Final Projet

Nelson Maloney

Table of Contents

Data Collection

Data Collection – Fast F1

- Fast F1 is an API that is built on top of Pandas DataFrames and Series
- Fast F1 has lots of data (laptime information, weather data, car telemetry, etc.)
- All of Fast F1's data is downloaded from 2 sources:
  - The official F1 data stream
  - Ergast web api
- Fast F1 is very flexible, precise, and has its own built in functions, such as pick_fastest() (get fastest lap)

Data Origins

- Fast F1 API to load in the data
  - Qualification and practice lap data
  - Weather data
- Data from every race from 2018 to 2021
- Data loaded by session per race per year

	Time	DriverNumber	LapTime	LapNumber	Stint	PitOutTime	PitInTime	Sector1Time	Sector2Time	Sector3Time	Sector1SessionTime	Sector2SessionTime	Sector3SessionTime	SpeedI1	SpeedI2	SpeedFL	SpeedST	IsPersonalBest	Compound	TyreLife	FreshTyre	LapStartTime
0	0 days 00:53:23.995000	2	0 days 00:01:34.233000	13.0	3.0	NaT	NaT	0 days 00:00:30.878000	0 days 00:00:25.279000	0 days 00:00:38.076000	0 days 00:52:20.640000	0 days 00:52:45.919000	0 days 00:53:23.995000	261.0	279.0	287.0	251.0	True	INTERMEDIATE	13.0	False	0 days 00:51:49.762000
1	0 days 00:40:45.211000	44	0 days 00:01:34.225000	4.0	1.0	NaT	NaT	0 days 00:00:30.848000	0 days 00:00:25.401000	0 days 00:00:37.976000	0 days 00:39:41.834000	0 days 00:40:07.235000	0 days 00:40:45.211000	268.0	286.0	290.0	236.0	True	INTERMEDIATE	4.0	True	0 days 00:39:10.986000
2	0 days 01:00:37.407000	35	0 days 00:01:35.589000	15.0	3.0	NaT	NaT	0 days 00:00:31.687000	0 days 00:00:25.439000	0 days 00:00:38.463000	0 days 00:59:33.505000	0 days 00:59:58.944000	0 days 01:00:37.407000	262.0	280.0	282.0	257.0	True	INTERMEDIATE	13.0	False	0 days 00:59:01.818000
3	0 days 01:03:19.272000	33	0 days 00:01:31.680000	7.0	2.0	NaT	NaT	0 days 00:00:30.049000	0 days 00:00:24.535000	0 days 00:00:37.096000	0 days 01:02:17.641000	0 days 01:02:42.176000	0 days 01:03:19.272000	269.0	285.0	289.0	251.0	True	INTERMEDIATE	8.0	False	0 days 01:01:47.592000
4	0 days 00:42:46.913000	28	0 days 00:01:35.438000	13.0	3.0	NaT	NaT	0 days 00:00:31.316000	0 days 00:00:25.559000	0 days 00:00:38.563000	0 days 00:41:42.791000	0 days 00:42:08.350000	0 days 00:42:46.913000	264.0	281.0	287.0	259.0	True	INTERMEDIATE	11.0	True	0 days 00:41:11.475000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1404	0 days 00:54:24.139000	6	0 days 00:01:25.322000	10.0	3.0	NaT	NaT	0 days 00:00:17.354000	0 days 00:00:36.801000	0 days 00:00:31.167000	0 days 00:53:16.171000	0 days 00:53:52.972000	0 days 00:54:24.139000	287.0	315.0	221.0	324.0	True	SOFT	3.0	True	0 days 00:52:58.817000
1405	0 days 01:05:29.762000	5	0 days 00:01:25.115000	17.0	4.0	NaT	NaT	0 days 00:00:17.450000	0 days 00:00:36.530000	0 days 00:00:31.135000	0 days 01:04:22.097000	0 days 01:04:58.627000	0 days 01:05:29.762000	285.0	314.0	222.0	321.0	True	SOFT	3.0	True	0 days 01:04:04.647000
1406	0 days 01:06:03.233000	4	0 days 00:01:24.106000	12.0	3.0	NaT	NaT	0 days 00:00:17.201000	0 days 00:00:36.345000	0 days 00:00:30.560000	0 days 01:04:56.328000	0 days 01:05:32.673000	0 days 01:06:03.233000	284.0	307.0	223.0	320.0	True	SOFT	3.0	True	0 days 01:04:39.127000
1407	0 days 00:39:45.771000	77	0 days 00:01:24.025000	9.0	3.0	NaT	NaT	0 days 00:00:17.191000	0 days 00:00:36.371000	0 days 00:00:30.463000	0 days 00:38:38.937000	0 days 00:39:15.308000	0 days 00:39:45.771000	288.0	316.0	218.0	321.0	True	SOFT	9.0	False	0 days 00:38:21.746000
1408	0 days 00:59:10.856000	99	0 days 00:01:25.048000	10.0	2.0	NaT	NaT	0 days 00:00:17.427000	0 days 00:00:36.646000	0 days 00:00:30.975000	0 days 00:58:03.235000	0 days 00:58:39.881000	0 days 00:59:10.856000	285.0	316.0	216.0	326.0	True	SOFT	3.0	True	0 days 00:57:45.808000

