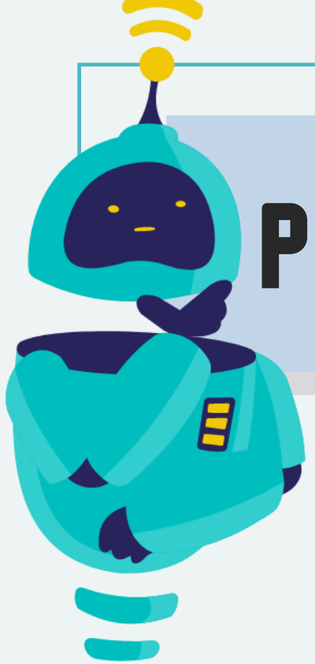
FORMULA 1 RACE TIME ANALYSIS WITH MACHINE LEARNING ALGORITHMS

TABLE OF CONTENTS



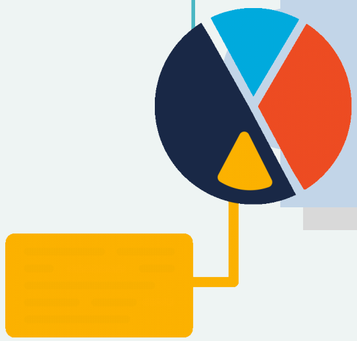
PURPOSE OF THE PROJECT

- **The aim is to create a prediction system that can develop race strategies, analyze performance and provide competitive advantage by using machine learning models to predict the finish time in Formula 1 races.**

IMPORTANCE OF THE PROJECT

- **Precise prediction models developed with simulations allow different scenarios to be tested before the race, while they can be used effectively to analyze the strengths and weaknesses of competitors and to increase accuracy in the legal betting sector and to obtain more reliable results.**

DATASET INTRODUCTION



races.csv

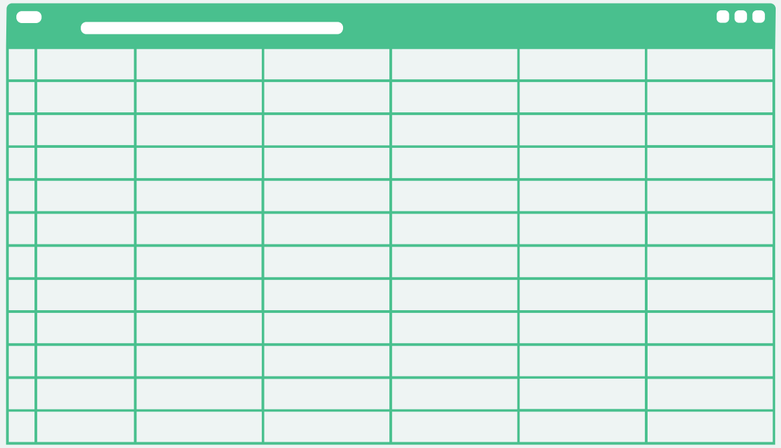
drivers.csv

result.csv

constr-uctors.csv

DATASET INTRODUCTION

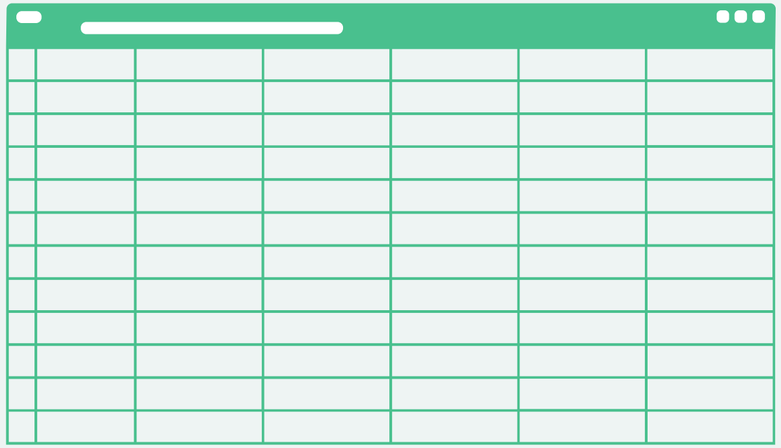
result.csv



- In this csv we have 19 columns and 26519 rows
- resultId, raceId, driverId, constructorId, number, grid, position, positionText, positionOrder, points, laps, time, milliseconds, fastestLap, rank, fastestLapTime, fastestLapSpeed, statusId

DATASET INTRODUCTION

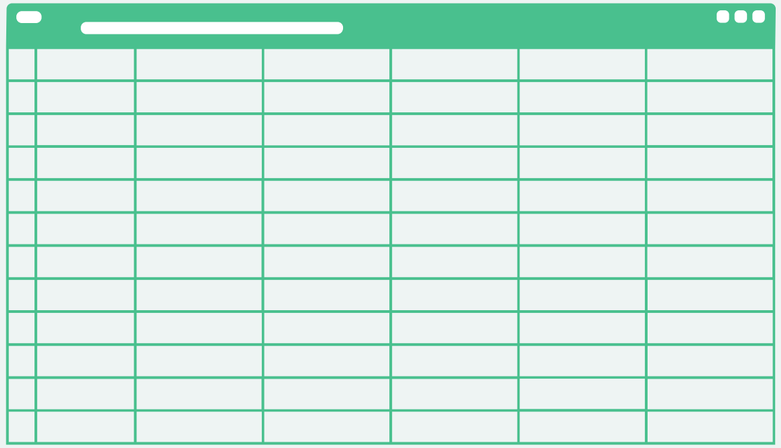
constructors.csv



- In this csv we have 5 columns and 212 rows
- constructorId, constructorRef, name, nationality, url
- We merge this csv with the result csv on constructorId, thus obtaining constructorRef

DATASET INTRODUCTION

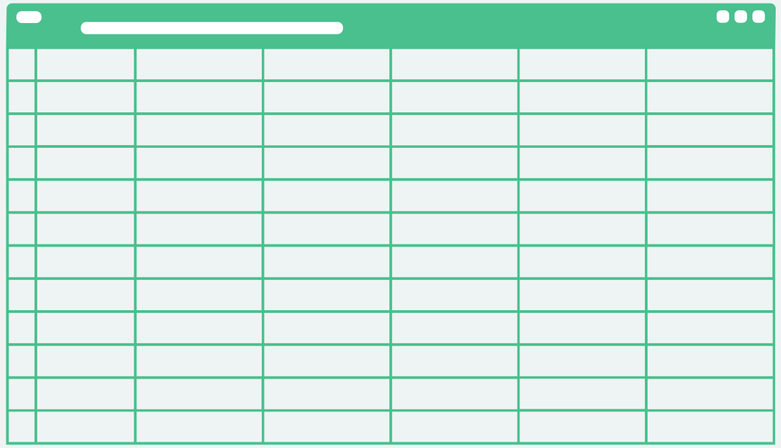
races.csv



- In this csv we have 8 columns and 1125 rows
- raceId, year, round, circuitId, name, date, time, url
- We merge this csv with the result csv on raceId, thus obtaining year

DATASET INTRODUCTION

drivers.csv

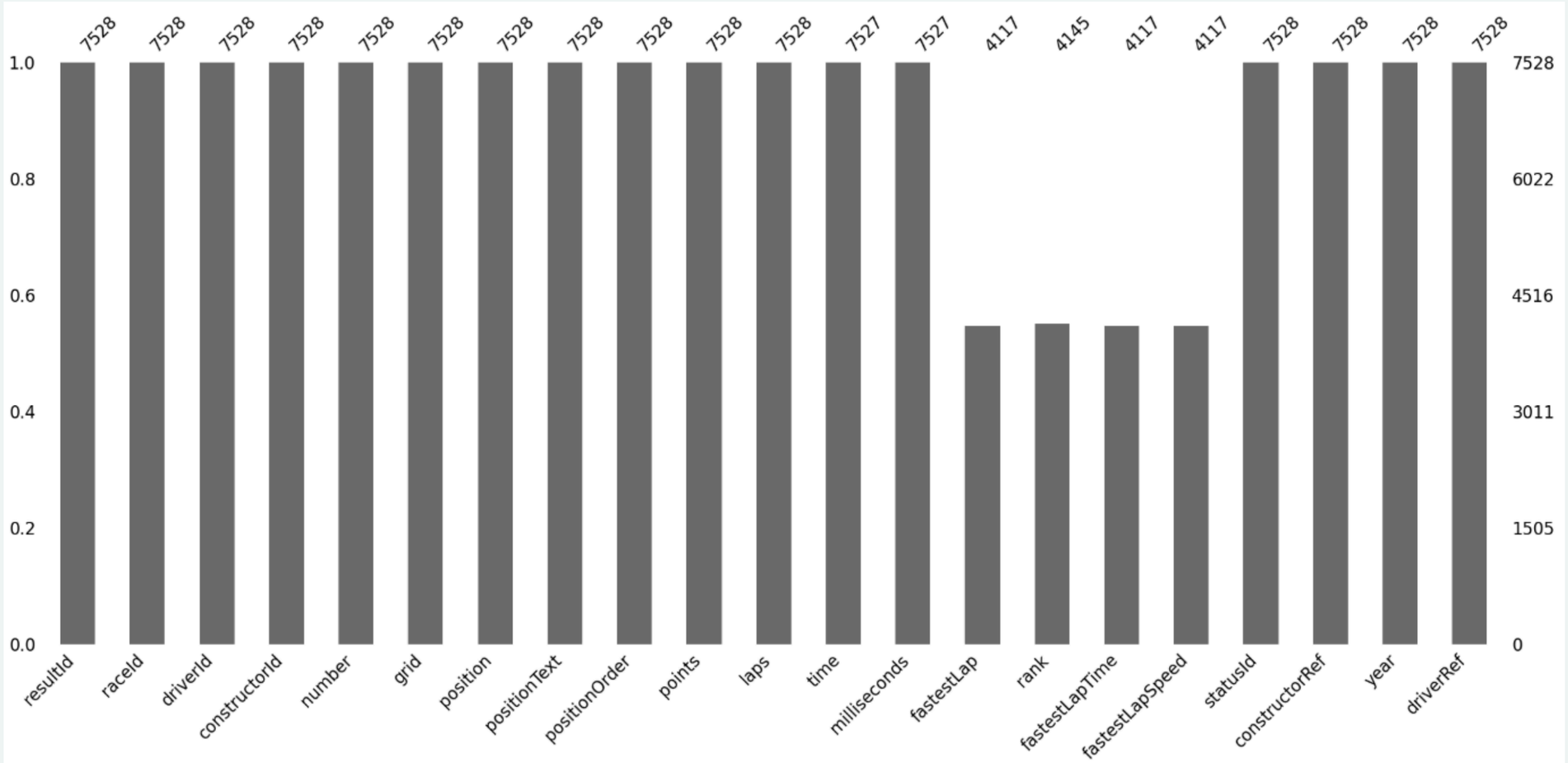


- In this csv we have 8 columns and 1125 rows
- driverId, driverRef, number, code, forename, surname, dob, nationality, url
- We merge this csv with the result csv on driverId, thus obtaining driverRef

