F1 Winner Prediction

Team #1 – Tejas, Shravan, Arish, Rohan

Contents

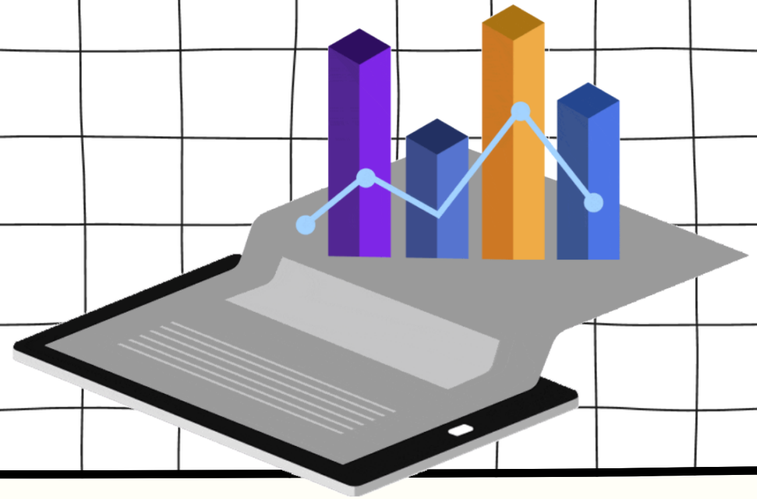
Introduction

1. Formula-1, overseen by FIA and owned by Formula One Group, is a globally followed single-seater auto racing pinnacle.
2. The sport demands comprehensive analysis of factors like weather, track characteristics, and historical performance influencing race outcomes.
3. This project aims to predict winners by considering these factors, utilizing classification algorithms such as Xgboost, KNN, Random Forest, Decision Tree, and Logistic Regression.
4. Algorithm accuracies are assessed to recommend the optimal one for predicting race outcomes.