Data Prep

```python
def create_laps(weather_data, session_data):
    '''
    Creates fastest laps dataframe with individual session data per driver. Data passed as arguments must be passed as a list of dataframes.
    '''
    concat_data = []
    fast_laps = []
    q_laps = []
    # Concatenating the weather and laps data and adding location and year data to identify our sessions
    for session in session_data:
        for weather in weather_data:
            weather.reset_index(inplace=True, drop=True)
            # weather.drop('Time', inplace=True, axis=1)
            lw = pd.concat([session.laps, weather], axis=1)
            lw['Location'] = session.event.Location
            lw['Year'] = session.event.EventDate.year
        concat_data.append(lw)
    # Creating a dataframe for the fastest laps per session per driver
    for session in concat_data:
        for driver in pd.unique(session.Driver):
            fast_laps.append(session.pick_driver(driver).pick_fastest())
    fast_laps = pd.DataFrame(fast_laps)

    return fast_laps
```

Data Cleaning

```python
# Automated processing helper function from previous project notebook.
def laps_processing(session):
    '''
    Automating data preprocessing, based on step by step preprocessing for FP1. All sessions share the same columns.
    '''
    # Used lap number for tyre life
    session.TyreLife.fillna(session.LapNumber, inplace=True)
    # Missing speed trap data from various sectors were sparse so just used mean to fill in missing values
    session.SpeedI1.fillna(session.SpeedI1.mean(), inplace=True)
    session.SpeedI2.fillna(session.SpeedI2.mean(), inplace=True)
    session.SpeedST.fillna(session.SpeedST.mean(), inplace=True)
    session.SpeedFL.fillna(session.SpeedFL.mean(), inplace=True)
    # Filling in the rest of missing timing data to 0 to indicate sensor failures and laptime deletions which happen when a driver exceeds track limits.
    session.LapTime.fillna(timedelta(0), inplace=True)
    session.Sector1Time.fillna(timedelta(0), inplace=True)
    session.Sector2Time.fillna(timedelta(0), inplace=True)
    session.Sector3Time.fillna(timedelta(0), inplace=True)
    # Turning the compounds into numbers with .replace, '' entries were only found during one race in 2021, the compound rules were different in 2021, determined fastest laps were set on SOFT tyres.
    session.Compound.replace({'HYPERSOFT':1, 'ULTRASOFT':2, 'SUPERSOFT':3, 'SOFT':4, 'MEDIUM':5, 'HARD':6, 'TEST':10, 'INTERMEDIATE':8, 'WET':9, 'UNKNOWN':0, 'C':11, '':4}, inplace=True)
    # Dropping unnecessary timimng information. Determined FreshTyre entries were not accurate.
    session.drop(['Time', 'DriverNumber', 'PitInTime', 'PitOutTime', 'Sector1SessionTime', 'Sector2SessionTime', 'Sector3SessionTime', 'FreshTyre', 'LapStartTime', 'LapStartDate', 'Rainfall', 'TrackStatus', 'Team'], inplace=True, axis=1)
    # Turning the timedelta objects into seconds while retaining millisecond precision.
    session.LapTime = session.LapTime.dt.total_seconds()
    session.Sector1Time = session.Sector1Time.dt.total_seconds()
    session.Sector2Time = session.Sector2Time.dt.total_seconds()
    session.Sector3Time = session.Sector3Time.dt.total_seconds()
    return session
```

