MIRIPLANTS

감귤 착과량 예측 AI 경진대회

EDA

In [3]:

```
display(data_train.head())
display(data_train.info())
display(data_train.shape)
display(data_train.isna().sum().sum())
```

	ID	착과량(int)	수고(m)	수관폭1(min)	수관폭2(max)	수관폭평균	2022-09-01 새순	2022-09-02 새순	2022-09-03 새순	2022-09-04 새순	...	2022-11-19 엽록소	2022-11-20 엽록소	2022-11-21 엽록소	2022-11-22 엽록소	2022-11-23 엽록소	2022-11-24 엽록소	2022-11-25 엽록소
0	TRAIN_0000	692	275.0	287.0	292.0	289.5	2.8	2.8	2.7	2.7	...	70.978249	70.876794	70.705253	70.559603	70.427356	70.340491	70.29383
1	TRAIN_0001	534	293.0	284.0	336.0	310.0	3.3	3.3	3.3	3.2	...	71.535483	71.382303	71.253604	71.092665	70.955608	70.796630	70.59755
2	TRAIN_0002	634	300.0	392.0	450.0	421.0	3.0	2.9	2.9	2.9	...	71.279804	71.199570	71.144020	71.026740	70.920038	70.876723	70.71012
3	TRAIN_0003	639	289.0	368.0	379.0	373.5	3.1	3.0	3.0	3.0	...	69.934615	69.884124	69.845683	69.794682	69.779813	69.614644	69.45540
4	TRAIN_0004	496	306.0	353.0	358.0	355.5	3.7	3.6	3.6	3.6	...	68.313016	68.285364	68.209860	68.209458	68.040083	67.859963	67.77555

5 rows × 184 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2207 entries, 0 to 2206
Columns: 184 entries, ID to 2022-11-28 엽록소
dtypes: float64(182), int64(1), object(1)
memory usage: 3.1+ MB

None

(2207, 184)

0
```

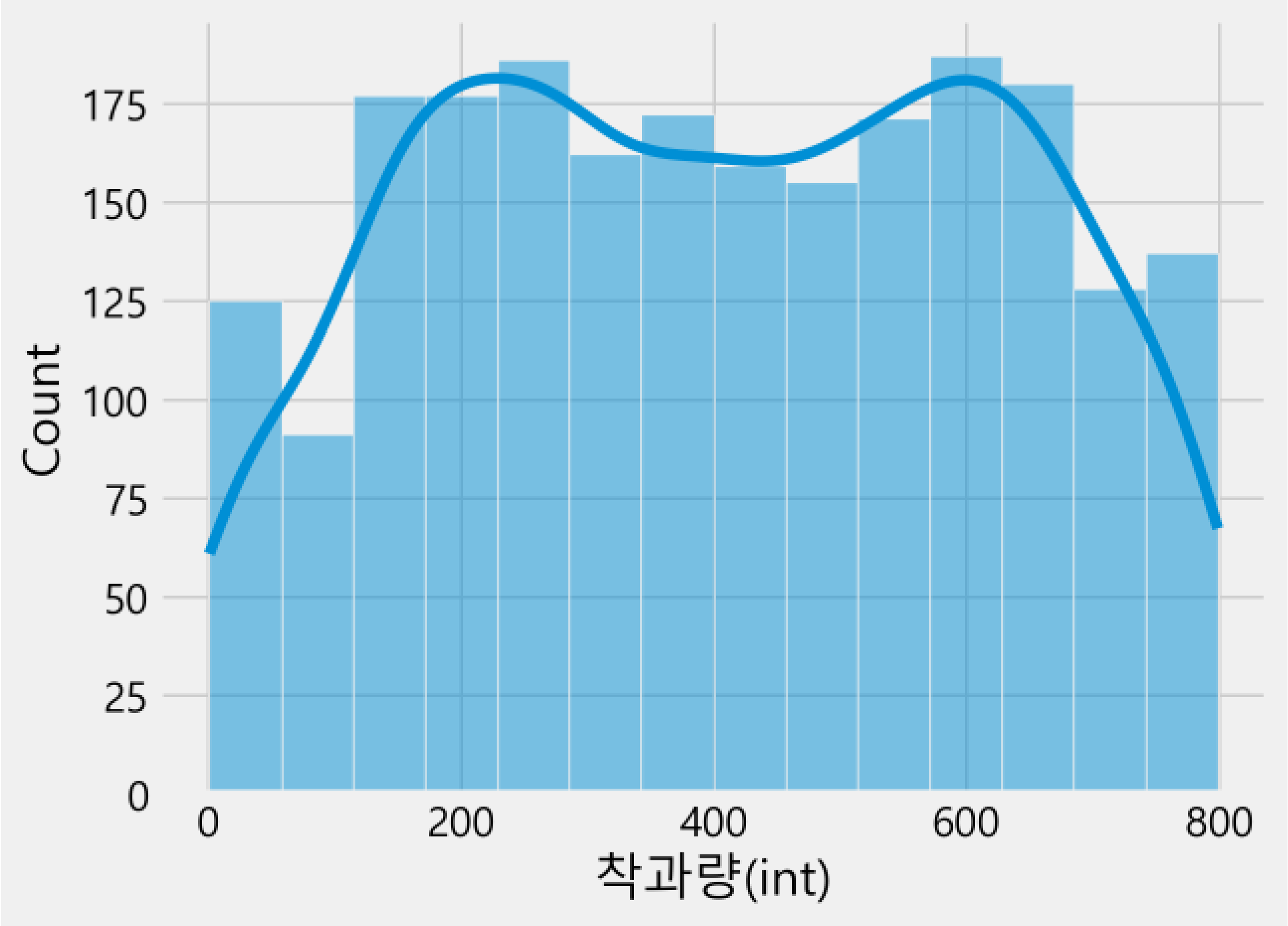
[Code]

display(data_train.head())
display(data_train.info())
display(data_train.shape)
display(data_train.isna().sum().sum())

[설명]

Train 및 Test 테이블 데이터 확인
Train 및 Test 데이터 type등 정보확인
Train 및 Test 데이터 형태 확인
Train 및 Test 데이터 결측치 확인

