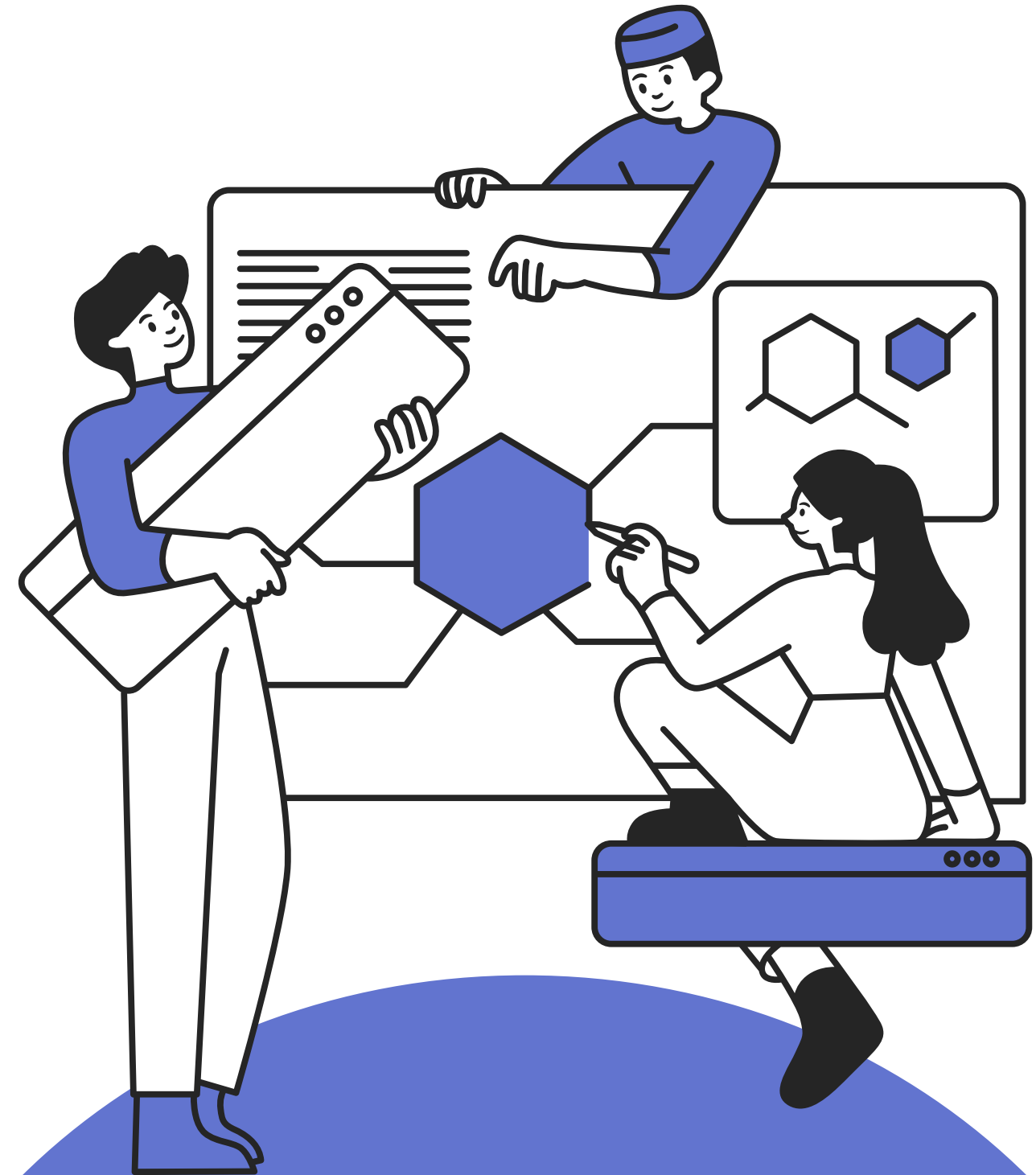


F1 등수 예측 머신러닝 프로젝트

서브웨이 (안태건, 이동재, 전유빈, 이영지)



CONTENTS

1 프로젝트 개요

2 데이터 수집

3 데이터 정제

4 모델 구축

5 결론

01

프로젝트 개요

주제명 : F1 등수 예측

프로젝트 주제 선정 배경

f1 그랑프리는 1950년부터 진행된 포뮬러 자동차 경기이다. 따라서 풍부한 데이터를 확보할 수 있다.