Result

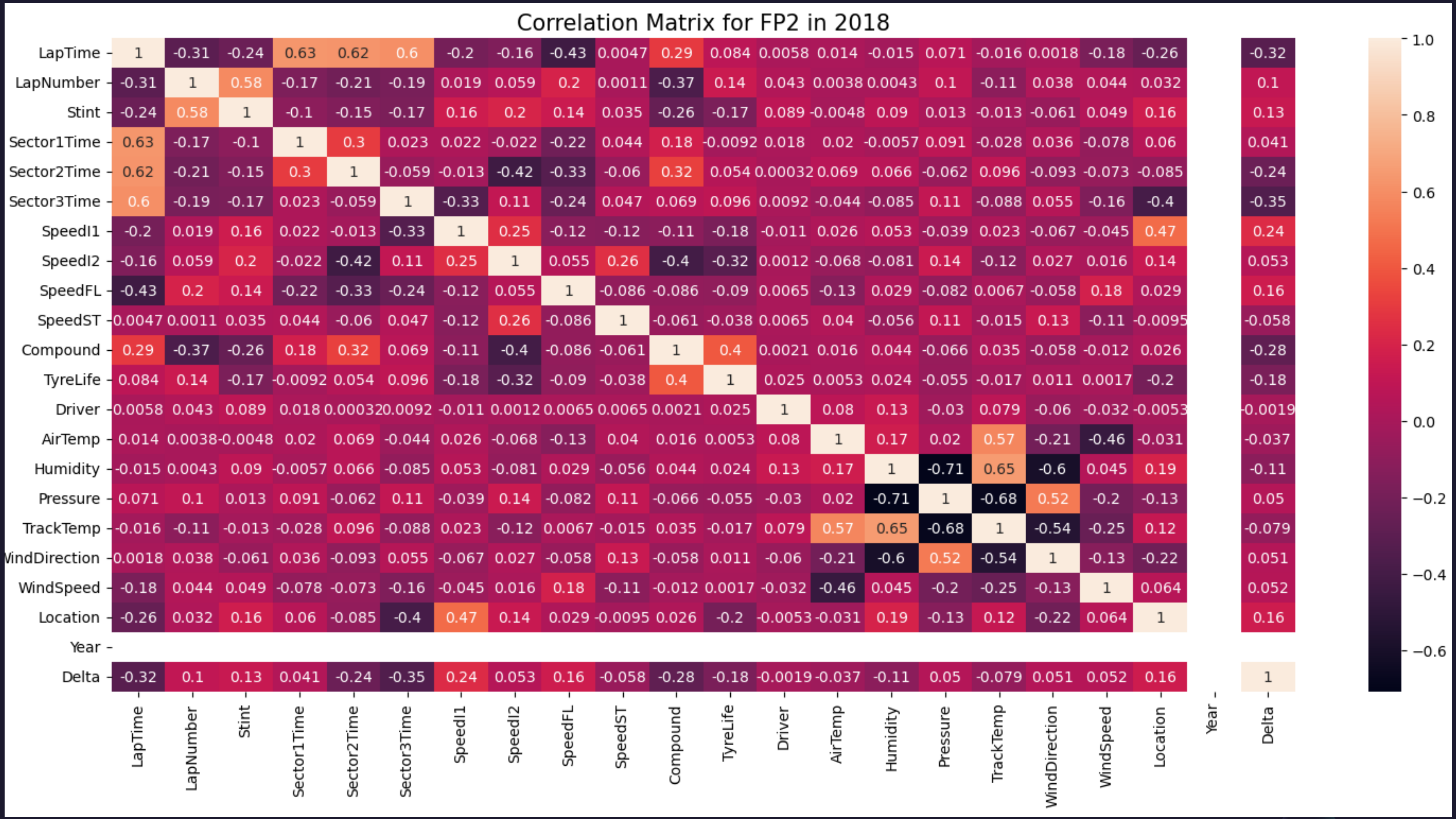
	LapTime	LapNumber	Stint	Sector1Time	Sector2Time	Sector3Time	SpeedI1	SpeedI2	SpeedFL	SpeedST	IsPersonalBest	Compound	TyreLife	Driver	IsAccurate	AirTemp	Humidity	Pressure	TrackTemp	WindDirection	WindSpeed	Location	Year
0	81.164	20.0	6.0	26.698	22.066	32.400	284.0	293.0	302.0	218.0	True	2	3.0	HAM	True	25.5	54.8	1018.5	29.9	290.0	0.3	Melbourne	2018.0
1	81.828	16.0	6.0	26.992	22.204	32.632	285.0	295.0	305.0	213.0	True	2	3.0	RAI	True	24.5	62.2	1018.8	27.8	330.0	0.2	Melbourne	2018.0
2	81.838	19.0	6.0	27.018	22.129	32.691	284.0	295.0	306.0	226.0	True	2	3.0	VET	True	24.5	62.6	1018.8	27.8	277.0	0.2	Melbourne	2018.0
3	81.879	17.0	6.0	26.971	22.241	32.667	279.0	288.0	298.0	257.0	True	2	3.0	VER	True	24.8	59.7	1018.7	28.6	291.0	0.3	Melbourne	2018.0
4	82.152	16.0	6.0	27.240	22.448	32.464	277.0	288.0	296.0	220.0	True	2	3.0	RIC	True	25.2	56.3	1018.6	29.3	289.0	0.4	Melbourne	2018.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1532	84.338	7.0	2.0	17.281	36.465	30.592	289.0	318.0	222.0	322.0	True	4	3.0	LAT	True	25.2	55.5	1018.5	29.3	294.0	0.4	Abu Dhabi	2021.0
1533	84.423	4.0	1.0	17.130	36.527	30.766	290.0	319.0	222.0	323.0	True	4	4.0	RUS	True	25.4	54.0	1018.6	29.9	333.0	0.5	Abu Dhabi	2021.0
1534	84.779	3.0	1.0	17.315	36.523	30.941	288.0	319.0	220.0	322.0	True	4	3.0	RAI	True	25.5	55.0	1018.5	29.9	273.0	0.5	Abu Dhabi	2021.0
1535	84.906	8.0	3.0	17.419	36.340	31.147	289.0	319.0	219.0	324.0	True	4	3.0	MSC	True	25.2	55.5	1018.6	29.4	283.0	0.2	Abu Dhabi	2021.0
1536	85.685	8.0	3.0	17.522	36.868	31.295	287.0	318.0	215.0	324.0	True	4	3.0	MAZ	True	25.2	55.5	1018.6	29.4	283.0	0.2	Abu Dhabi	2021.0