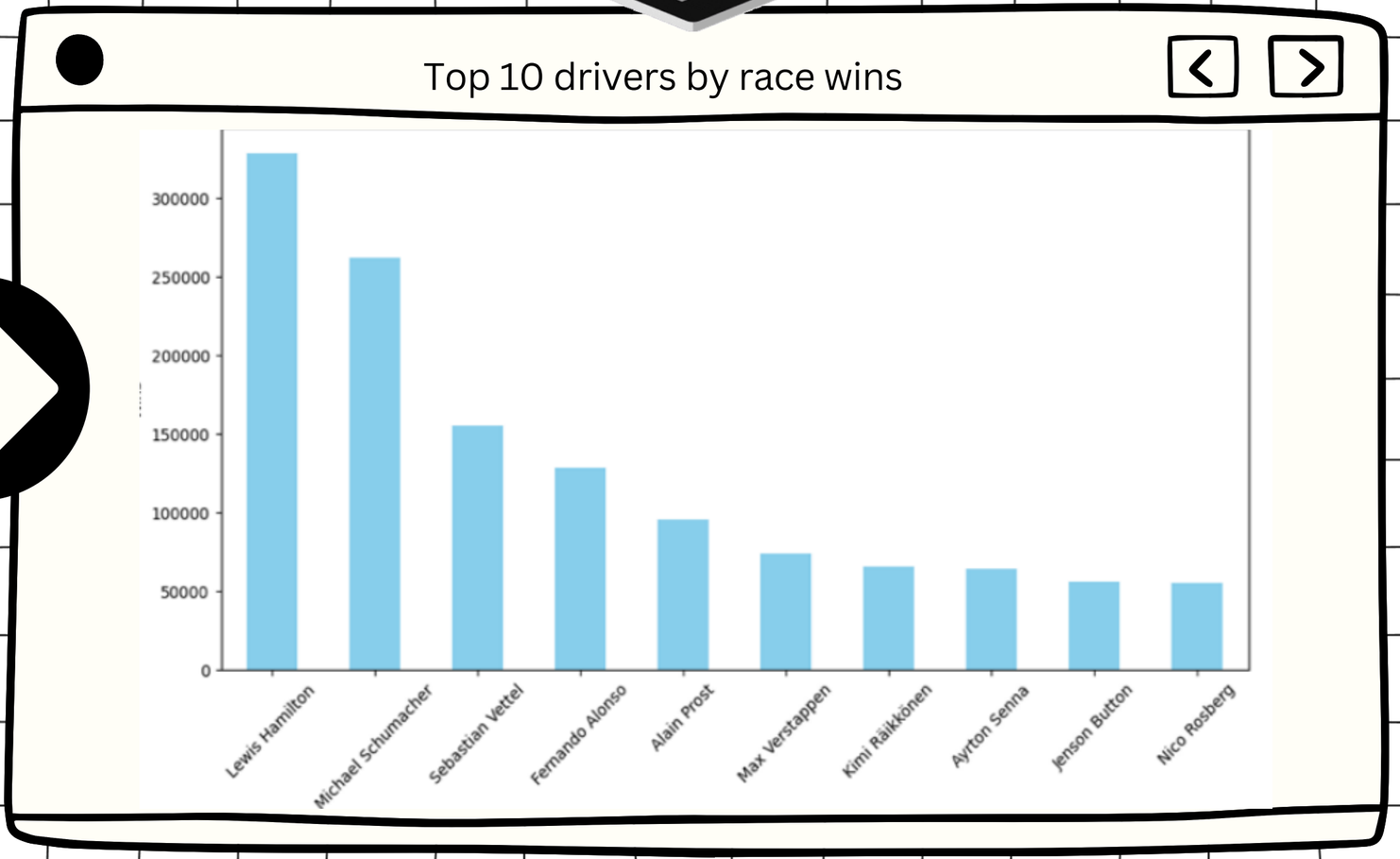
Motivation

- In sports, analysis of player performances and predicting winners is crucial for evaluating overall performance.
- It serves as a motivational tool, creating a strategic "Home Ground" advantage for teams with past victories.
- Media outlets leverage predictions to craft captivating narratives, emphasizing rivalries and adding excitement.
- Predictions, driven by sophisticated algorithms and data analysis, contribute to technological advancements in the sport, offering insights into driver skill, car performance, and track design.
- Predicting race outcomes propels F1 Esports and fantasy racing, allowing fans to test their predictive abilities and compete for virtual glory.

