root_mean_squared_error(y_train, train_pred))
print("Validation mean squared error: ", root_mean_squared_error(y_valid, test_pred))
```

Linear Regression:
Number of used featues: 346
Train mean squared error: 0.5340242905222083
Validation mean squared error: 0.5358824169618749