2024 f1 그랑프리는 2월 29일부터 12월 8일까지 총 24라운드에 걸쳐 진행된다. 따라서 지금까지의 데이터를 기반으로 남은 경기의 순위를 예측하는 프로젝트는 시의적절하다고 볼 수 있다.



02

데이터 수집

Kaggle을 통한 F1 데이터 수집

머신러닝 및 데이터 사이언스 커뮤니티인 Kaggle에 제공되어 있는 데이터 중 'Formula 1 World Championship(1950 - 2024)'의 데이터를 사용하였다.

해당 데이터에는 어떤 선수가 몇 등을 하였는지, 어떤 서킷에서 언제 하였는지 등이 다양하고 자세하게 수록되어 있다.

≡ kaggle

+ Create

🏠 Home

🏆 Competitions

📁 Datasets

🤖 Models

🔗 Code

🗨 Discussions

🎓 Learn

⌵ More

📅 View Active Events

🔍 Search

👤 VOPANI · UPDATED 3 MONTHS AGO

📈 1698

New Notebook

📄 Download (6 MB)

👤

⋮

Formula 1 World Championship (1950 - 2024)

F1 race data from 1950 to 2024

Data Card Code (114) Discussion (43) Suggestions (0)

About Dataset

Context

Formula 1 (a.k.a. F1 or Formula One) is the highest class of single-seater auto racing sanctioned by the Fédération Internationale de l'Automobile (FIA) and owned by the Formula One Group. The FIA Formula One World Championship has been one of the premier forms of racing around the world since its inaugural season in 1950. The word "formula" in the name refers to the set of rules to

Usability ⓘ
10.00

License
CC0: Public Domain

Expected update frequency
Monthly

circuits.csv (10.1 kB)

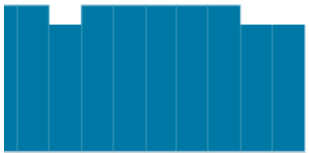
📄 🗨 >

Detail Compact Column

9 of 9 columns ⌵

About this file

Circuits where F1 races are held

circuitId	circuitRef	name	location
	Reference name of circuit	Actual name of circuit	City
	77 unique values	77 unique values	California Barcelona Other (73)

Data Explorer

Version 23 (21.49 MB)

📄 circuits.csv

📄 constructor_results.csv

📄 constructor_standings.csv

📄 constructors.csv

📄 driver_standings.csv

📄 drivers.csv

📄 lap_times.csv

📄 pit_stops.csv

📄 qualifying.csv

📄 races.csv

📄 results.csv

📄 seasons.csv

📄 sprint_results.csv

📄 status.csv

03

데이터 정제

변수 설명

starting_grid	각 드라이버가 서게 되는 출발 위치
laps	특정 레이스에서 차량이 서킷을 도는 수
total_laptime	드라이버가 레이스 동안 모든 랩을 완주하는 데 소요된 총 시간
fastestLapTime	한 드라이버가 서킷을 완주하는 데 걸린 시간 중 가장 짧은 시간
fastestLapSpeed	특정 드라이버가 기록한 가장 빠른 한 바퀴 주행 중의 평균 속도

03

데이터 정제

변수 설명

pitstop_count	<div>피트스톱 횟수</div> <div>** 피트스톱 : 레이스 중 차량이 피트 레인으로 들어와 타이어 교체, 간단한 정비 등을 수행하는 것</div>
pitstop_duration	차량이 피트 레인에 진입해서 다시 나갈 때까지 걸린 총 시간
average_quali	3번의 예선 세션 동안 드라이버가 기록한 랩타임의 평균
lastest_standing	해당 경기 이전까지의 합산 순위
result_position	최종 순위