Dataset Overview



- The F1 World Championships and races dataset is from Kaggle, from 1950 to 2023.
- The dataset covers individual races, drivers, constructors, and circuits.
- 300k+ data points for training, validation, and testing models.
- There are 21 predictors used for model generation.
- The target label is the name of the driver who will be the predicted winner.



Data Pre-processing

01 Consolidated the data into a single data frame using pandas merge.

02 Handling null values.

03 Dropping unwanted columns.

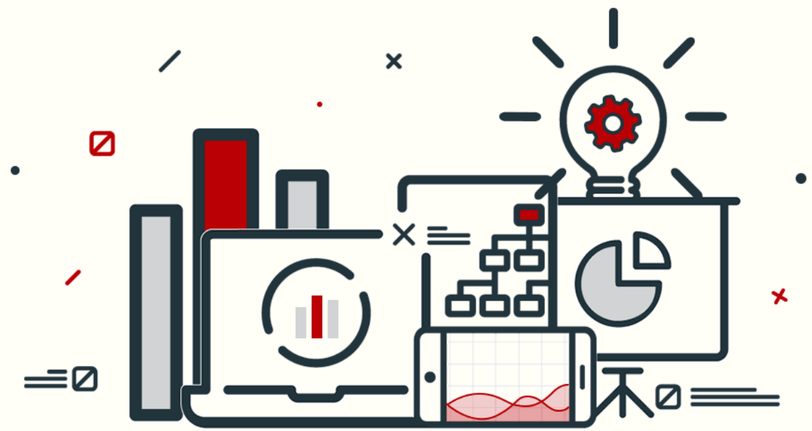
04 Renaming columns to meaningful names.