```
착과량(int)의 최대값 : 799
착과량(int)의 최소값 : 1
착과량(int)의 평균값 : 406.22247394653374
착과량(int)의 중앙값 : 404.0
착과량(int)의 최빈값 1순위 : 300 & 9개
착과량(int)의 최빈값 2순위 : 231 & 9개
착과량(int)의 최빈값 3순위 : 632 & 9개
```



[Target 데이터 분포 확인]

Target 데이터 분포가 확인
정규분포와 많이 차이 나지 않는 형태를
띠우고 있어서 Log변환은 따로 진행하지
않았음

Feature set_1

```
#train, target split
y_train = data_train['착과량(int)']
X_drop_list = ['ID']
X_train = data_train.drop(X_drop_list, axis = 1)
X_test = data_test.drop(["ID"], axis = 1)
```

```
#feature selection(target correlation limit)
high_corr = data_train.corr().abs().sort_values(by='착과량(int)',ascending=False).iloc[:,:1]
features_name = high_corr[high_corr['착과량(int)']>0.9].index
features_name = list(features_name)
features_name.remove('착과량(int)')
X,y = X_train.drop(['착과량(int)'], axis=1) , X_train['착과량(int)']

X_1 = X[features_name]
X_test_1 = X_test[features_name]
```

```
#이상치(outlier)처리
X_1 = X_1.apply(lambda x: x.clip(x.quantile(.01), x.quantile(.99)), axis=0)
```

[Feature set_1]

1. Train데이터 Correlation을 찍은 후 0.9초과인 피처셋들을 사용
2. Clip을 이용하여 상,하위 1% 이상치를 대체

Feature set_2