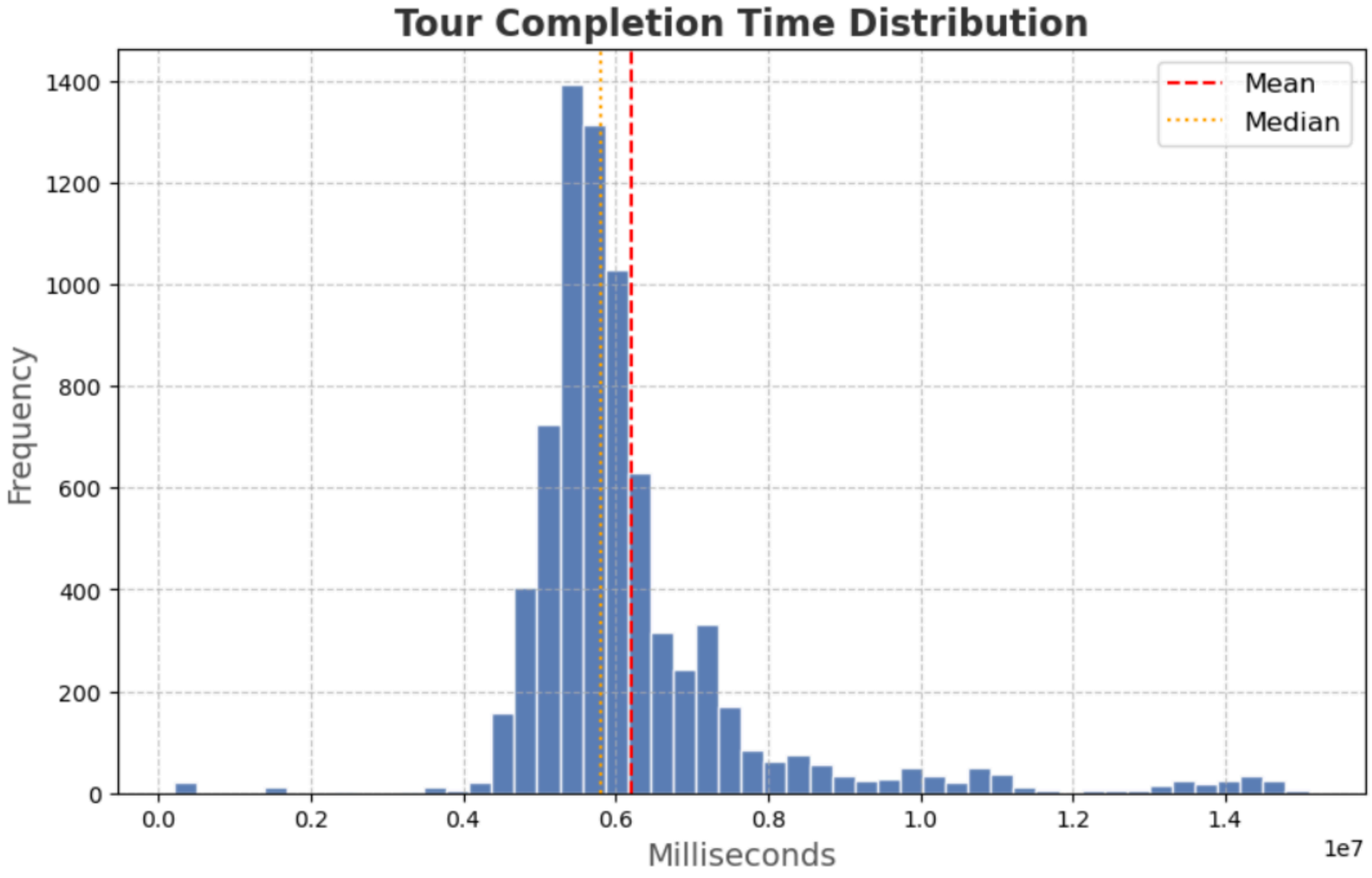
DATASET INTRODUCTION

```
result_df = result_df.merge(constructor_df[['constructorId', 'constructorRef']], on='constructorId', how='left')
result_df = result_df.merge(races_df[['raceId', 'year']], on='raceId', how='left')
result_df = result_df.merge(drivers_df[['driverId', 'driverRef']], on='driverId', how='left')
result_df

result_df=result_df[result_df.statusId==1]
result_df.reset_index(drop=True, inplace=True)
result_df
```