05 Converting to appropriate data types.

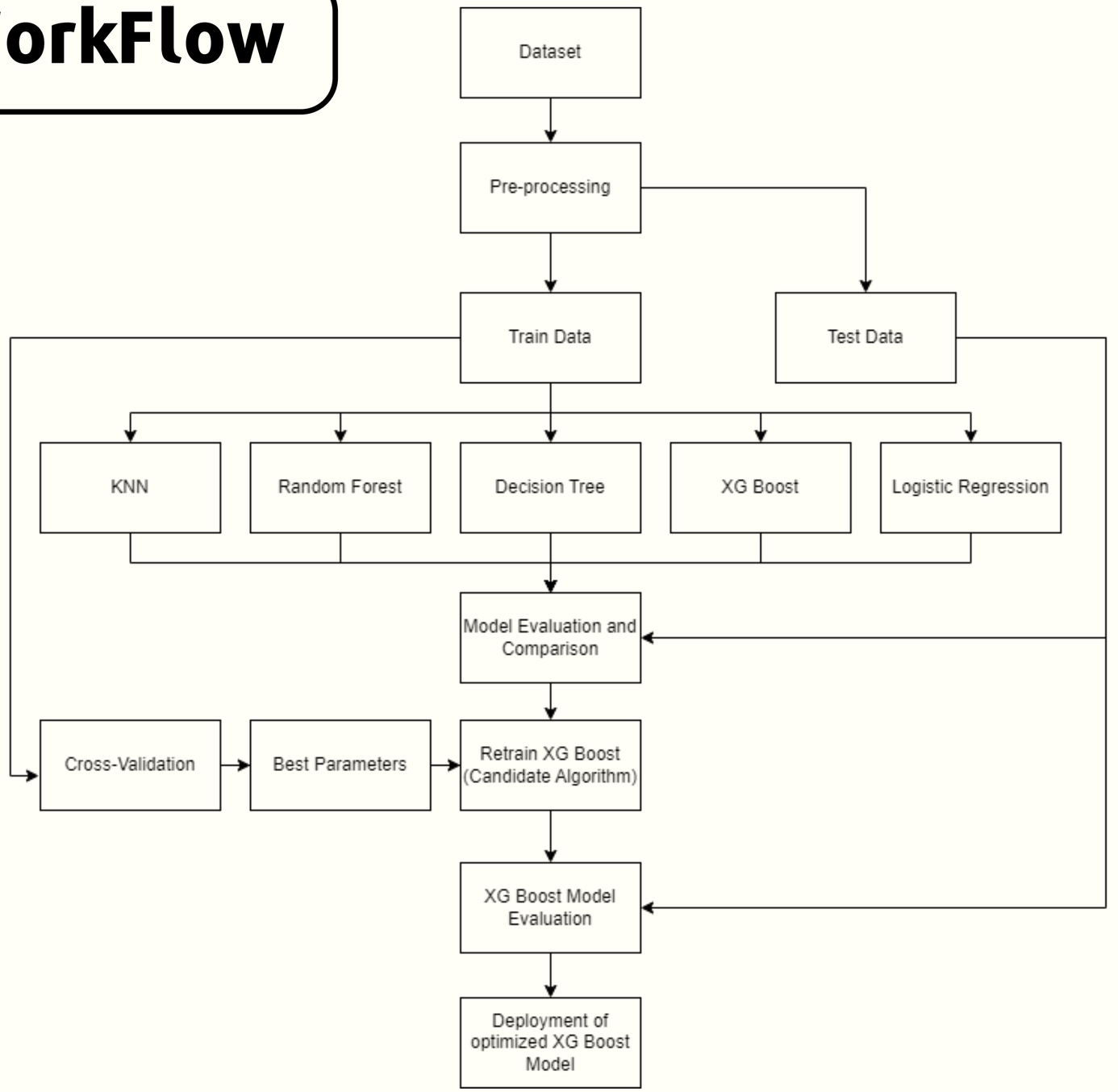
06 Calculating necessary columns.