03

데이터 정제

데이터 전처리

시간 데이터 처리	<ul style="list-style-type: none">• 시간 단위를 초 단위로 변환
DataFrame 생성	<ul style="list-style-type: none">• 2015년 이전 데이터 제거• raceId, driverId를 이용하여 하나의 DataFrame으로 병합
변수 처리	<ul style="list-style-type: none">• 결측치를 0으로 변환• q1, q2, q3 변수의 값을 세 변수의 평균값으로 대체• result_position 값을 3 이하는 1, 3 초과는 0으로 변환

03

데이터 정제

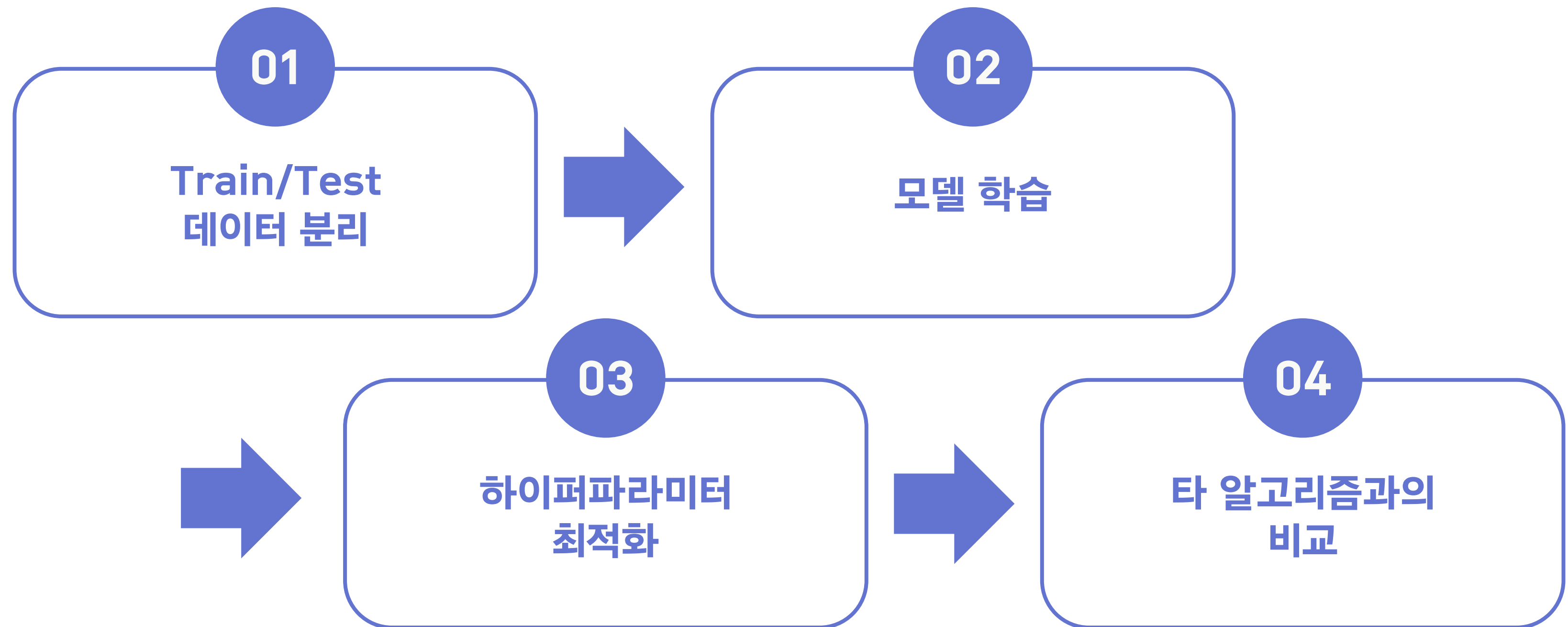
데이터 전처리 완료 후

	starting_grid	laps	total_laptime	fastestLapTime	fastestLapSpeed	pitstop_count	pitstop_duration	average_quali	lastest_standing	result_position
0	2	57	2895.600	90.6	210.815	2	18.155000	85.300000	3	1
1	1	57	2903.660	90.6	210.608	2	18.168333	84.600000	4	1
2	3	57	2905.243	90.0	212.235	2	18.155000	85.633333	2	1
3	8	57	2919.930	89.0	214.510	2	18.163333	86.033333	1	0
4	6	57	2954.579	92.3	206.861	2	18.163333	85.666667	9	0
...
1803	10	52	1410.677	89.7	236.401	1	29.341000	88.300000	7	0
1804	9	52	1415.487	89.7	236.380	2	29.410000	88.200000	8	0
1805	13	52	1426.403	90.2	235.041	2	29.658000	89.150000	14	0
1806	12	52	1436.060	90.0	235.713	2	29.444000	89.400000	12	0
1807	17	52	1437.253	90.1	235.396	2	29.884000	92.900000	13	0

1808 rows × 10 columns

04

모델 구축



04

모델 구축

Train/Test 데이터 분리

1. Train data 70% / Test data 30%로 분리
2. 정규화

```
# Train set / Test set 나누기
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.3, random_state=0)

# 데이터 컬럼 단위 정규화 하기
normalizer = StandardScaler()
X_train = normalizer.fit_transform(X_train)
X_test = normalizer.transform(X_test)

print(f"Train set dimension is {X_train.shape}")
print(f"Test set dimension is {X_test.shape}")
```

```