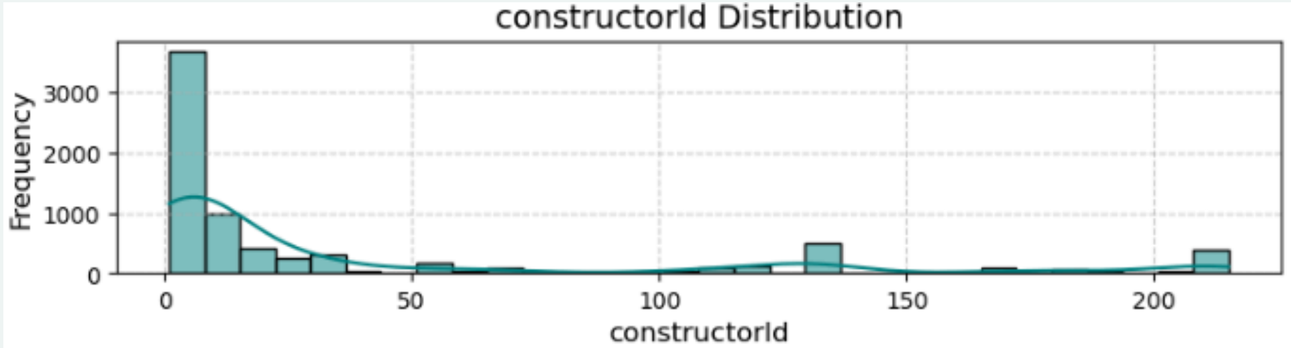
DATASET INTRODUCTION

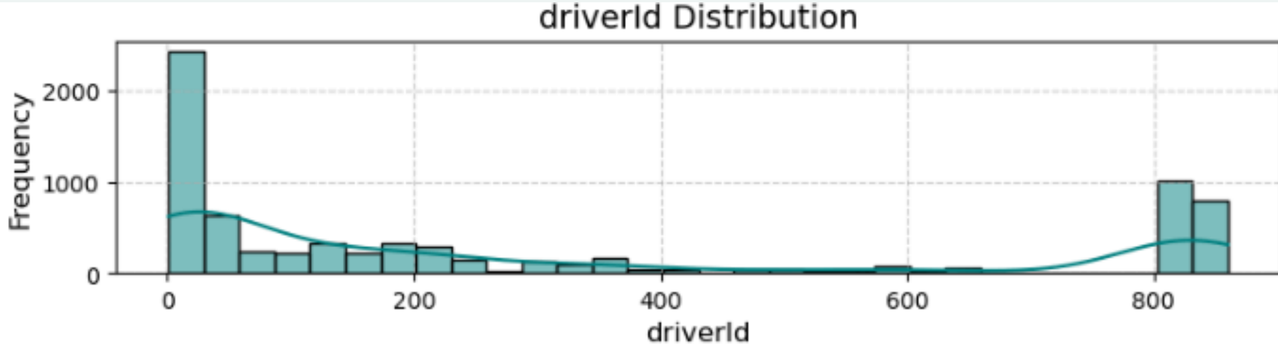


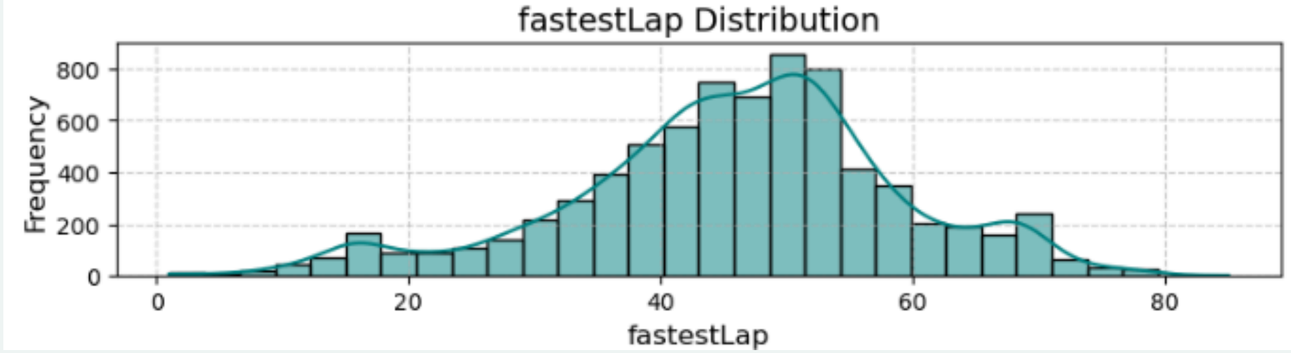
DATASET VISUALIZE

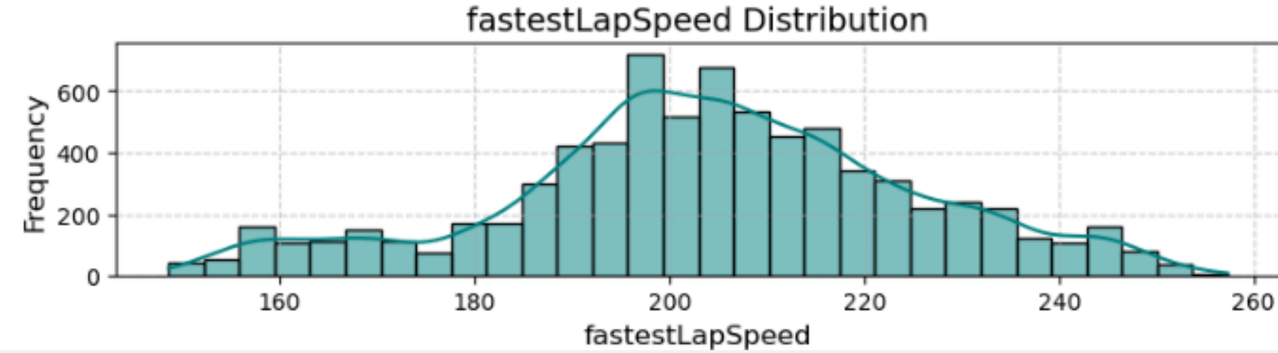


DATASET VISUALIZE

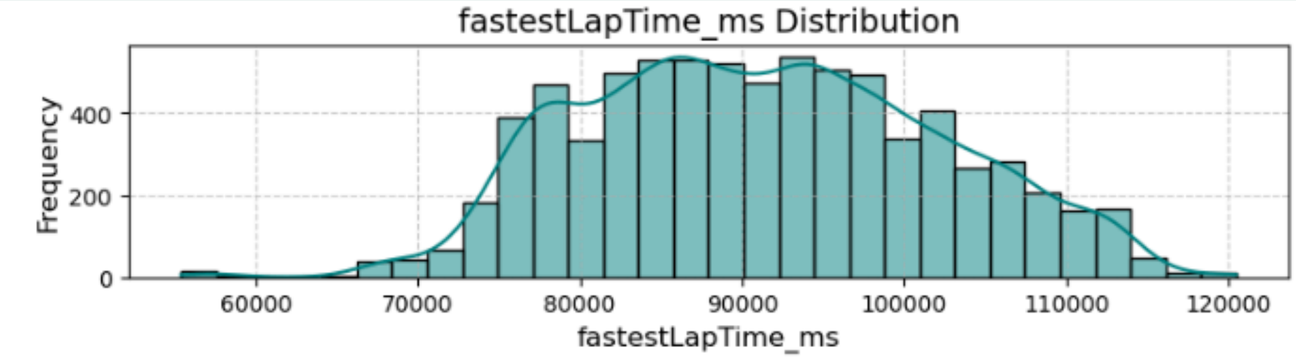


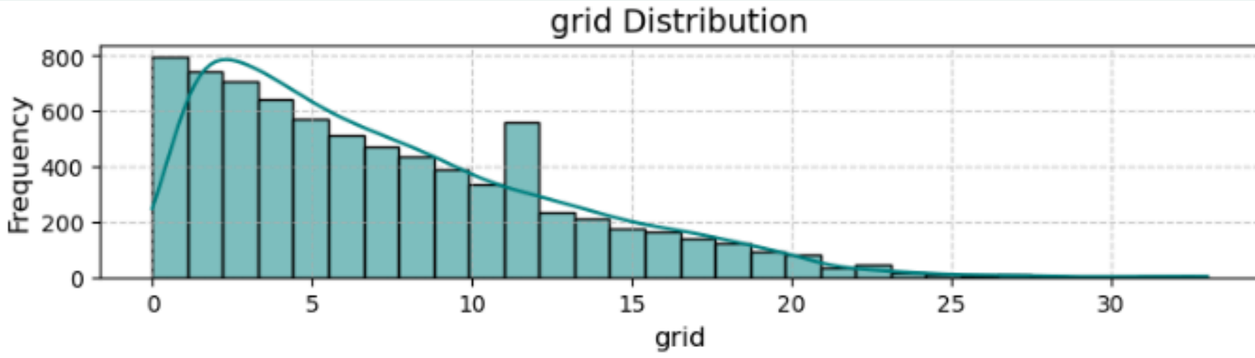


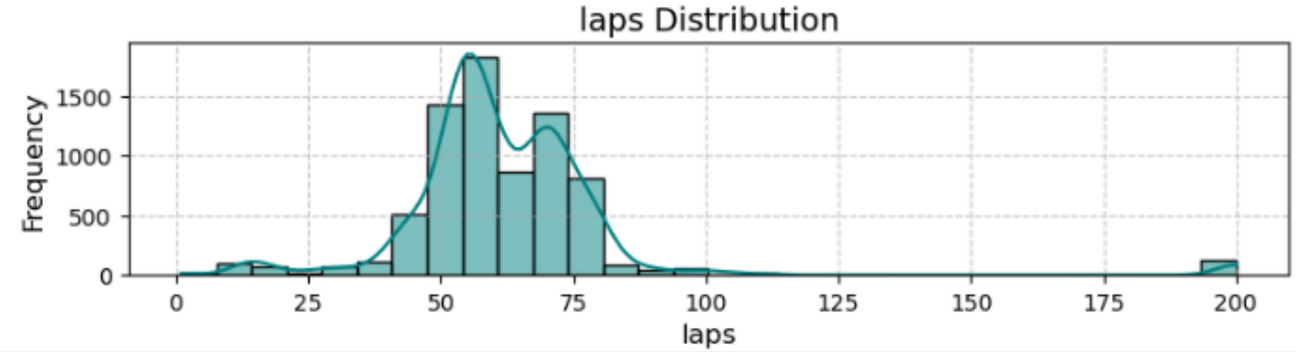


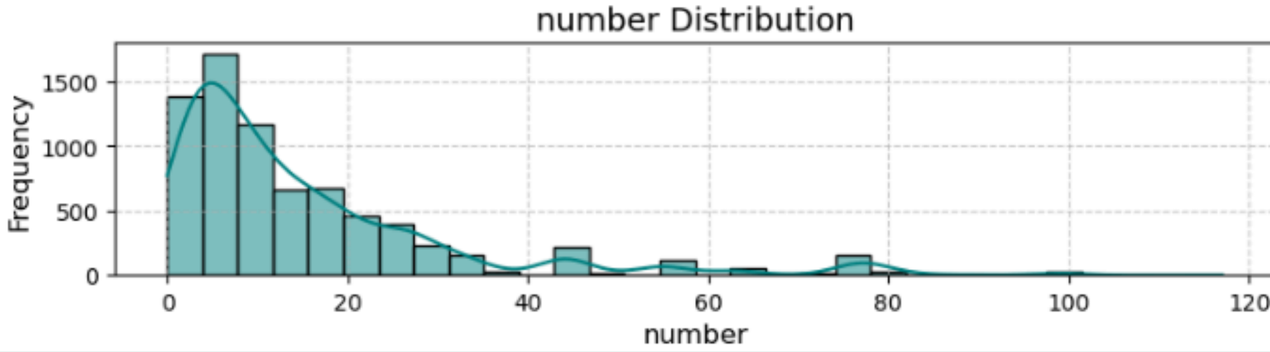


DATASET VISUALIZE

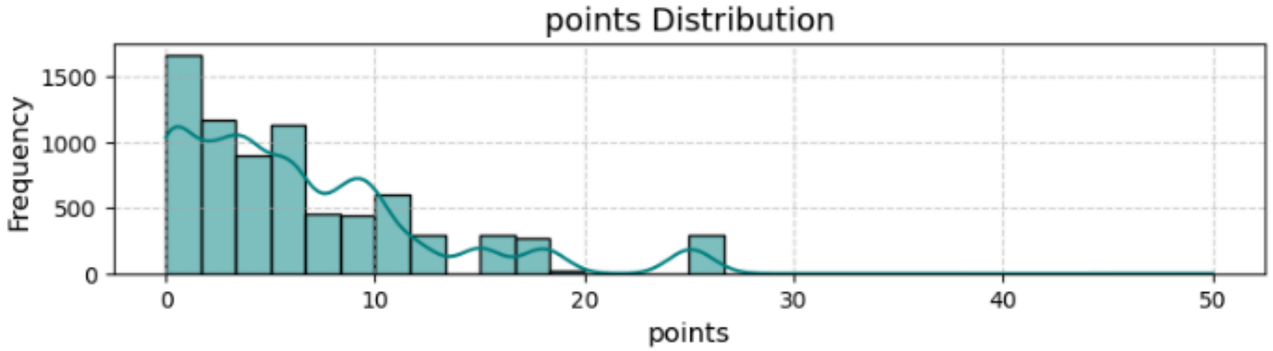


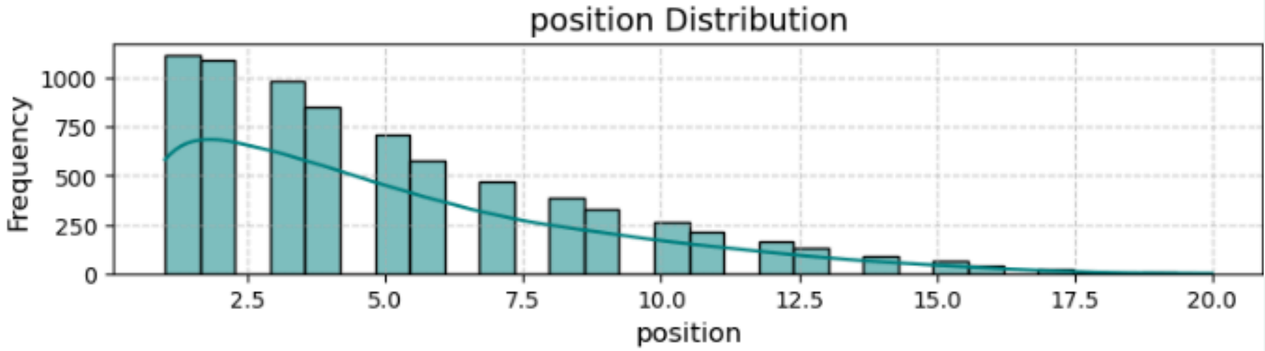


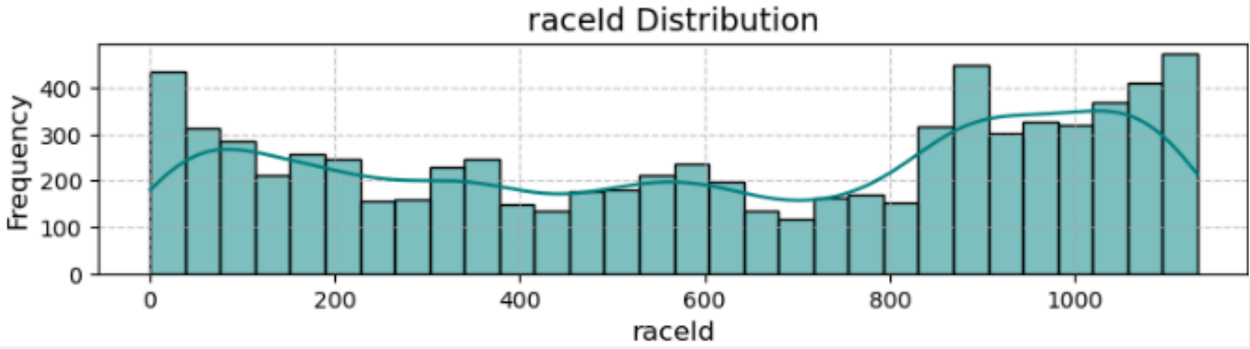




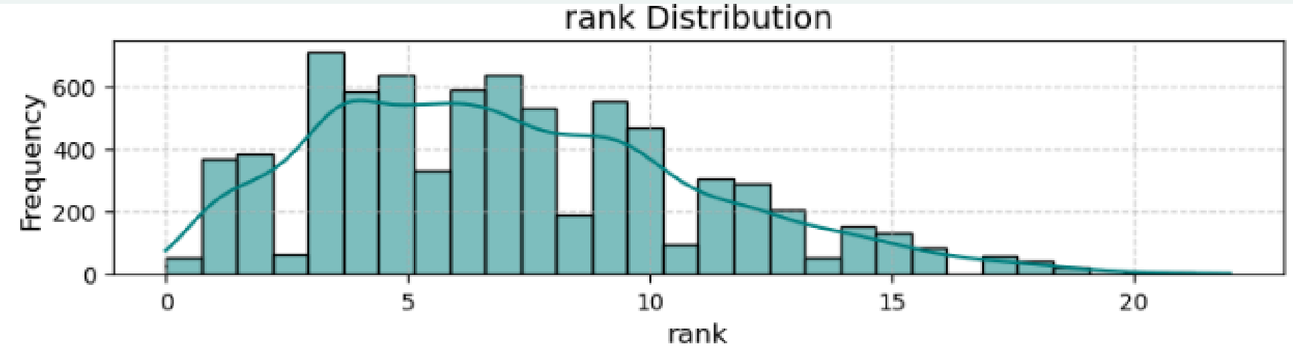
DATASET VISUALIZE

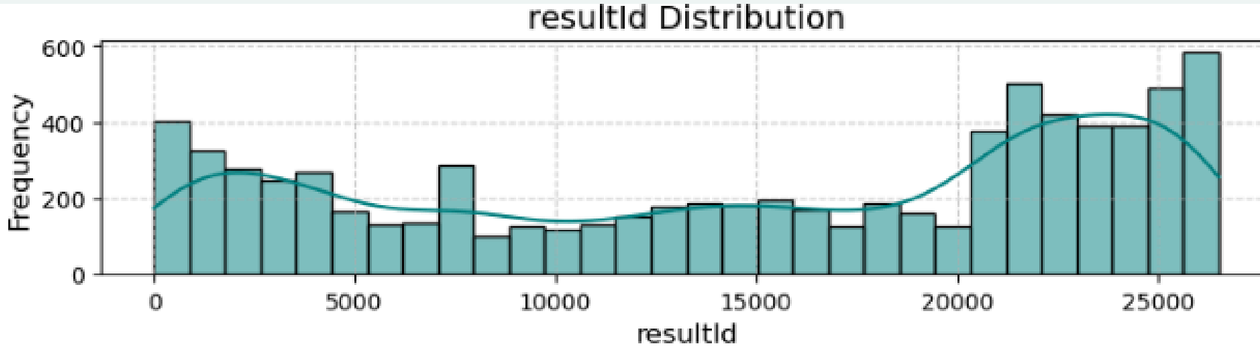


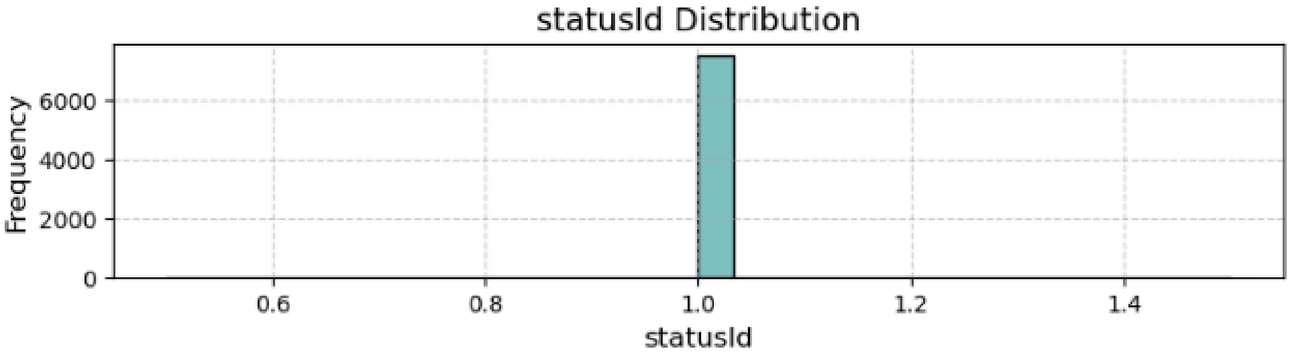


