Step 1: Setup and File Upload

```
from google.colab import files
import pandas as pd
import os


# Upload all files
uploaded = files.upload()
```

Choose Files 25 files

- **Formula1_2013season_raceResults.csv**(text/csv) - 29063 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2014season_raceResults.csv**(text/csv) - 27951 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2015season_raceResults.csv**(text/csv) - 25619 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2016Season_RaceResults.csv**(text/csv) - 30991 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2017Season_RaceResults.csv**(text/csv) - 26584 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2018Season_RaceResults.csv**(text/csv) - 28785 bytes, last modified: 4/15/2025 - 100% done
- **formula1_2019season_drivers.csv**(text/csv) - 1939 bytes, last modified: 4/15/2025 - 100% done
- **formula1_2019season_raceResults.csv**(text/csv) - 27377 bytes, last modified: 4/15/2025 - 100% done
- **formula1_2019season_tracks.csv**(text/csv) - 2474 bytes, last modified: 4/15/2025 - 100% done
- **formula1_2020season_calendar.csv**(text/csv) - 2212 bytes, last modified: 4/15/2025 - 100% done
- **formula1_2020season_drivers.csv**(text/csv) - 2253 bytes, last modified: 4/15/2025 - 100% done
- **formula1_2020season_raceResults.csv**(text/csv) - 22034 bytes, last modified: 4/15/2025 - 100% done
- **formula1_2021season_calendar.csv**(text/csv) - 2845 bytes, last modified: 4/15/2025 - 100% done
- **formula1_2021season_drivers.csv**(text/csv) - 2114 bytes, last modified: 4/15/2025 - 100% done
- **formula1_2021season_raceResults.csv**(text/csv) - 32446 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2022season_calendar.csv**(text/csv) - 3234 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2022season_drivers.csv**(text/csv) - 2238 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2022season_raceResults.csv**(text/csv) - 32389 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2023season_calendar.csv**(text/csv) - 3382 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2023season_drivers.csv**(text/csv) - 2243 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2023season_raceResults.csv**(text/csv) - 32778 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2024season_calendar.csv**(text/csv) - 3534 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2024season_drivers.csv**(text/csv) - 2388 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2024season_raceResults.csv**(text/csv) - 35449 bytes, last modified: 4/15/2025 - 100% done
- **Formula1_2025Season_RaceResults.csv**(text/csv) - 3012 bytes, last modified: 4/15/2025 - 100% done

```
Saving Formula1_2013season_raceResults.csv to Formula1_2013season_raceResults (1).csv
Saving Formula1_2014season_raceResults.csv to Formula1_2014season_raceResults (1).csv
Saving Formula1_2015season_raceResults.csv to Formula1_2015season_raceResults (1).csv
Saving Formula1_2016Season_RaceResults.csv to Formula1_2016Season_RaceResults (1).csv
Saving Formula1_2017Season_RaceResults.csv to Formula1_2017Season_RaceResults (1).csv
Saving Formula1_2018Season_RaceResults.csv to Formula1_2018Season_RaceResults (1).csv
Saving formula1_2019season_drivers.csv to formula1_2019season_drivers (1).csv
Saving formula1_2019season_raceResults.csv to formula1_2019season_raceResults (1).csv
Saving formula1_2019season_tracks.csv to formula1_2019season_tracks (1).csv
Saving formula1_2020season_calendar.csv to formula1_2020season_calendar (1).csv
Saving formula1_2020season_drivers.csv to formula1_2020season_drivers (1).csv
Saving formula1_2020season_raceResults.csv to formula1_2020season_raceResults (1).csv
Saving formula1_2021season_calendar.csv to formula1_2021season_calendar (1).csv
Saving formula1_2021season_drivers.csv to formula1_2021season_drivers (1).csv
Saving formula1_2021season_raceResults.csv to formula1_2021season_raceResults (1).csv
Saving Formula1_2022season_calendar.csv to Formula1_2022season_calendar (1).csv
Saving Formula1_2022season_drivers.csv to Formula1_2022season_drivers (1).csv
Saving Formula1_2022season_raceResults.csv to Formula1_2022season_raceResults (1).csv
Saving Formula1_2023season_calendar.csv to Formula1_2023season_calendar (1).csv
Saving Formula1_2023season_drivers.csv to Formula1_2023season_drivers (1).csv
Saving Formula1_2023season_raceResults.csv to Formula1_2023season_raceResults (1).csv
Saving Formula1_2024season_calendar.csv to Formula1_2024season_calendar (1).csv
Saving Formula1_2024season_drivers.csv to Formula1_2024season_drivers (1).csv
Saving Formula1_2024season_raceResults.csv to Formula1_2024season_raceResults (1).csv
Saving Formula1_2025Season_RaceResults.csv to Formula1_2025Season_RaceResults (1).csv
```