07 Encoding categorical values using Label Encoding.

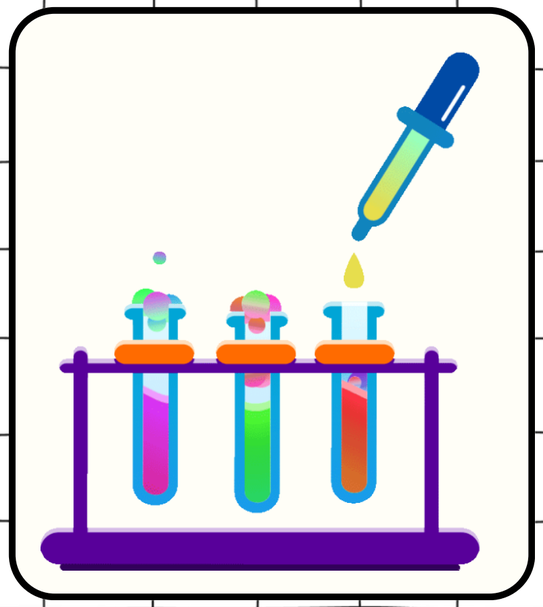
```
# merging all seperate dataframe into single dataframe

con1 = pd.merge(result_df, races_df, on ='raceId')
con2 = pd.merge(con1, drivers_df, on = 'driverId')
con3 = pd.merge(con2, driver_standings_df, on = 'driverId')
con4 = pd.merge(con3, constructor_df, on ='constructorId')
df = pd.merge(con4, stats_df, on ='statusId')
pd.get_option("display.max_columns",None)
df.head()
```



Project WorkFlow



Baseline Algorithms



Label Encoded Data

KNN
Non-parametric, proximity-based classification

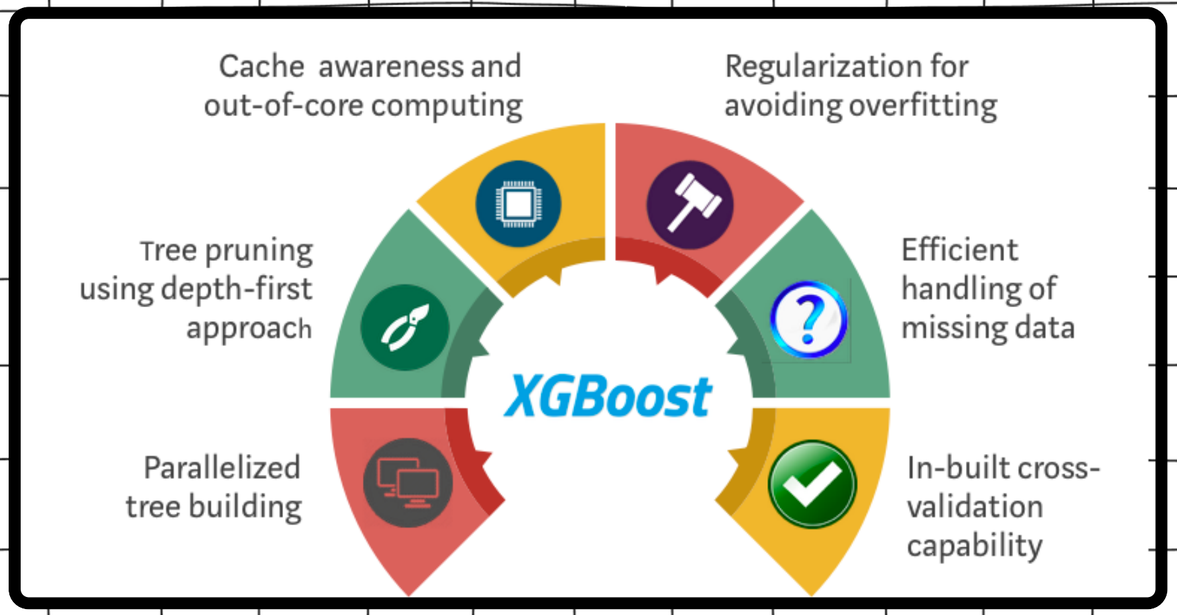
Logistic Regression
Linear model for binary classification