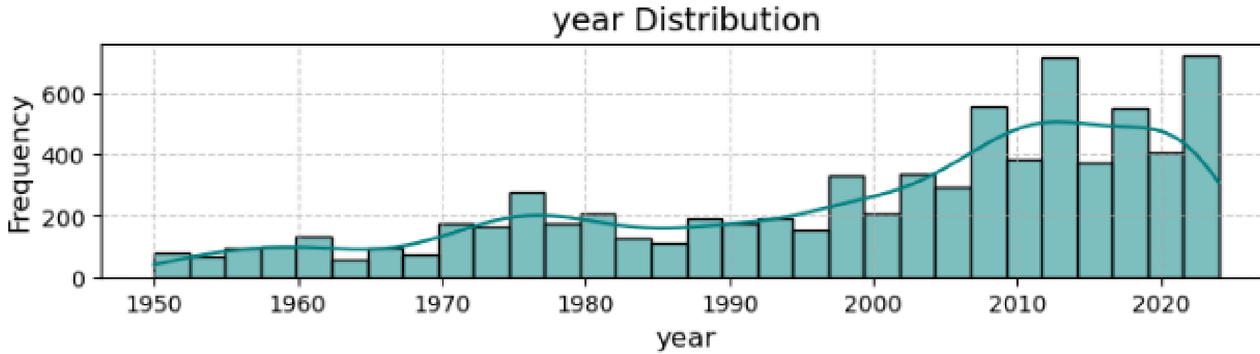


DATASET VISUALIZE

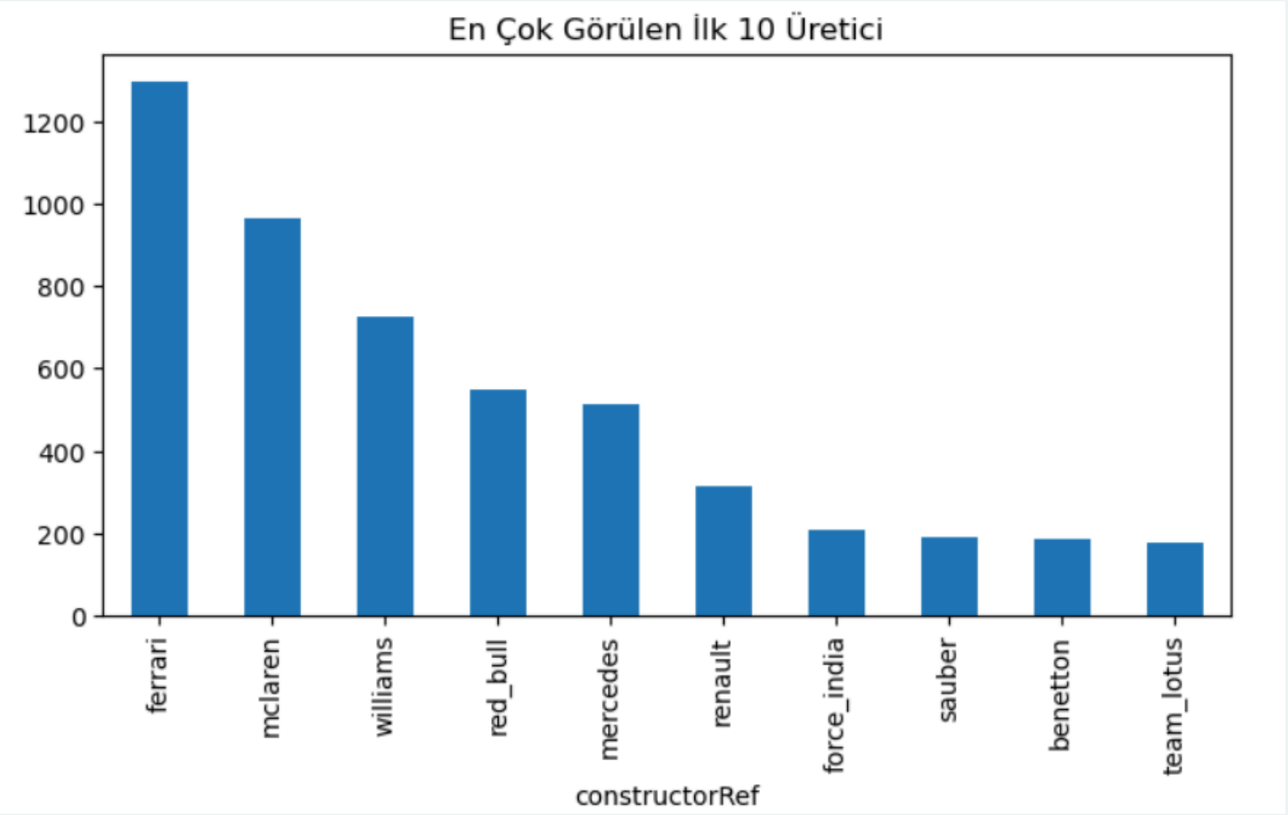


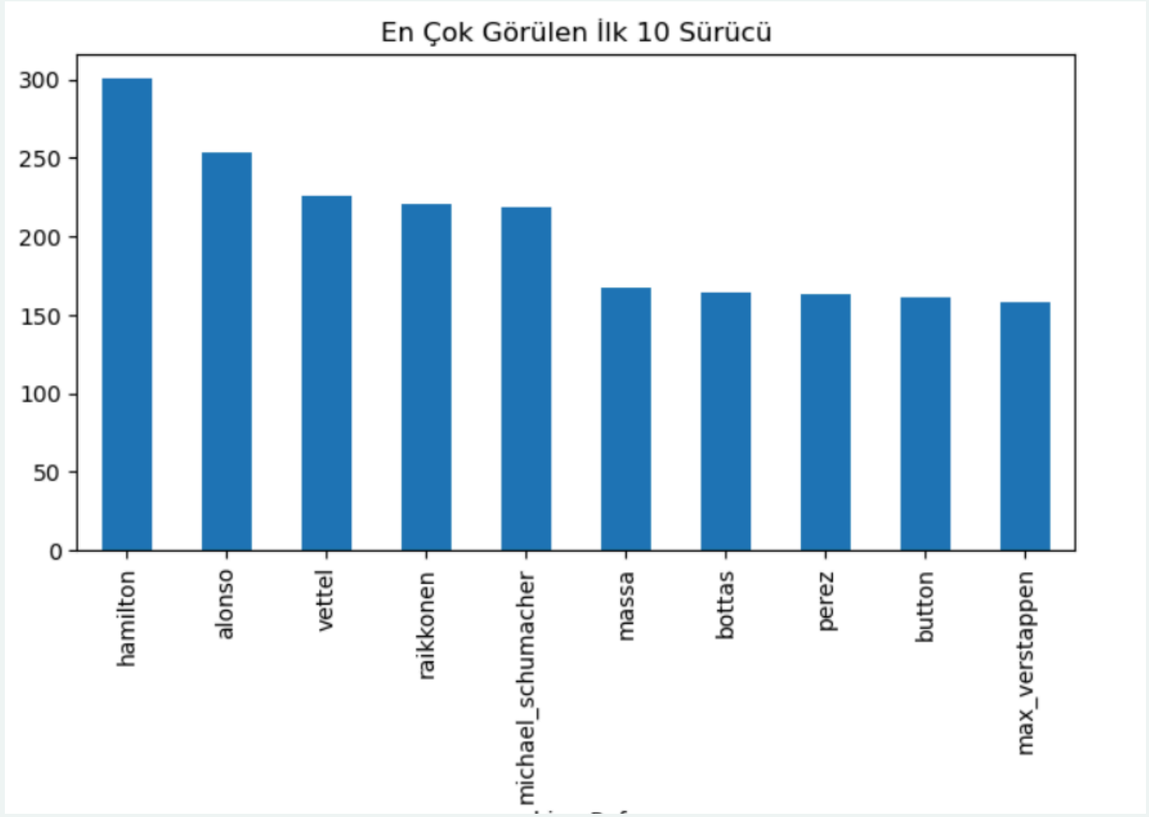




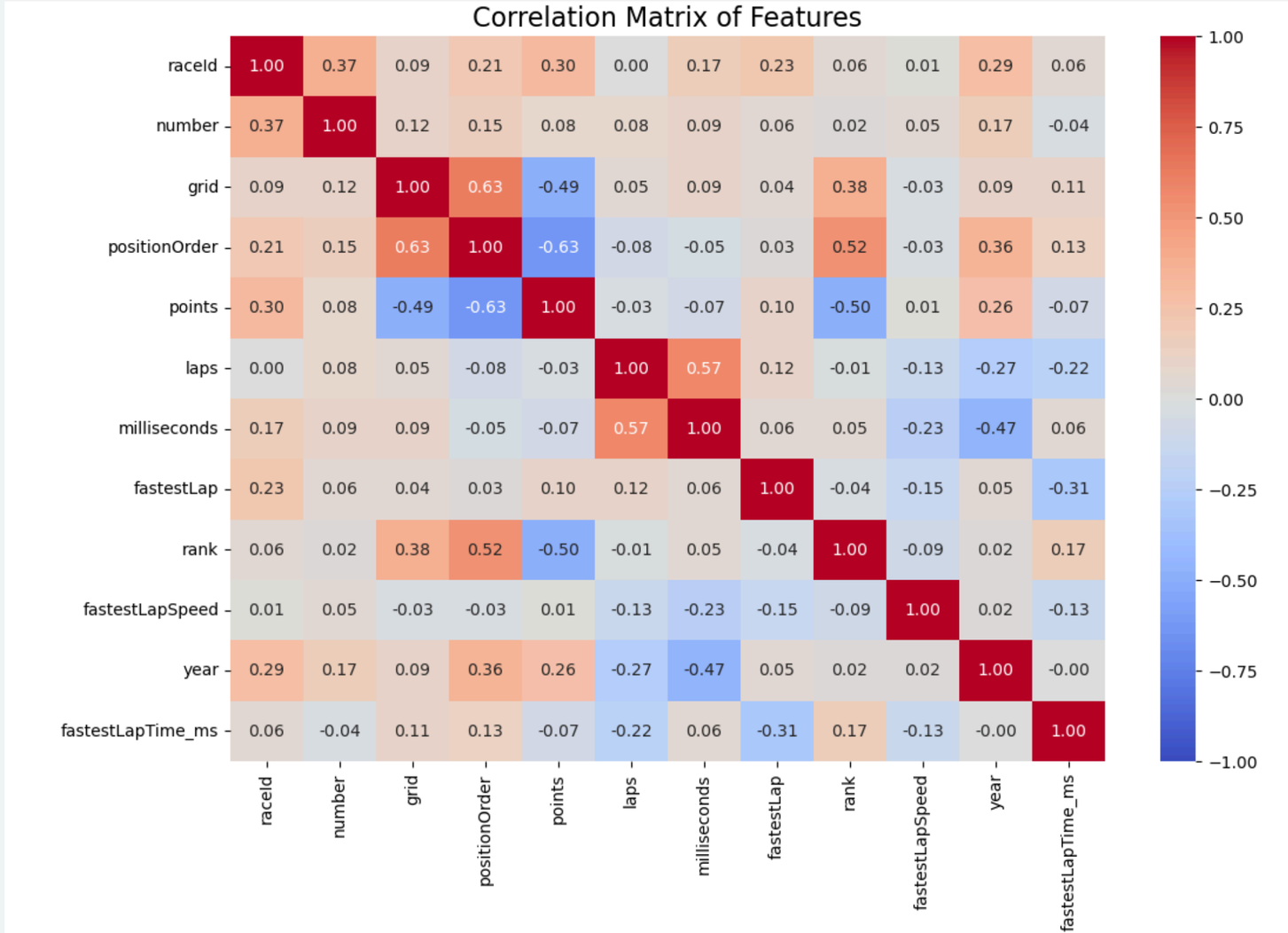


DATASET VISUALIZE





DATASET VISUALIZE



FEATURE SELECTION

- **Feature selection is a critical step in machine learning projects to improve model performance, facilitate data processing processes, and make the algorithm faster and more efficient.**
- **Avoids the Curse of Dimensionality. In highly dimensional data, the distances between data points become increasingly homogeneous, weakening the learning ability of the model.**

FEATURE SELECTION

Mutual Information

Random Forest