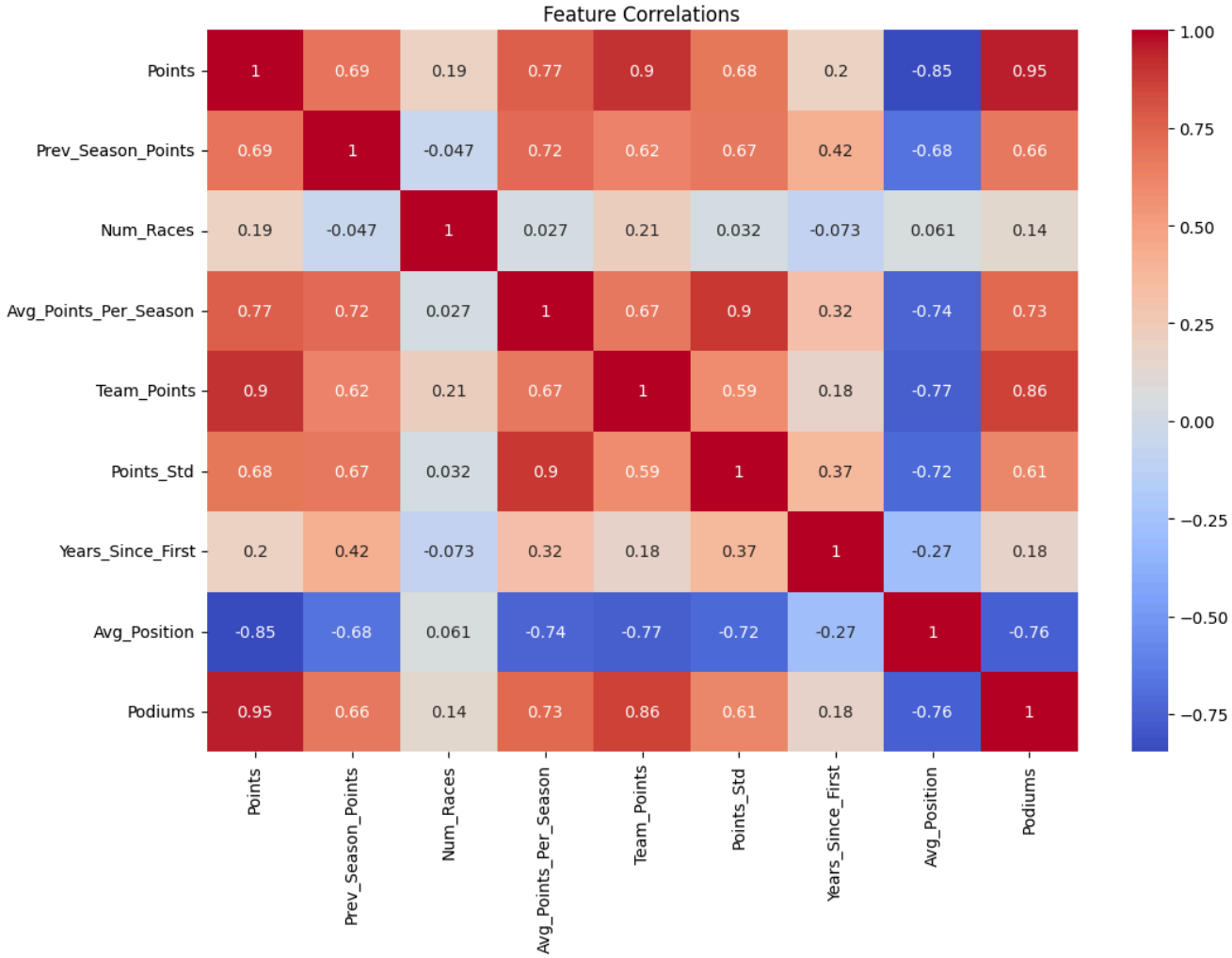
+ Code + Text

```
# File lists
race_results_files = [
    'Formula1_2013season_raceResults.csv', 'Formula1_2014season_raceResults.csv',
    'Formula1_2015season_raceResults.csv', 'Formula1_2016Season_RaceResults.csv',
    'Formula1_2017Season_RaceResults.csv', 'Formula1_2018Season_RaceResults.csv',
    'formula1_2019season_raceResults.csv', 'formula1_2020season_raceResults.csv',
    'formula1_2021season_raceResults.csv', 'Formula1_2022season_raceResults.csv',
    'Formula1_2023season_raceResults.csv', 'Formula1_2024season_raceResults.csv',
    'Formula1_2025Season_RaceResults.csv'
]
drivers_files = [
    'formula1_2019season_drivers.csv', 'formula1_2020season_drivers.csv',
    'formula1_2021season_drivers.csv', 'Formula1_2022season_drivers.csv',
    'Formula1_2023season_drivers.csv', 'Formula1_2024season_drivers.csv'
]
calendar_files = [
    'formula1_2019season_tracks.csv', 'formula1_2020season_calendar.csv',
    'formula1_2021season_calendar.csv', 'Formula1_2022season_calendar.csv',
    'Formula1_2023season_calendar.csv', 'Formula1_2024season_calendar.csv'
]
```