Train set dimension is (1265, 9)
Test set dimension is (543, 9)
```

04

모델 구축

모델 학습

1. Test set에 대한 초기 성능 확인 (정확도:0.8398)

```
rfc = RandomForestClassifier(n_estimators=50, random_state=0)
rfc.fit(X_train, y_train)
```

RandomForestClassifier

```
RandomForestClassifier(n_estimators=50, random_state=0)
```

```
# Train set에 대한 성능
y_pred = rfc.predict(X_train)
acc = accuracy_score(y_true = y_train, y_pred = y_pred)
print("Train set에 대한 성능")
print(f"정확도:{acc:0.4f}")

# Test set에 대한 성능
y_pred = rfc.predict(X_test)
acc = accuracy_score(y_true = y_test, y_pred = y_pred)
print("\n")
print("Test set에 대한 성능")
print(f"정확도:{acc:0.4f}")
```

Train set에 대한 성능
정확도:1.0000

Test set에 대한 성능
정확도:0.8398

04

모델 구축

하이퍼파라미터 최적화

1. 파라미터(n_estimators, max_depth, criterion, max_features) 최적화
2. GridSearchCV 모듈 사용

```
rfc = RandomForestClassifier(random_state=0)
param_grid = {
    "n_estimators" : [50,60,70],
    "max_depth" : [10,15,20],
    "criterion" : ["gini","entropy"],
    "max_features" : ["auto","sqrt","log2"]
}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=10, verbose=1, n_jobs=-1)
CV_rfc.fit(X_train,y_train)
```

04

모델 구축

하이퍼파라미터 최적화

1. 최적화 결과 정확도 향상(0.8398 -> 0.8398)

n_estimators : 50 | max_features : sqrt | criterion : entropy

```
# Train set에 대한 성능
y_pred = best_rfc.predict(X_train)
acc = accuracy_score(y_true = y_train, y_pred = y_pred)
print("Train set에 대한 성능")
print(f"정확도:{acc:0.4f}")

# Test set에 대한 성능
y_pred = best_rfc.predict(X_test)
acc = accuracy_score(y_true = y_test, y_pred = y_pred)
print("\n")
print("Test set에 대한 성능")
print(f"정확도:{acc:0.4f}")
```