FEATURE SELECTION

Mutual Information

- **Mutual information is a measure of the degree of dependence between two variables.**
- **It is a more powerful measure than correlation because it can also detect nonlinear dependencies.**
- **It is used to evaluate how well a feature explains a target variable.**
  A value close to zero indicates that the feature is independent of the target.
  A high value indicates strong dependence.

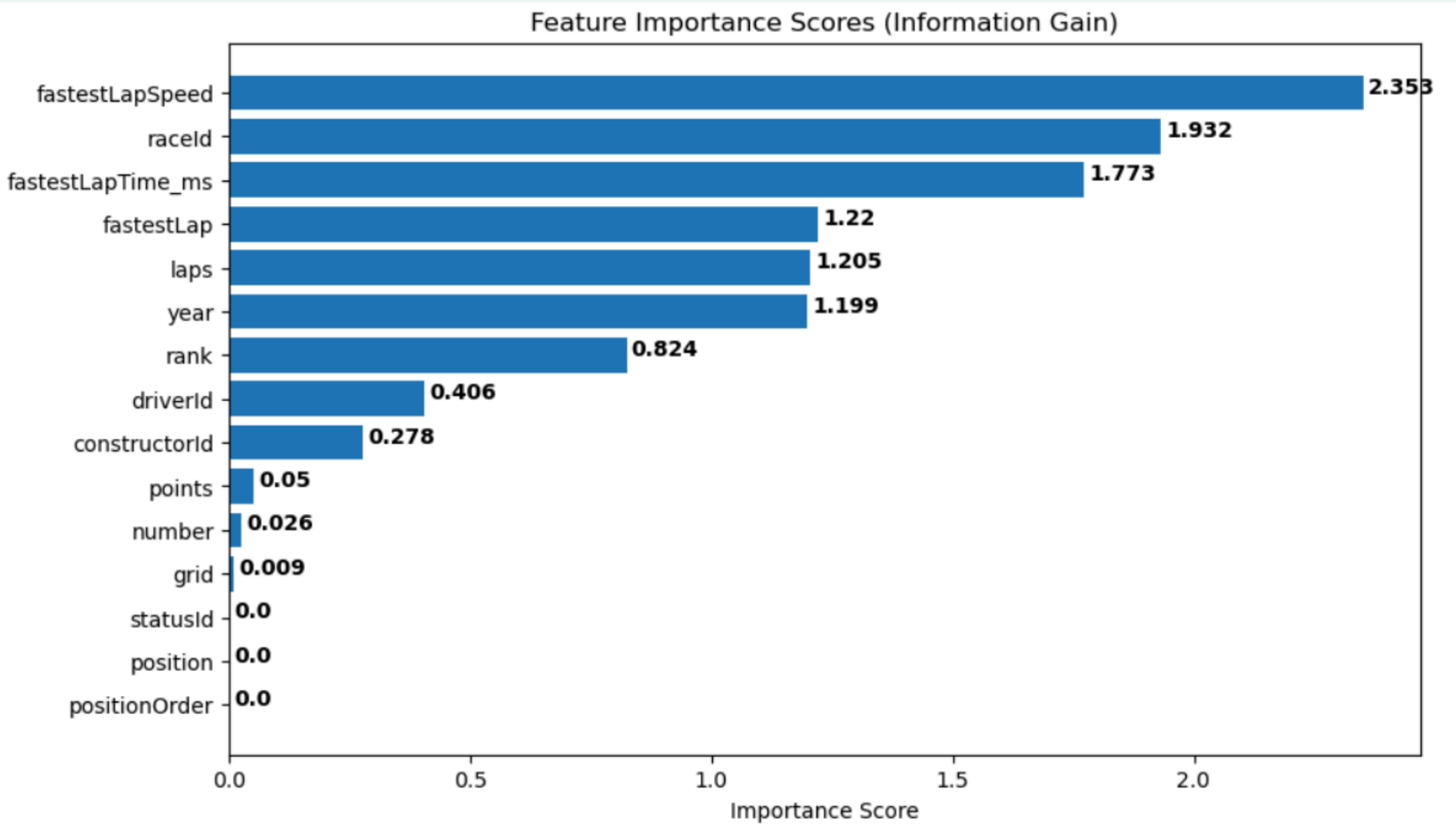
FEATURE SELECTION

Mutual Information

1. **Entropy** $H(X)=-X\in X\sum P(X)LOG2P(X)$
2. **Joint Entropy** $H(X,Y)=-X\in X\sum Y\in Y\sum P(X,Y)LOG2P(X,Y)$
3. **Mutual Information** $I(X;Y)=H(X)+H(Y)-H(X,Y)$