Step 2: Load and Combine Datasets

```
race_dfs = []
for file in race_results_files:
    df = pd.read_csv(file)
    if 'Formula1' in file:
        season = file.split('_')[1].split('Season')[0] if 'Season' in file else file.split('_')[1][:4]
    else:
        season = file.split('_')[1][:4]
    df['Season'] = int(season)
    race_dfs.append(df)

race_results_df = pd.concat(race_dfs, ignore_index=True)
drivers_dfs = [pd.read_csv(file) for file in drivers_files]
drivers_df = pd.concat(drivers_dfs, ignore_index=True).drop_duplicates(subset=['Driver'])
calendar_dfs = [pd.read_csv(file) for file in calendar_files]
calendar_df = pd.concat(calendar_dfs, ignore_index=True)
race_results_df['Track'] = race_results_df['Track'].str.strip()
calendar_df['Circuit Name'] = calendar_df['Circuit Name'].str.strip()


# Step 3: Preprocessing with 9 Features
race_results_df['Points'] = race_results_df['Points'].fillna(0).astype(float)
race_results_df = race_results_df[race_results_df['Driver'].notna()]
race_results_df['Position'] = pd.to_numeric(race_results_df['Position'], errors='coerce')

season_points_df = race_results_df.groupby(['Season', 'Driver'])['Points'].sum().reset_index()
season_points_df['Prev_Season_Points'] = season_points_df.groupby('Driver')['Points'].shift(1).fillna(0)

races_per_season = race_results_df.groupby('Season')['Track'].nunique().reset_index()
races_per_season.columns = ['Season', 'Num_Races']
season_points_df = season_points_df.merge(races_per_season, on='Season', how='left')

avg_points = season_points_df.groupby('Driver')['Points'].mean().reset_index()
avg_points.columns = ['Driver', 'Avg_Points_Per_Season']
season_points_df = season_points_df.merge(avg_points, on='Driver', how='left')

team_points = race_results_df.groupby(['Season', 'Team'])['Points'].sum().reset_index()
team_points.columns = ['Season', 'Team', 'Team_Points']
race_results_df = race_results_df.merge(team_points, on=['Season', 'Team'], how='left')
season_points_df = season_points_df.merge(
    race_results_df[['Season', 'Driver', 'Team_Points']].drop_duplicates(),
    on=['Season', 'Driver'],
    how='left'
)

consistency = season_points_df.groupby('Driver')['Points'].std().reset_index()
consistency.columns = ['Driver', 'Points_Std']
season_points_df = season_points_df.merge(consistency, on='Driver', how='left').fillna({'Points_Std': 0})

season_points_df['Years_Since_First'] = season_points_df['Season'] - season_points_df.groupby('Driver')['Season'].transform('min')

avg_position = race_results_df.groupby(['Season', 'Driver'])['Position'].mean().reset_index()
avg_position.columns = ['Season', 'Driver', 'Avg_Position']
season_points_df = season_points_df.merge(avg_position, on=['Season', 'Driver'], how='left')

podiums = race_results_df[race_results_df['Position'].isin([1, 2, 3])].groupby(['Season', 'Driver']).size().reset_index(name='Podiums')
season_points_df = season_points_df.merge(podiums, on=['Season', 'Driver'], how='left').fillna({'Podiums': 0})

from sklearn.preprocessing import LabelEncoder
le_driver = LabelEncoder()
season_points_df['Driver_Encoded'] = le_driver.fit_transform(season_points_df['Driver'])


# Step 4: Basic EDA (Optional, for verification)
import matplotlib.pyplot as plt
import seaborn as sns
print(season_points_df.describe())
plt.figure(figsize=(12, 8))
sns.heatmap(season_points_df[['Points', 'Prev_Season_Points', 'Num_Races', 'Avg_Points_Per_Season', 'Team_Points', 'Points_Std', 'Years_
plt.title('Feature Correlations')
plt.show()
```