1537 rows × 23 columns

Data Visualizations

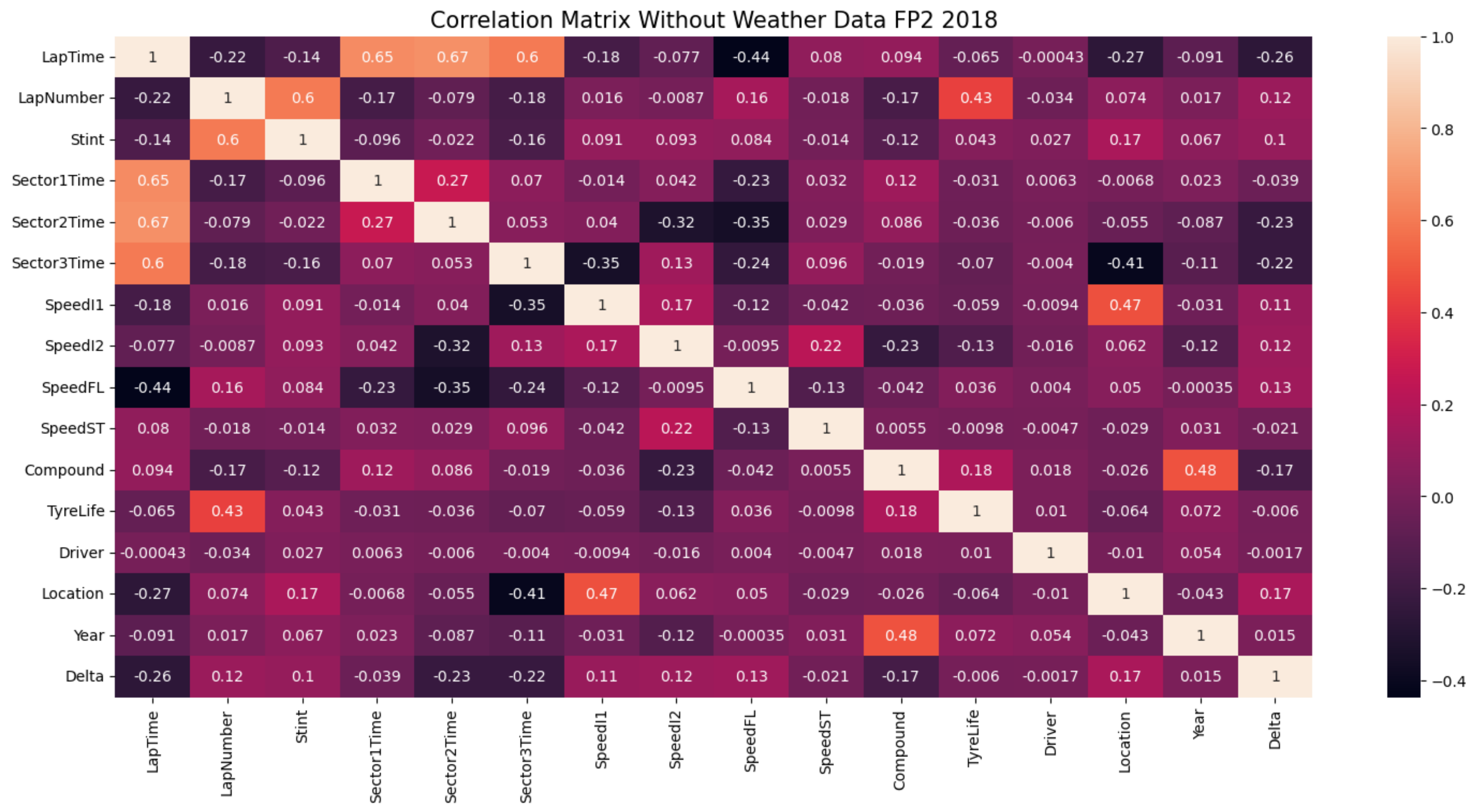
Correlation Matrix

- High Correlation
  - Sector times – Lap time
  - Tire compound – Lap time
- Negative Correlation
  - Finish line speed – Lap time
  - Lap number – Lap time
- Low Correlation
  - Weather data – Lap time
  - Driver – Lap time