```
X_train['9월_새순_mean'] = X_train.iloc[:,4:34].mean(axis = 1)
X_train['9월_새순_std'] = X_train.iloc[:,4:34].std(axis = 1)
X_train['9월_새순_var'] = X_train.iloc[:,4:34].var(axis = 1)
X_train['10월_새순_mean'] = X_train.iloc[:,34:65].mean(axis = 1)
X_train['10월_새순_std'] = X_train.iloc[:,34:65].std(axis = 1)
X_train['10월_새순_var'] = X_train.iloc[:,34:65].var(axis = 1)
X_train['11월_새순_mean'] = X_train.iloc[:,65:93].mean(axis = 1)
X_train['11월_새순_std'] = X_train.iloc[:,65:93].std(axis = 1)
X_train['11월_새순_var'] = X_train.iloc[:,65:93].var(axis = 1)
X_train['9월_엽록소_mean'] = X_train.iloc[:,93:123].mean(axis = 1)
X_train['9월_엽록소_std'] = X_train.iloc[:,93:123].std(axis = 1)
X_train['9월_엽록소_var'] = X_train.iloc[:,93:123].var(axis = 1)
X_train['10월_엽록소_mean'] = X_train.iloc[:,123:154].mean(axis = 1)
X_train['10월_엽록소_std'] = X_train.iloc[:,123:154].std(axis = 1)
X_train['10월_엽록소_var'] = X_train.iloc[:,123:154].var(axis = 1)
X_train['11월_엽록소_mean'] = X_train.iloc[:,154:182].mean(axis = 1)
X_train['11월_엽록소_std'] = X_train.iloc[:,154:182].std(axis = 1)
X_train['11월_엽록소_var'] = X_train.iloc[:,154:182].var(axis = 1)
```

```
X_train['새순max'] = X_train.iloc[:,4:93].max(axis=1)
X_train['새순min'] = X_train.iloc[:,4:93].min(axis=1)
X_train['엽록소max'] = X_train.iloc[:,93:182].max(axis=1)
X_train['엽록소min'] = X_train.iloc[:,93:182].min(axis=1)
X_train['새순차이'] = X_train['새순max']-X_train['새순min']
X_train['엽록소차이'] = X_train['엽록소max']-X_train['엽록소min']
X_train['수고X수관폭'] = X_train['수고(m)']*X_train['수관폭평균']
X_train['수관폭차이'] = X_train['수관폭2(max)']-X_train['수관폭1(min)']
```

```
for i in range(0,89):
    X_train[f'새순+엽록소_{i}'] = X_train.iloc[:,4:93].iloc[:,i]+X_train.iloc[:,93:182].iloc[:,i]
for i in range(0,89):
    X_train[f'새순-엽록소_{i}'] = X_train.iloc[:,4:93].iloc[:,i]-X_train.iloc[:,93:182].iloc[:,i]
for i in range(0,89):
    X_train[f'새순*엽록소_{i}'] = X_train.iloc[:,4:93].iloc[:,i]*X_train.iloc[:,93:182].iloc[:,i]
for i in range(0,89):
    X_train[f'새순/엽록소_{i}'] = X_train.iloc[:,4:93].iloc[:,i]/X_train.iloc[:,93:182].iloc[:,i]
```

```
X_train.shape, X_test.shape
```

```
((2207, 564), (2208, 564))
```

[Feature set_2]

1. 파생변수 생성
   - 월별 새순,엽록소 mean,std,var
   - 새순 max,min,(max-min)
   - 수고X수관폭, 수관폭차이
   - 새순(X,/,+,-)엽록소

2. X_train.shape(2207, 564)
   - 401개 Featrue 생성

```python
def remove_outlier(X, q=0.02):
    df = pd.DataFrame(X)
    return df.apply(lambda x: x.clip(x.quantile(q), x.quantile(1-q)), axis=0).values

numeric_transformer = Pipeline(
    steps=[
        ("outlier", FunctionTransformer(remove_outlier, kw_args={'q':0.02})),
        ("scaler", MinMaxScaler()),
    ]
)

column_transformer = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
    ]
)

preprocessor = Pipeline(
    steps=[
        ("column", column_transformer),
    ]
)

model = Pipeline(
    steps=[
        ("preprocessor", preprocessor),
        ("Regressor", LGBMRegressor(objective="regression", metric="mae", random_state=SEED)),
    ]
)
```