```
            Season      Points  Prev_Season_Points   Num_Races  \
count   304.000000  304.000000          304.000000  304.000000
mean   2018.828947   86.939145           83.228618   19.500000
std       3.735966  110.891863          112.400625    4.984464
min    2013.000000    0.000000            0.000000    2.000000
25%    2016.000000    4.000000            0.000000   19.000000
50%    2019.000000   37.000000           30.500000   21.000000
75%    2022.000000  134.500000          125.250000   22.000000
max    2025.000000  530.000000          530.000000   24.000000

       Avg_Points_Per_Season  Team_Points  Points_Std  Years_Since_First  \
count             304.000000   304.000000  304.000000         304.000000
mean               87.798601   179.315789   58.558812           3.335526
std                85.992839   215.604057   46.823223           3.217001
min                 0.000000     0.000000    0.000000           0.000000
25%                17.400000    22.000000   14.730920           1.000000
50%                49.555556    74.500000   41.872041           2.000000
75%               124.916667   290.000000   93.789114           5.250000
max               300.653846   790.000000  151.753906          12.000000

       Avg_Position     Podiums  Driver_Encoded
count    299.000000  304.000000      304.000000
mean       9.660204    2.539474       33.763158
std        4.201066    4.618633       19.900159
min        1.272727    0.000000        0.000000
25%        6.550000    0.000000       14.750000
50%       10.000000    0.000000       34.500000
75%       12.916667    2.000000       51.000000
max       18.222222   21.000000       66.000000
```




```
# Step 5: Model Building with XGBoost
from sklearn.model_selection import train_test_split, cross_val_score
from xgboost import XGBRegressor
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import mean_squared_error

# Define 9 features
features = ['Driver_Encoded', 'Prev_Season_Points', 'Num_Races', 'Avg_Points_Per_Season', 'Team_Points', 'Points_Std', 'Years_Since_Fi
X = season_points_df[features]
y = season_points_df['Points']
```

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features with RobustScaler
scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Train XGBoost
xgb_model = XGBRegressor(
    n_estimators=200,
    max_depth=5,
    learning_rate=0.05,
    subsample=0.8,
    random_state=42
)
xgb_model.fit(X_train_scaled, y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.05, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=5, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=200, n_jobs=None,
             num_parallel_tree=None, random_state=42, ...)
```

```
# Predict and evaluate
y_pred = xgb_model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
print(f'MSE: {mse:.2f}')
```

```
MSE: 339.85
```

```
# Cross-validation
cv_scores = cross_val_score(xgb_model, scaler.transform(X), y, cv=5, scoring='neg_mean_squared_error')
print(f'Cross-Validated MSE: {-cv_scores.mean():.2f} (+/- {cv_scores.std() * 2:.2f})')
```

```
Cross-Validated MSE: 698.39 (+/- 1404.95)
```

```
# Save artifacts
import pickle
with open('driver_encoder.pkl', 'wb') as f:
    pickle.dump(le_driver, f)
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
with open('points_model.pkl', 'wb') as f:
    pickle.dump(xgb_model, f)

# Download files
files.download('driver_encoder.pkl')
files.download('scaler.pkl')
files.download('points_model.pkl')
```

```
Start coding or generate with AI.
```