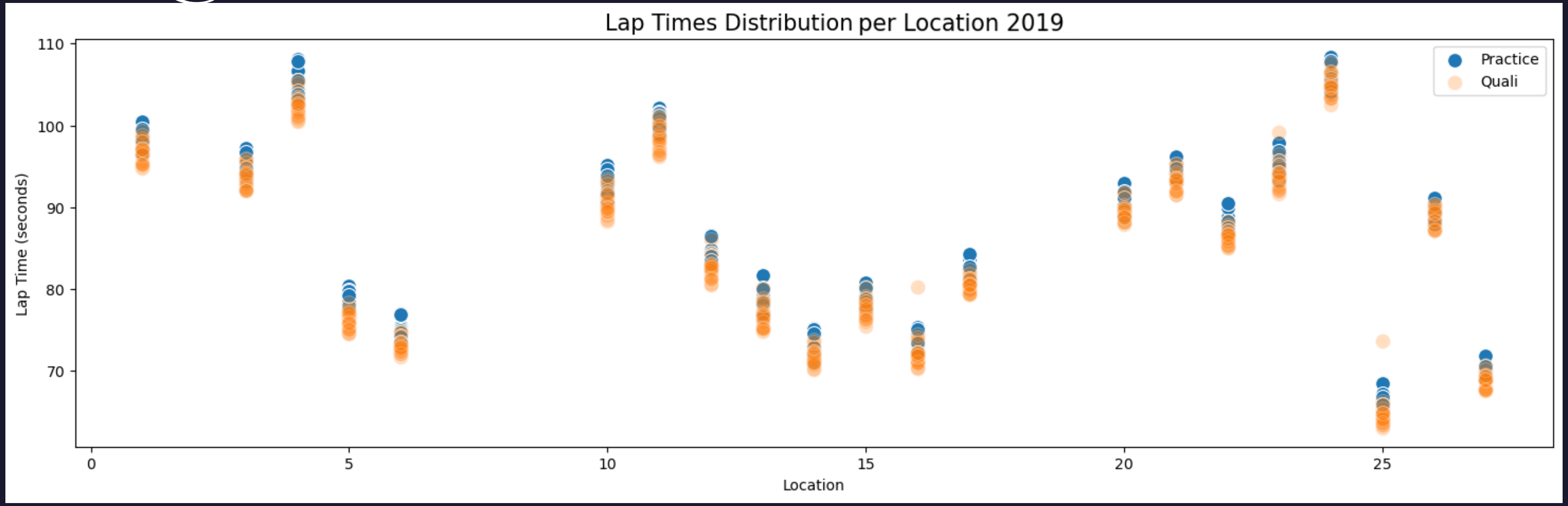


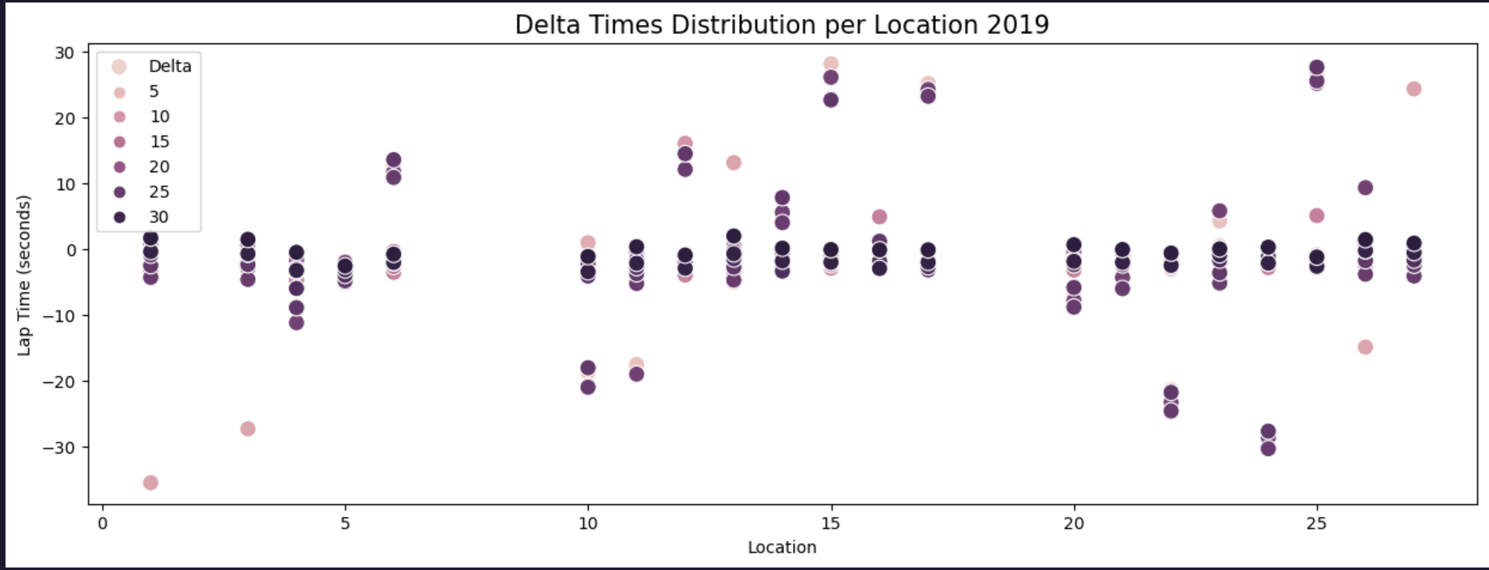
Correlation Matrix

Correlation matrix without weather data

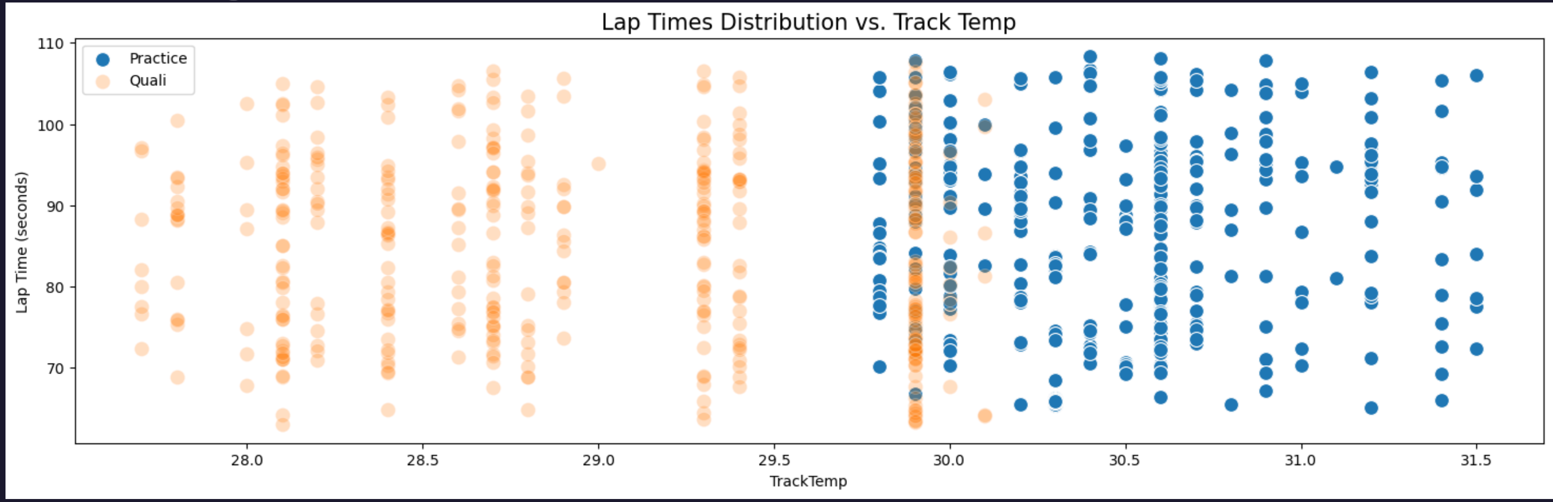


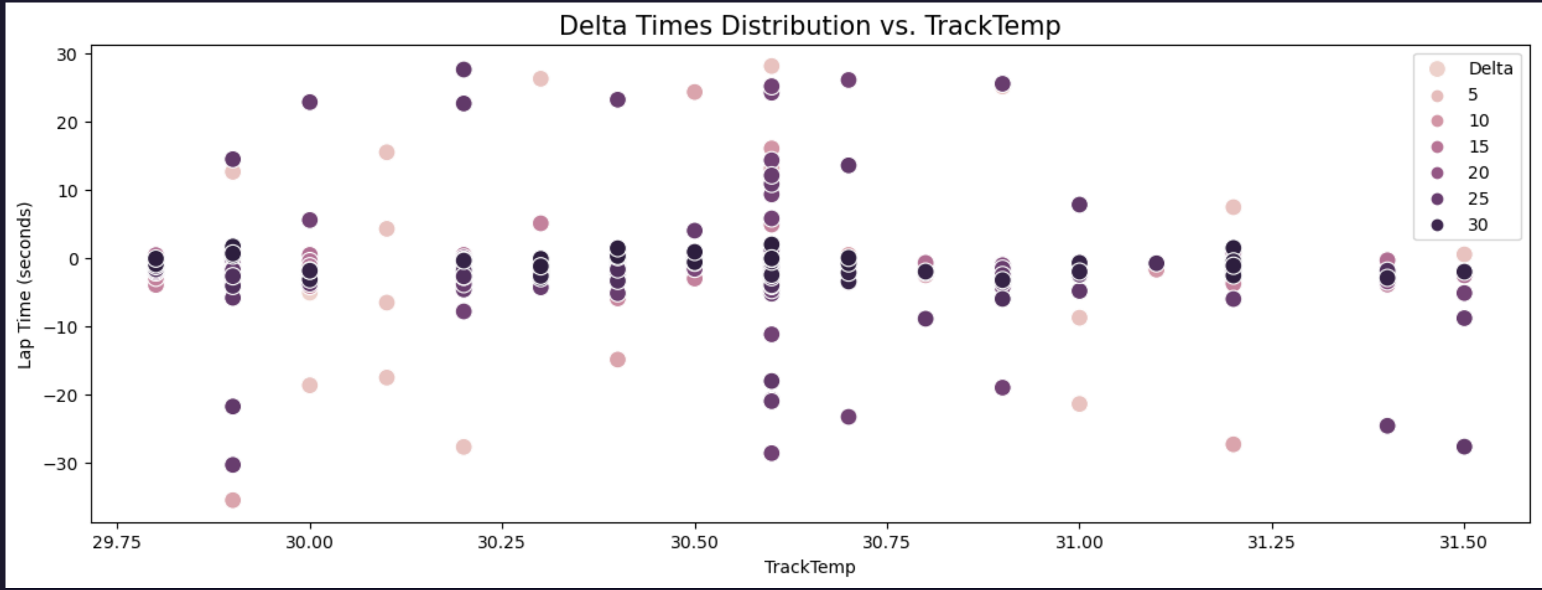
Comparing Distributions - Location





Comparing Distributions – Track Temp





Modeling

Models

- Predicting Lap Time
- Regession problem because a specific number is predicted
- Models used:
    - Radom Forest Regressor
    - Extra Trees Regressor
    - XGBoost Regressor
    - MLP Regressor

Training the Models

```python
def train_model(practice_list, model, filename):
    # Seperating X & y
    X = practice_list.drop(['LapTime', 'Year', 'IsAccurate', 'IsPersonalBest', 'AirTemp', 'Humidity', 'Pressure', 'WindSpeed', 'WindDirection', 'TrackTemp', 'Driver'], axis=1)
    y = practice_list.LapTime
    # Scaling data
    StandardScaler().fit_transform(X, y)
    # Train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    # Training, saving, loading and making predictions for evaluation metrics
    model.fit(X_train, y_train)
    joblib.dump(model, filename)
    model = joblib.load(filename)
    train_preds = model.predict(X_test)
    print(f'Training R2 score:{model.score(X_test, y_test)}')