```python
# 최적값으로 파이프라인 재설정
model.set_params(preprocessor__column__num__outlier__kw_args =  {'q': 0.02}, preprocessor__column__num__scaler = MinMaxScaler())

# 전처리 파이프라인만 수행
X_train = preprocessor.fit_transform(X_train, y_train)
X_test = preprocessor.transform(X_test)
```

```python
X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)
```

```python
# 피처셀렉션
fs = SelectPercentile(percentile=13).fit(X_train, y_train)
X_train = fs.transform(X_train)
X_test = fs.transform(X_test)
```

[Feature set_2]

3. Pipeline 사용(Outlier -> Scaler)
   - Outlier -> Clip사용하여 상하위 2% 대체
   - Scaler -> MinMaxScaler 사용

4. FeatureSelection
   - SelectPercentile 사용 -> 564개중에서 중요도높은 13%만 사용

XGBRegressor

```python
#optuna를 이용해 hyperparameter tuning
xgb_best_params_1 = {'lambda': 0.002645916029508221,
                     'alpha': 0.06770804282734474,
                     'colsample_bytree': 0.42500508042724955,
                     'subsample': 0.7135736798352763,
                     'learning_rate': 0.003449175996248127,
                     'n_estimators': 2538,
                     'max_depth': 4,
                     'min_child_weight': 2,
                     'objective': 'reg:squarederror',
                     'tree_method': 'gpu_hist',
                     'predictor': 'gpu_predictor'}

xgb_best_params_2 = {'lambda': 0.059360963228304024,
                     'alpha': 0.985629252513064,
                     'colsample_bytree': 0.4569397260113678,
                     'subsample': 0.4754658082470086,
                     'learning_rate': 0.0029407888288556297,
                     'n_estimators': 2020,
                     'max_depth': 11,
                     'min_child_weight': 49,
                     'objective': 'reg:squarederror',
                     'tree_method': 'gpu_hist',
                     'predictor': 'gpu_predictor'}
```

```python
#multi-kfold1(과적합 방지를 이용해 사용)
xgb_pred_1 = []

kfold_list = [4, 5, 6]
for kfold in kfold_list:
    print(f"{kfold} Fold start")
    i = 0
    xgb_nmae = []
    kf = KFold(n_splits=kfold, random_state=42, shuffle=True)
    for fold, (tr_idx, val_idx) in enumerate(kf.split(X_1)):
        tr_x, tr_y = X_1.iloc[tr_idx], y.iloc[tr_idx]
        val_x, val_y = X_1.iloc[val_idx], y.iloc[val_idx]

        #사이킷 런 래퍼 XGB 학습
        xgb = XGBRegressor(**xgb_best_params_1)
        xgb.fit(tr_x, tr_y, eval_set = [(val_x, val_y)], early_stopping_rounds = 100, verbose = 50, eval_metric = 'mae')
        val_pred = xgb.predict(val_x).astype(int)
        fold_nmae = NMAE(val_y, val_pred)
        xgb_nmae.append(fold_nmae)
        print(f"{i + 1}/{kfold} Fold NMAE = {fold_nmae}")
        i += 1
        fold_pred = xgb.predict(X_test_1)
        xgb_pred_1.append(fold_pred)

    print(f"\nAVG of NMAE = {np.mean(xgb_nmae)}")
```

[XGBRegressor]

1. Optuna를 사용하여 파라미터를 추출
2. Kfold를 사용하여 교차검증
3. Optuna에서 나온 Best parameter를 XGBRegressor에 적용

LGBMRegressor

```
lgb_param = {'objective' : 'regression',
            'device' : 'gpu',
            'metric' : 'mae'}
```