Random Forest
Ensemble of decision trees

Decision Tree
Tree structure for binary decision-making

Core Algorithm
XG Boost

WHY?




```python
import xgboost as xgb
from xgboost import XGBClassifier

xgb_model = XGBClassifier(objective='multi:softmax',
                          num_class=len(set(ytrain)),
                          max_depth=3, learning_rate=0.1)

# Train the XGBoost model
xgb_model.fit(xtrain, ytrain)

# Make predictions on the test set
y_predxg = xgb_model.predict(xtest)

# Calculate accuracy
accuracy = accuracy_score(ytest, y_predxg)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Hyper parameter tuning the XG Boost

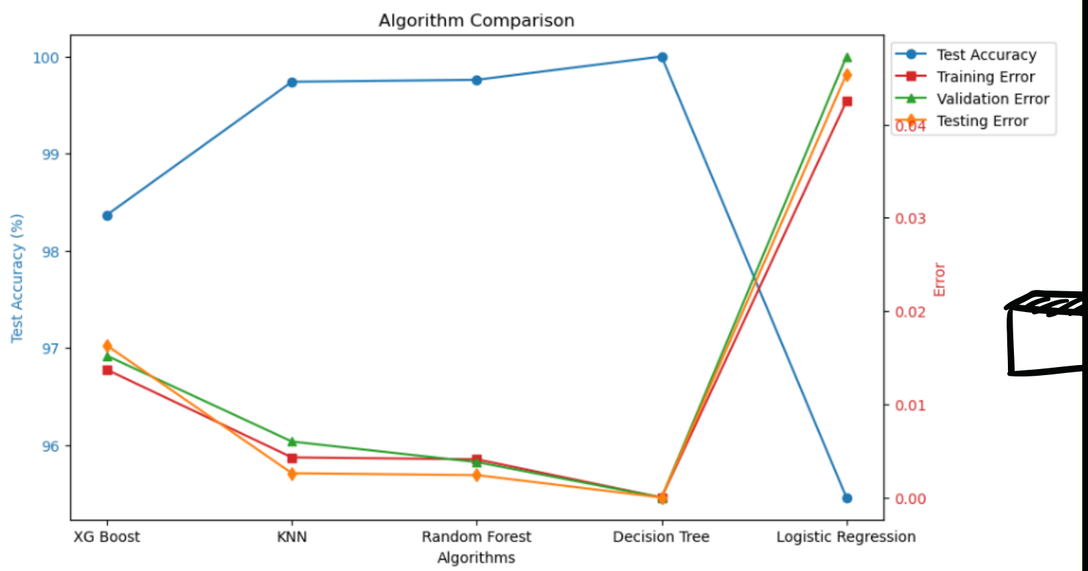
Why is it necessary?

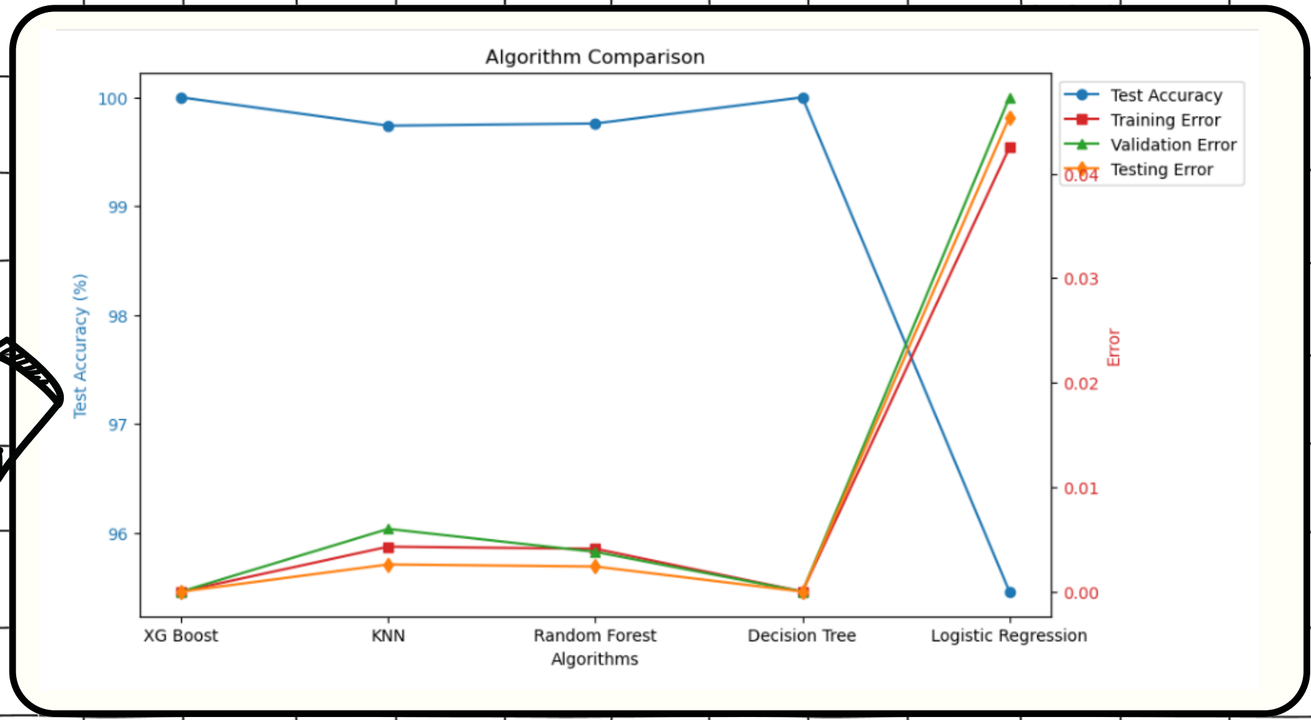
Tuning helps find the best combination, improving model accuracy, generalization, and preventing overfitting. It ensures the model is well-calibrated for the specific dataset, leading to better predictive capabilities.

How did we do it?