    print(f'Training MSE: {mean_squared_error(y_test, train_preds)}')
    print(f'Train MAE: {mean_absolute_error(y_test, train_preds)}')
    return model
```

Training Results

MLP TRAINING SCORES

```
Training R2 score:0.9972779649778754
Training MSE: 0.40629735317697224
Train MAE: 0.4981446255043841
```

EXTRA TREES TRAINING SCORE

```
Training R2 score:0.9992244094322533
Training MSE: 0.12089605303170213
Train MAE: 0.2469746341463352
```

RANDOM FOREST TRAINING SCORE

```
Training R2 score:0.9988195257739632
Training MSE: 0.3517124790875036
Train MAE: 0.3414141071428647
```

XGBOOST TRAINING SCORE

```
Training R2 score:0.9964318382844187
Training MSE: 0.8351853145959757
Train MAE: 0.46549240166800515
```

Hold Out Prediction Scores

MLP PREDICTION SCORE

```
MLP Regressor Hold Out Predictions MSE: 23.58911345582122
MLP Regressor Hold Out Predictions MAE: 3.3469207458795394
```

EXTRA TREES PREDICTION SCORE

```
Extra Trees Regressor Hold Out PredictionsMSE: 18.61427264383672
Extra Trees Regressor Hold Out Predictions MAE: 2.7670727168949685
```

RANDOM FOREST PREDICTION SCORE

```
RandomForestRegressor Hold Out Predictions MSE: 105.21875730540634
RandomForestRegressor Hold Out Predictions MAE: 6.537633630136983
```