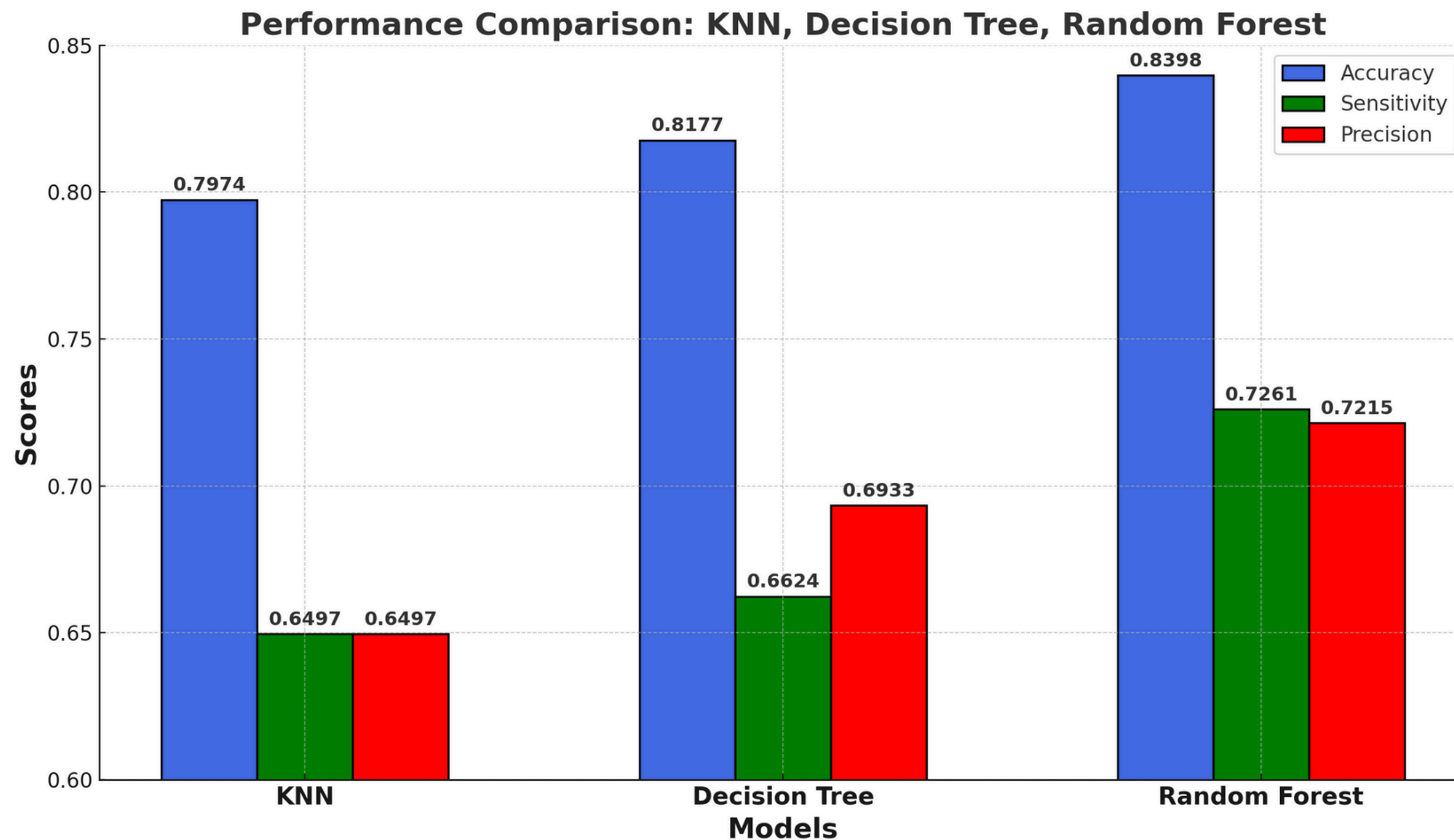
Train set에 대한 성능
정확도:1.0000

Test set에 대한 성능
정확도:0.8398

04

모델 구축

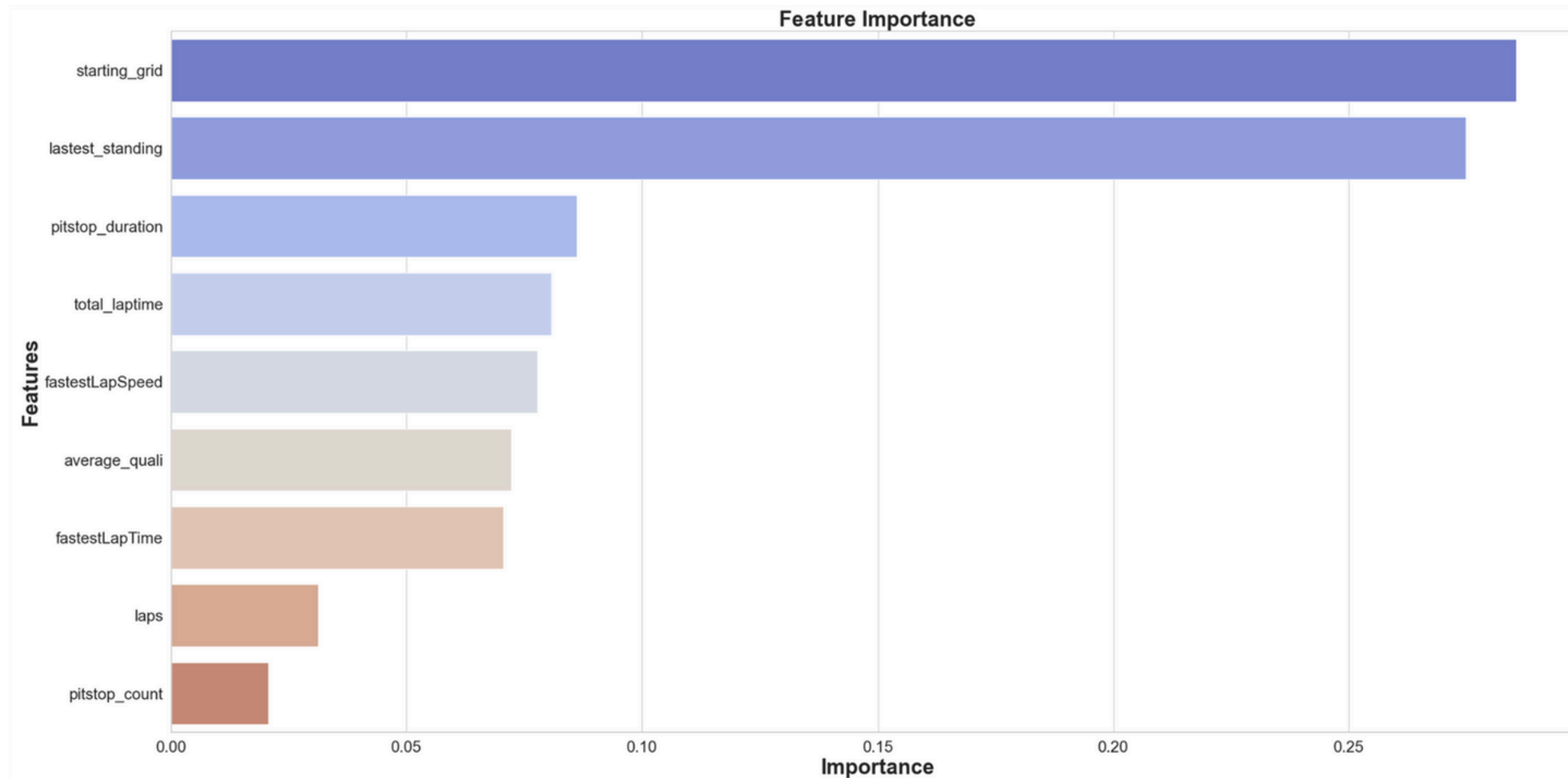
타 알고리즘과의 비교



05

결론

1. Random Forest를 이용한 모델의 정확도(0.8398)가 가장 높았다.
2. starting_grid 변수와 lastest_standing 변수의 예측 중요도가 가장 높았다.



THANK YOU

서브웨이 (안태건, 이동재, 전유빈, 이영지)