```python
from sklearn.model_selection import GridSearchCV

xgb_model = XGBClassifier(objective='multi:softmax',
                          num_class=len(set(ytrain)))

# Define the hyperparameter grid to search
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [50, 100, 200]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=xgb_model,
                           param_grid=param_grid, cv=3)

# Train the XGBoost model with hyperparameter tuning
grid_search.fit(xtrain, ytrain)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Make predictions on the test set using the best model
y_predxg = grid_search.predict(xtest)
```

Final results and performance comparison





Algorithm	Test Accuracy	Training error	Validation error	Testing Error
XG Boost	98.37%	0.0137	0.0152	0.0163
KNN	99.74%	0.0043	0.0060	0.0026
Random Forest	99.76%	0.0041	0.0038	0.0024
Decision Tree	100%	0.00	0.00	0.00
Logistic Regression	95.46%	0.0425	0.0473	0.0454

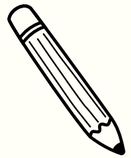
Algorithm	Test Accuracy	Training error	Validation error	Testing Error
XG Boost	**100 %**	**0.0000**	**0.0000**	**0.0000**
KNN	99.74 %	0.0043	0.0060	0.0026
Random Forest	99.76 %	0.0041	0.0038	0.0024
Decision Tree	100 %	0.0000	0.0000	0.0000
Logistic Regression	95.46 %	0.0425	0.0473	0.0454

Conclusions

Conclusion 01

Comparison of multiple algorithms to choose the best core algorithm as Xg boost with 100% test accuracy.

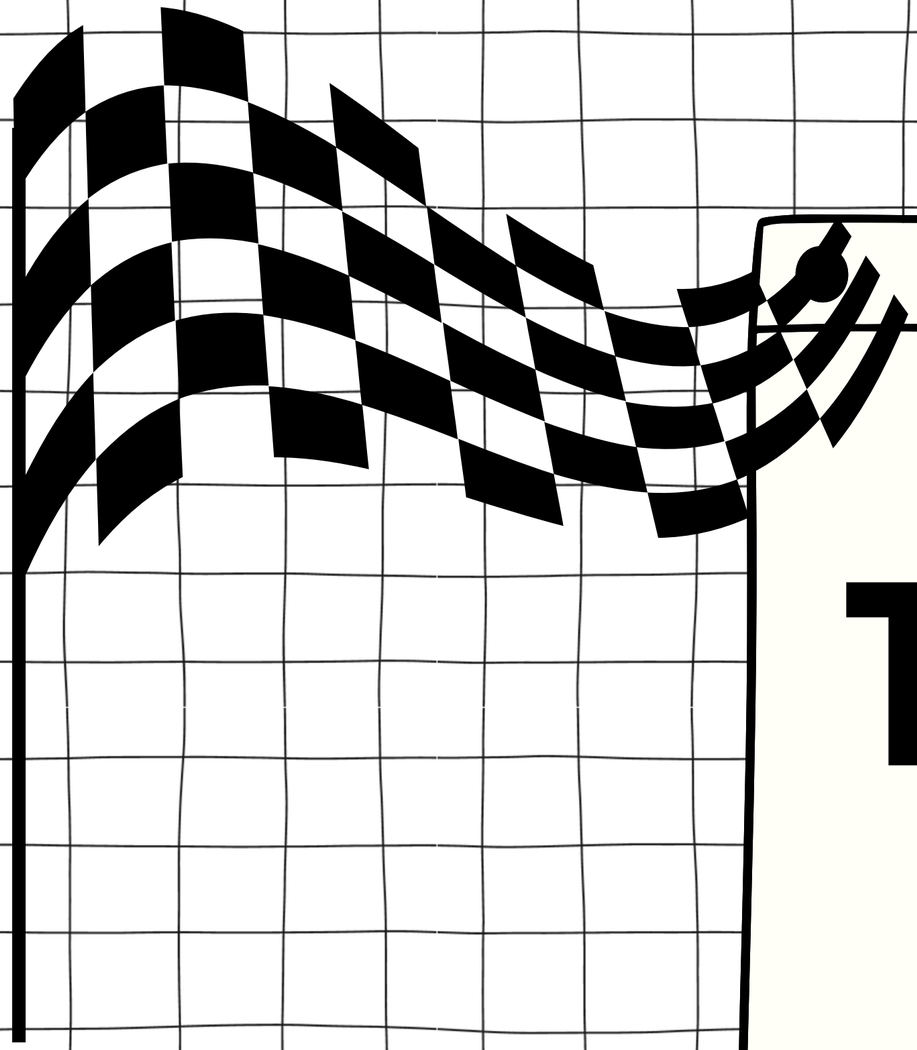
Conclusion 02



The testing results demonstrate the Xg boost model's promising abilities in tackling categorical prediction

Conclusion 03

Hyperparameter tuning, robust scalar and data cleansing contribute towards the perfectly trained model.



Thank you