XGBOOST PREDICTION SCORE

```
XGBoost Regressor Hold Out Predictions MSE: 28.29304706106421
XGBoost Regressor Hold Out Predictions MAE: 3.8292436979110938
```

Hyperparameter Tuning

```python
X = fp2.drop(['LapTime', 'Year', 'IsAccurate', 'IsPersonalBest', 'AirTemp', 'Humidity', 'Pressure', 'WindSpeed', 'WindDirection', 'TrackTemp', 'Driver'], axis=1)
y = fp2.LapTime

# Building parameter grids for our models for hyperparameter tuning.
rf_grid = {'n_estimators': [25, 25, 50, 75],
           'max_depth': [2, 5, 10, 15],
           'min_samples_leaf': [2, 5, 7, 10],
           'max_leaf_nodes': [5, 10, 15, 20]}
ext_grid = {'n_estimators': [15, 25, 50, 75],
            'max_depth': [2, 5, 10, 15],
            'min_samples_leaf': [2, 5, 7, 10],
            'max_leaf_nodes': [5, 10, 15, 20]}
xgb_grid = {'n_estimators': [25, 25, 50, 75],
            'max_depth': [2, 5, 10, 15],
            'gamma': [0.01, 0.001, 0.0001, 0.00001],
            'min_child_weight': [0.9, 0.5, 0.25, 0.1, 0.01, 0.001],
            'subsample': [0.9, 0.5, 0.25, 0.1],
            'colsample_bytree': [0.9, 0.5, 0.25, 0.1]}
mlp_grid = {'hidden_layer_sizes': [(15,), (25,), (50,), (75,)],
            'alpha': [0.1, 0.01, 0.001],
            'learning_rate_init': [0.25, 0.1, 0.01],
            'max_iter': [2000],
            'tol': [0.1, 0.01, 0.001],
            'early_stopping': [False, True],
            'n_iter_no_change': [15, 18, 20, 25]}

rs_rf_rg_jl = RandomizedSearchCV(rf_rg_jl, rf_grid, cv=10).fit(X, y)
rs_ext_rg_jl = RandomizedSearchCV(ext_rg_jl, ext_grid, cv=10).fit(X, y)
rs_xgb_rg_jl = RandomizedSearchCV(xgb_rg_jl, xgb_grid, cv=10).fit(X, y)
rs_mlp_rg_jl = RandomizedSearchCV(mlp_rg_jl, mlp_grid, cv=10).fit(X, y)
```

Hyperparameter Tuning - Results

MLP PREDICTION SCORE

```
Randomized Search MLP Regressor Hold Out Predictions R2 score: 0.9793511358208975
Randomized Search MLP Regressor Regressor Hold Out Predictions MSE: 2.572224339331646
Randomized Search MLP Regressor Regressor Hold Out Predictions MAE: 0.17262879716296148
```

EXTRA TREES PREDICTION SCORE

```
Randomized Search Extra Trees Regressor Hold Out Predictions R2 score: 0.9821502820844555
Randomized Search Extra Trees Regressor Hold Out Predictions MSE: 2.2235353225401058
Randomized Search Extra Trees Regressor Hold Out Predictions MAE: 1.1568782833634457
```

RANDOM FOREST PREDICTION SCORE

```
Randomized Search RandomForestRegressor Hold Out Predictions R2 score: 0.977658874625929
Randomized Search RandomForestRegressor Hold Out Predictions MSE: 2.7830289335431604
Randomized Search RandomForestRegressor Hold Out Predictions MAE: 1.1925070682549153
```

XGBOOST PREDICTION SCORE

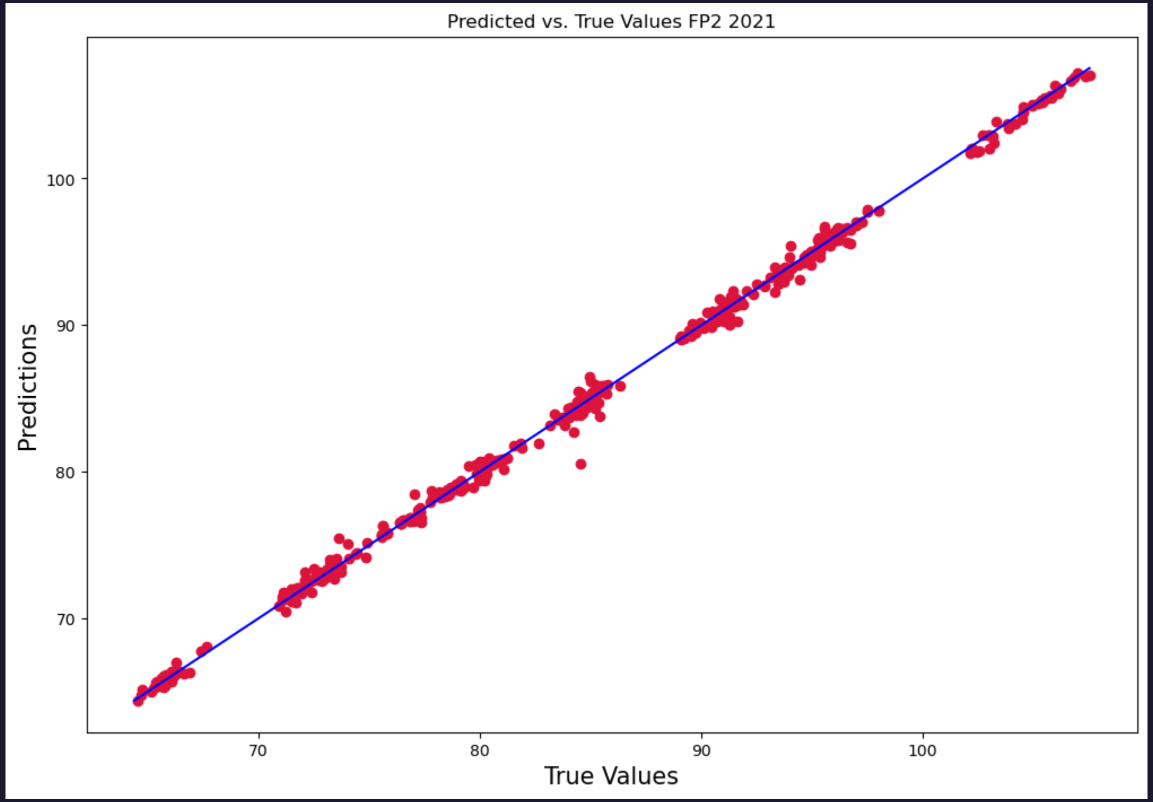
```
Randomized Search XGBoost Regressor Hold Out Predictions R2 score: 0.9982118776215452
Randomized Search XGBoost Regressor Hold Out Predictions MSE: 0.22274600015142476
Randomized Search XGBoost Regressor Hold Out Predictions MAE: 0.318373037259873
```