```
#multi-kfold1
lgb_pred_1 = []

kfold_list = [4, 5, 6]
for kfold in kfold_list:
    print(f"{kfold} Fold start")
    i = 0
    lgb_nmae = []
    kf = KFold(n_splits=kfold, random_state=42, shuffle=True)
    for fold, (tr_idx, val_idx) in enumerate(kf.split(X_1)):
        tr_x, tr_y = X_1.iloc[tr_idx], y.iloc[tr_idx]
        val_x, val_y = X_1.iloc[val_idx], y.iloc[val_idx]

        lgb = LGBMRegressor(**lgb_param)
        lgb.fit(tr_x, tr_y, eval_set = [(val_x, val_y)], early_stopping_rounds = 100, verbose = 50, eval_metric = 'mae')
        val_pred = lgb.predict(val_x).astype(int)
        fold_nmae = NMAE(val_y, val_pred)
        lgb_nmae.append(fold_nmae)
        print(f"{i + 1}/{kfold} Fold NMAE = {fold_nmae}")
        i += 1
        fold_pred = lgb.predict(X_test_1)
        lgb_pred_1.append(fold_pred)

    print(f"\nAVG of NMAE = {np.mean(lgb_nmae)}")
```

[LGBMRegressor]

1. Parameter : 평가 메트릭 및 GPU사용만 사용
2. Kfold를 사용하여 교차검증
3. 과적합 방지를 위해 기본 모델 및 early_stopping사용

CatboostRegressor

```
#multi-kfold1
cat_pred_1 = []

kfold_list = [4, 5, 6]
for kfold in kfold_list:
    print(f"{kfold} Fold start")
    i = 0
    cat_nmae = []
    kf = KFold(n_splits=kfold, random_state=42, shuffle=True)
    for fold, (tr_idx, val_idx) in enumerate(kf.split(X_1)):
        tr_x, tr_y = X_1.iloc[tr_idx], y.iloc[tr_idx]
        val_x, val_y = X_1.iloc[val_idx], y.iloc[val_idx]

        cat = CatBoostRegressor(use_best_model = True,
                                task_type = 'GPU',
                                iterations = 10000,
                                eval_metric = 'MAE')
        cat.fit(tr_x, tr_y, eval_set = [(val_x, val_y)], early_stopping_rounds = 100, verbose = 50)
        val_pred = cat.predict(val_x).astype(int)
        fold_nmae = NMAE(val_y, val_pred)
        cat_nmae.append(fold_nmae)
        print(f"{i + 1}/{kfold} Fold NMAE = {fold_nmae}")
        i += 1
        fold_pred = cat.predict(X_test_1)
        cat_pred_1.append(fold_pred)

    print(f"₩nAVG of NMAE = {np.mean(cat_nmae)}")
```

[CatboostRegressor]

1. Catboost같은 경우는 파라미터 튜닝을 안해도 성능이 기본적으로 잘나오기 때문에 Optuna를 사용안함
2. Kfold를 사용하여 교차검증 사용
3. Early_stopping_rounds를 사용하여 과적합 방지

Model Ensemble

```
submission1 = submission.copy()
submission2 = submission.copy()

submission1['착과량(int)'] = xgb_pred_sum_1*0.4 + lgb_pred_sum_1*0.4 + cat_pred_sum_1*0.2
submission2['착과량(int)'] = xgb_pred_sum_2*0.4 + lgb_pred_sum_2*0.4 + cat_pred_sum_2*0.2
```

Submission Ensemble

```
submission['착과량(int)'] = submission1['착과량(int)']*0.8 + submission2['착과량(int)']*0.2
```

```
submission
```

	ID	착과량(int)
0	TEST_0000	240.355276
1	TEST_0001	749.922491
2	TEST_0002	151.966817
3	TEST_0003	446.575871
4	TEST_0004	685.091691
...	...	...
2203	TEST_2203	751.960723
2204	TEST_2204	338.593112

[Ensemble]

1. [Model Ensemble]
   Feature_set_1, Feature_set_2 각각 Xgb 0.4, Lgb 0.4, Cat 0.2 가중치 사용하여 모델 앙상블진행
2. [Submission Ensemble]
   Feature_set_1, Feature_set_2를 모델 앙상블 하여 나온 Submission을 Feature_set_1 : 0.8, Feature_set_2 : 0.2 가중치를 사용하여 앙상블

MIRIPLANTS LOGO

MIRIPLANTS

Thank You!!

miri@miriplants.com