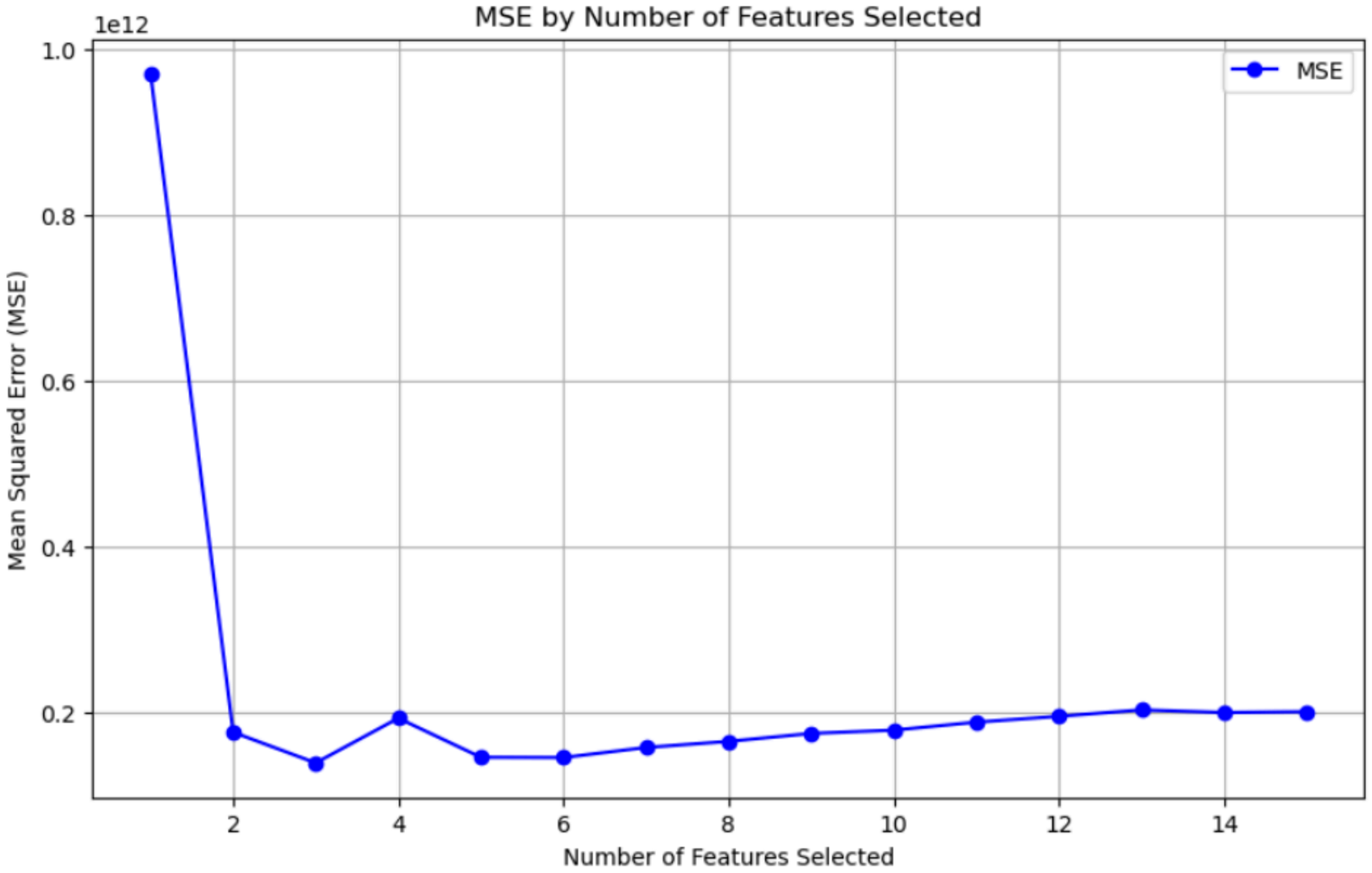
FEATURE SELECTION

Mutual Information



FEATURE SELECTION

Mutual Information



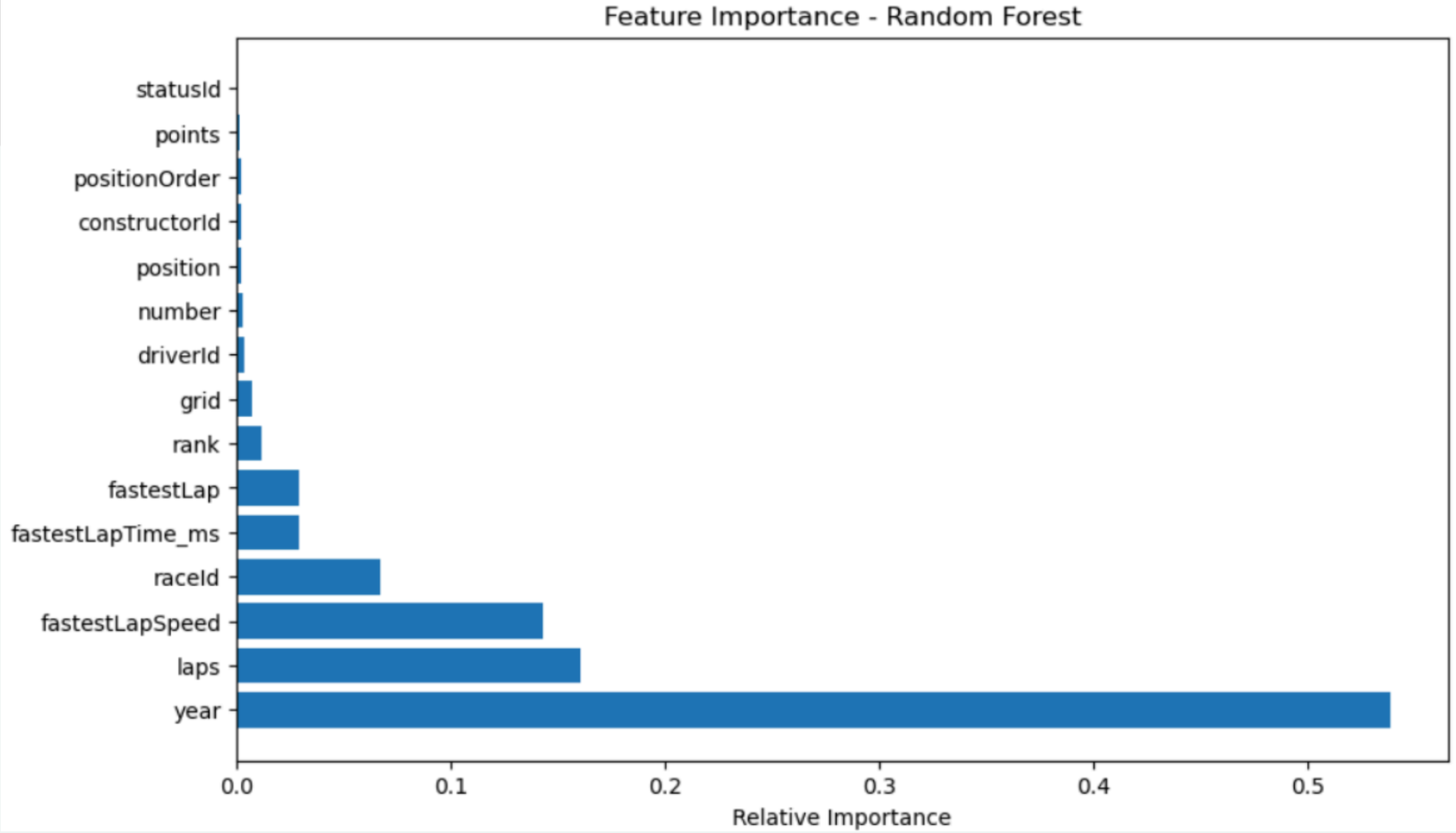
FEATURE SELECTION

Random Forest

- It learns the interactions between the features naturally within the model.
- It evaluates the effect of the features on the branching of the decision trees.
- Typically, the split decision is made based on Gini Impurity or Entropy and the contribution of each feature to the model accuracy is monitored.

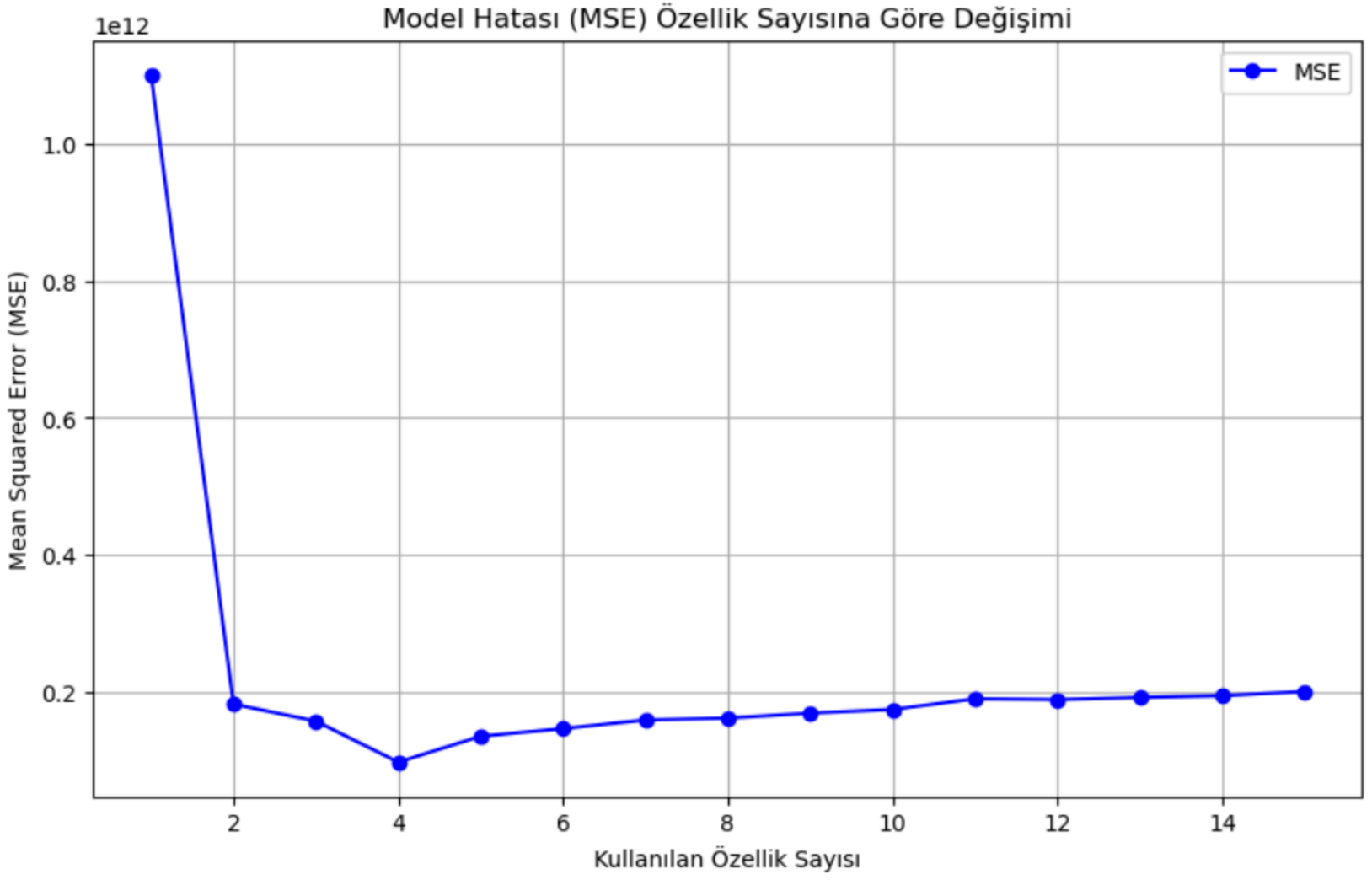
FEATURE SELECTION

Random Forest



FEATURE SELECTION

Random Forest



MIN-MAX SCALER

KNN-REGRESSION

DECISION TREE

RANDOM FOREST REGRESSION

SUPPORT VECTOR REGRESSION

ANN

GET DUMMIES

- **get_dummies() allows categorical variables to be evaluated independently by creating separate columns for each category.**
- **The get_dummies() function assigns a value of 0 and 1 to each category in the data frame. This eliminates problems such as sorting categorical data because each category is separated by different columns and is meaningful to the model.**