XGBOOST BEST PARAMETERS

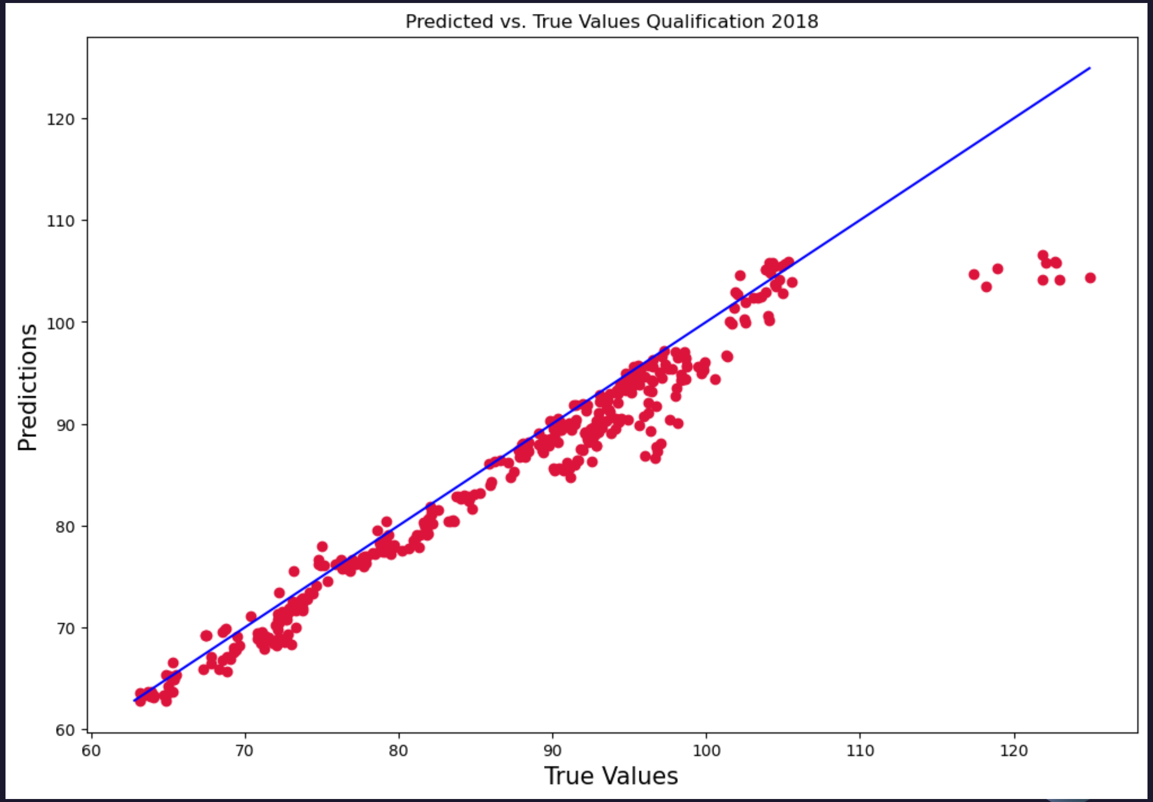
```
{'subsample': 0.5,
 'n_estimators': 25,
 'min_child_weight': 0.1,
 'max_depth': 10,
 'gamma': 0.01,
 'colsample_bytree': 0.5}
```

Results

REGRESSION LINE FOR PRACTICE DATA (HOLD OUT DATA)



REGESSION LINE FOR QUALIFICATION DATA



References

- Fast FI : https://theoehrly.github.io/Fast-F1/index.html
- Official F1 data stream: https://www.formula1.com/en/f1-live.html
- Ergast web api: http://ergast.com/mrd/
- AWS Machine Learning Blog: https://aws.amazon.com/fr/blogs/machine-learning/predicting-qualification-ranking-based-on-practice-session-performance-for-formula-1-grand-prix/
- XGBoost documentation: https://xgboost.readthedocs.io/en/stable/index.html
- Sklearn:
  - Extra Trees: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html#sklearn.ensemble.ExtraTreesRegressor.html
  - MLP: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html
  - Random Forest: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Thank you