driverRef_hamilton	driverRef_alonso	driverRef_verstappen
1	0	0
0	1	0
0	0	1

MACHINE L. ALGORITHMS

- ANN
- DECISION TREE
- RANDOM FOREST REGRESSION
- SUPPORT VECTOR REGRESSION
- KNN-REGRESSION

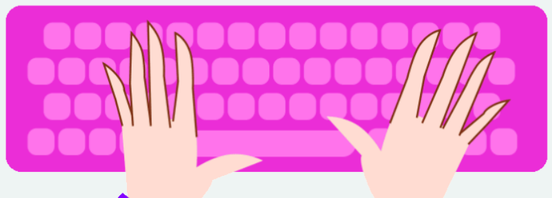
MACHINE L. ALGORITHMS

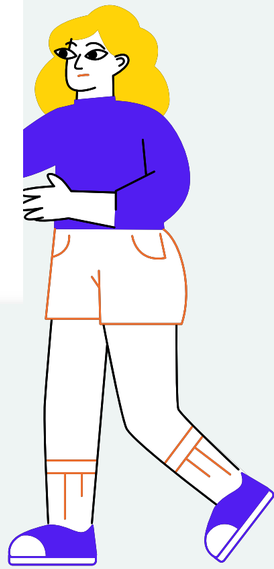
1 ANN

With Important Features

```
random_index = np.random.choice(X_test.index)
X_test_single = X_test.loc[[random_index]]
y_test_single = y_test.loc[random_index]



X_train_top = X_train.loc[:, features]
X_val_top = X_val.loc[:, features]
X_test_single_top = X_test_single.loc[:, features]
```

```
ANN Validation R-squared (R²): 0.7844941157047483
ANN Validation Mean Squared Error (MSE): 0.0024960496031191564


ANN Test Single Prediction: [[0.38880053]]
ANN Test Single Actual Value: 0.35798764387522825
ANN Percentage Error: [[8.607249]] %
ANN Test Single Mean Squared Error (MSE): 0.0009494340515261107
```





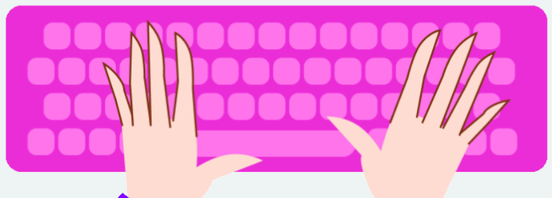
MACHINE L. ALGORITHMS

1 ANN

With All Features

```
random_index = np.random.choice(X_test.index)
X_test_single = X_test.loc[[random_index]]
y_test_single = y_test.loc[random_index]



X_train_top = X_train
X_val_top = X_val
X_test_single_top = X_test_single
```

```
ANN Validation R-squared (R²): 0.6505640110265111
ANN Validation Mean Squared Error (MSE): 0.004047265644022344

ANN Test Single Prediction: [[0.5219771]]
ANN Test Single Actual Value: 0.6229951498538412
ANN Percentage Error: [[16.214895]] %
ANN Test Single Mean Squared Error (MSE): 0.010204641022445312
```



MACHINE L. ALGORITHMS

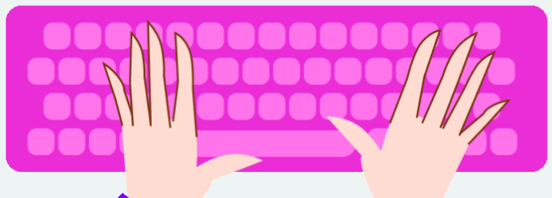
2

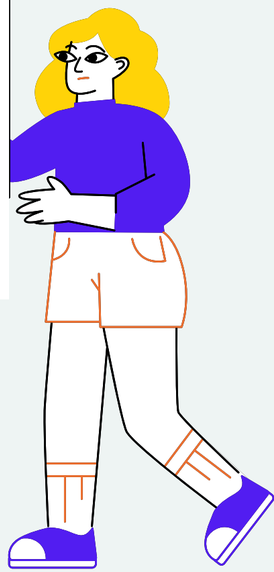
RANDOM FOREST

With Important Features

```
random_index = np.random.choice(X_test.index)
X_test_single = X_test.loc[[random_index]]
y_test_single = y_test.loc[random_index]



X_train_top = X_train.loc[:, features]
X_val_top = X_val.loc[:, features]
X_test_single_top = X_test_single.loc[:, features]
```

```
RF Validation R-squared (R²): 0.9736064604284964
RF Validation Mean Squared Error (MSE): 0.00030569737892680265

RF Test Single Prediction: [0.54813171]
RF Test Single Actual Value: 0.583098537041331
RF Percentage Error: 5.996727534069902 %
RF Test Single Mean Squared Error (MSE): 0.0012226792367204425
```





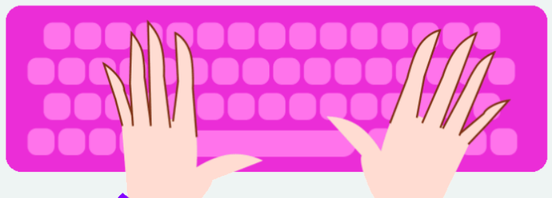
MACHINE L. ALGORITHMS

2 RANDOM FOREST

With All Features

```
random_index = np.random.choice(X_test.index)
X_test_single = X_test.loc[[random_index]]
y_test_single = y_test.loc[random_index]



X_train_top = X_train
X_val_top = X_val
X_test_single_top = X_test_single
```

```
RF Validation R-squared (R²): 0.9379755040450931
RF Validation Mean Squared Error (MSE): 0.0007183851105420702

RF Test Single Prediction: [0.45528189]
RF Test Single Actual Value: 0.4379046981587424
RF Percentage Error: 3.9682582740945147 %
RF Test Single Mean Squared Error (MSE): 0.00030196671204586674
```



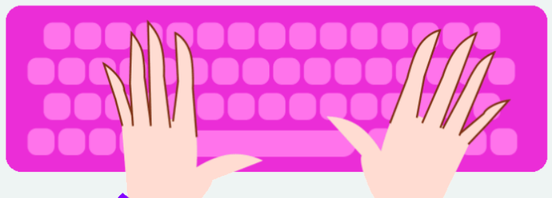
MACHINE L. ALGORITHMS

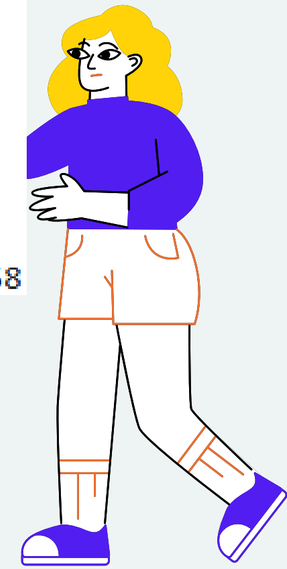
3 SUPPORT VECTOR REGRESSION

With Important Features

```
random_index = np.random.choice(X_test.index)
X_test_single = X_test.loc[[random_index]]
y_test_single = y_test.loc[random_index]



X_train_top = X_train.loc[:, features]
X_val_top = X_val.loc[:, features]
X_test_single_top = X_test_single.loc[:, features]
```

```
SVR Validation R-squared (R²): 0.6181484352276378
SVR Validation Mean Squared Error (MSE): 0.004733213736778346

SVR Test Single Prediction: [0.44596438]
SVR Test Single Actual Value: 0.4200651743219272
SVR Percentage Error: 6.165519709381232 %
SVR Test Single Mean Squared Error (MSE): 0.0006707686183985868
```





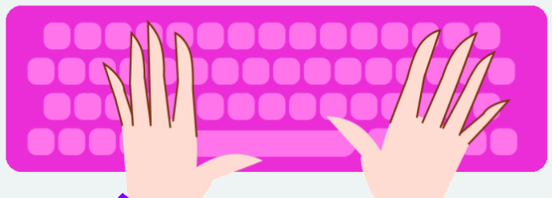
MACHINE L. ALGORITHMS

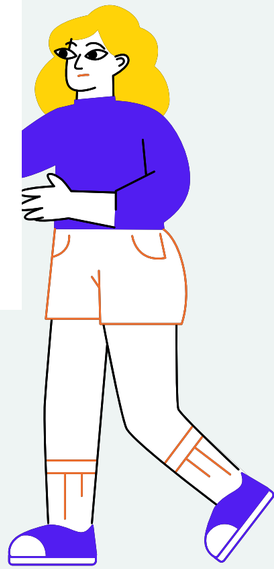
3 SUPPORT VECTOR REGRESSION

With All Features

```
random_index = np.random.choice(X_test.index)
X_test_single = X_test.loc[[random_index]]
y_test_single = y_test.loc[random_index]



X_train_top = X_train
X_val_top = X_val
X_test_single_top = X_test_single
```

```
SVR Validation R-squared (R²): 0.5536580832601906
SVR Validation Mean Squared Error (MSE): 0.0055325992781312065

SVR Test Single Prediction: [0.36551162]
SVR Test Single Actual Value: 0.32737723980881067
SVR Percentage Error: 11.648451935828001 %
SVR Test Single Mean Squared Error (MSE): 0.0014542309706251173
```





MACHINE L. ALGORITHMS

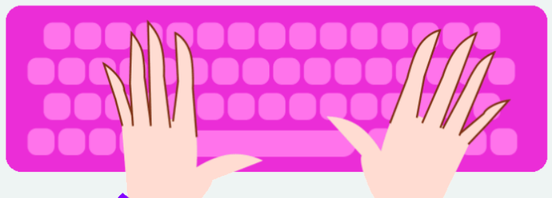
4

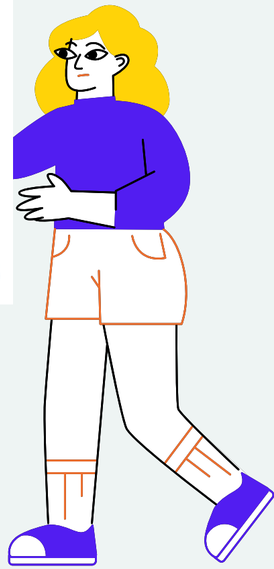
DECISION TREE

With Important Features

```
random_index = np.random.choice(X_test.index)
X_test_single = X_test.loc[[random_index]]
y_test_single = y_test.loc[random_index]



X_train_top = X_train.loc[:, features]
X_val_top = X_val.loc[:, features]
X_test_single_top = X_test_single.loc[:, features]
```

```
DT Validation R-squared (R²): 0.9745050650248008
DT Validation Mean Squared Error (MSE): 0.00027847320312425906

DT Test Single Prediction: [0.57563388]
DT Test Single Actual Value: 0.5867871932276004
DT Percentage Error: 1.9007425376676244 %
DT Test Single Mean Squared Error (MSE): 0.00012439640843714913
```





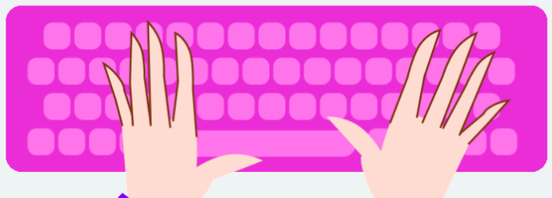
MACHINE L. ALGORITHMS

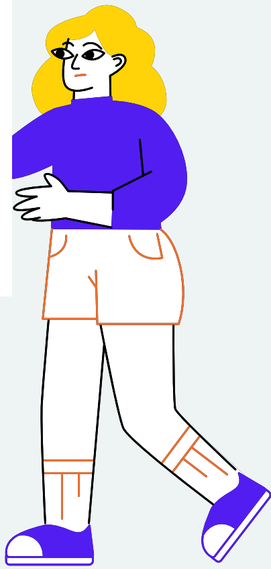
4 DECISION TREE

With All Features

```
random_index = np.random.choice(X_test.index)
X_test_single = X_test.loc[[random_index]]
y_test_single = y_test.loc[random_index]



X_train_top = X_train
X_val_top = X_val
X_test_single_top = X_test_single
```

```
DT Validation R-squared (R²): 0.8356061896768641
DT Validation Mean Squared Error (MSE): 0.0017956221884471715

DT Test Single Prediction: [0.36205907]
DT Test Single Actual Value: 0.3141377860228687
DT Percentage Error: 15.254862623142976 %
DT Test Single Mean Squared Error (MSE): 0.002296449815321817
```





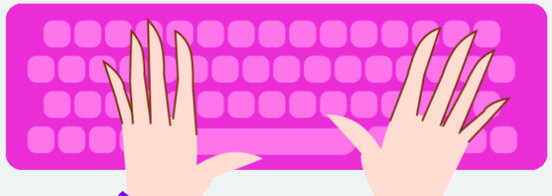
MACHINE L. ALGORITHMS

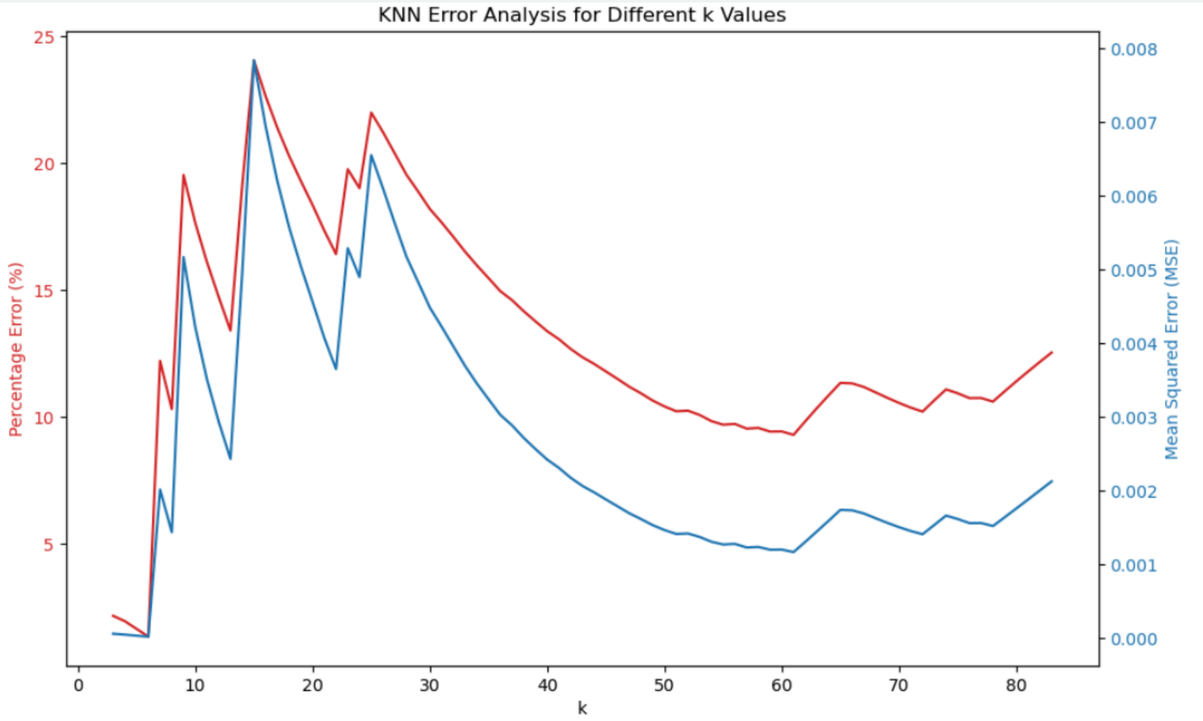
5

KNN

With Important Features

```
random_index = np.random.choice(range(X_test.shape[0]))
X_test_single = X_test[random_index].reshape(1, -1)
y_test_single = y_test[random_index]

X_train_top = X_train_df.loc[:, features]
X_train_top = X_train_df.loc[:, features]
```






```
Optimal k value based on minimum MSE: 6
Optimal k value based on minimum Percentage Error: 6
```

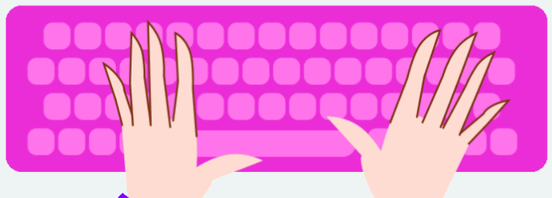
MACHINE L. ALGORITHMS

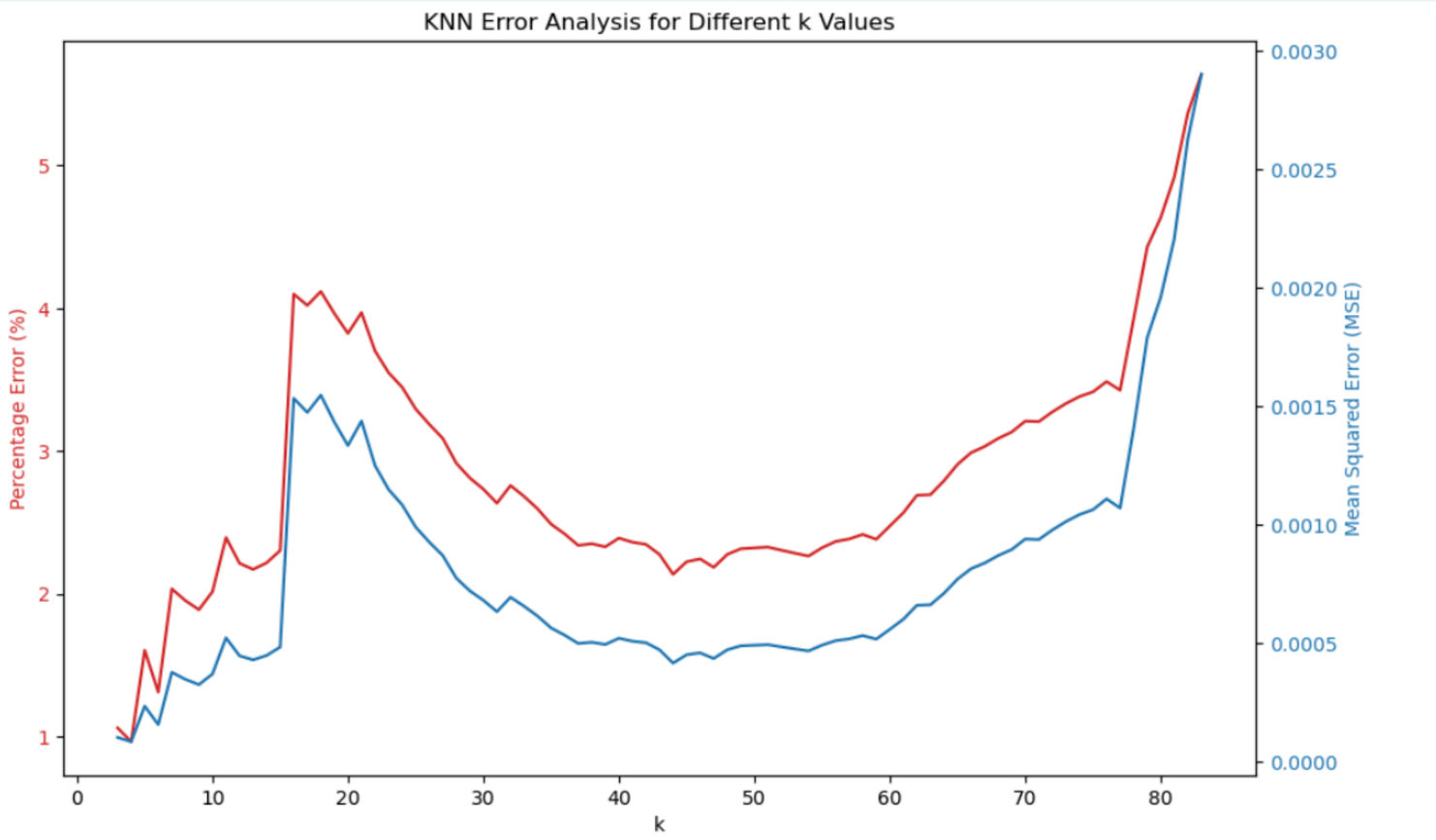
5

KNN

With All Features

```
random_index = np.random.choice(range(X_test.shape[0]))
X_test_single = X_test[random_index].reshape(1, -1)
y_test_single = y_test[random_index]

X_train_top = X_train_df
X_train_top = X_train_df
```





Ömer Aysal
Melisa Aslan
Ege Işık Altan
Devrim Bilgiç