

プロ野球Freak

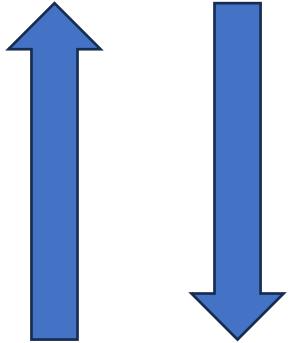


22vr039t Sunggon Park

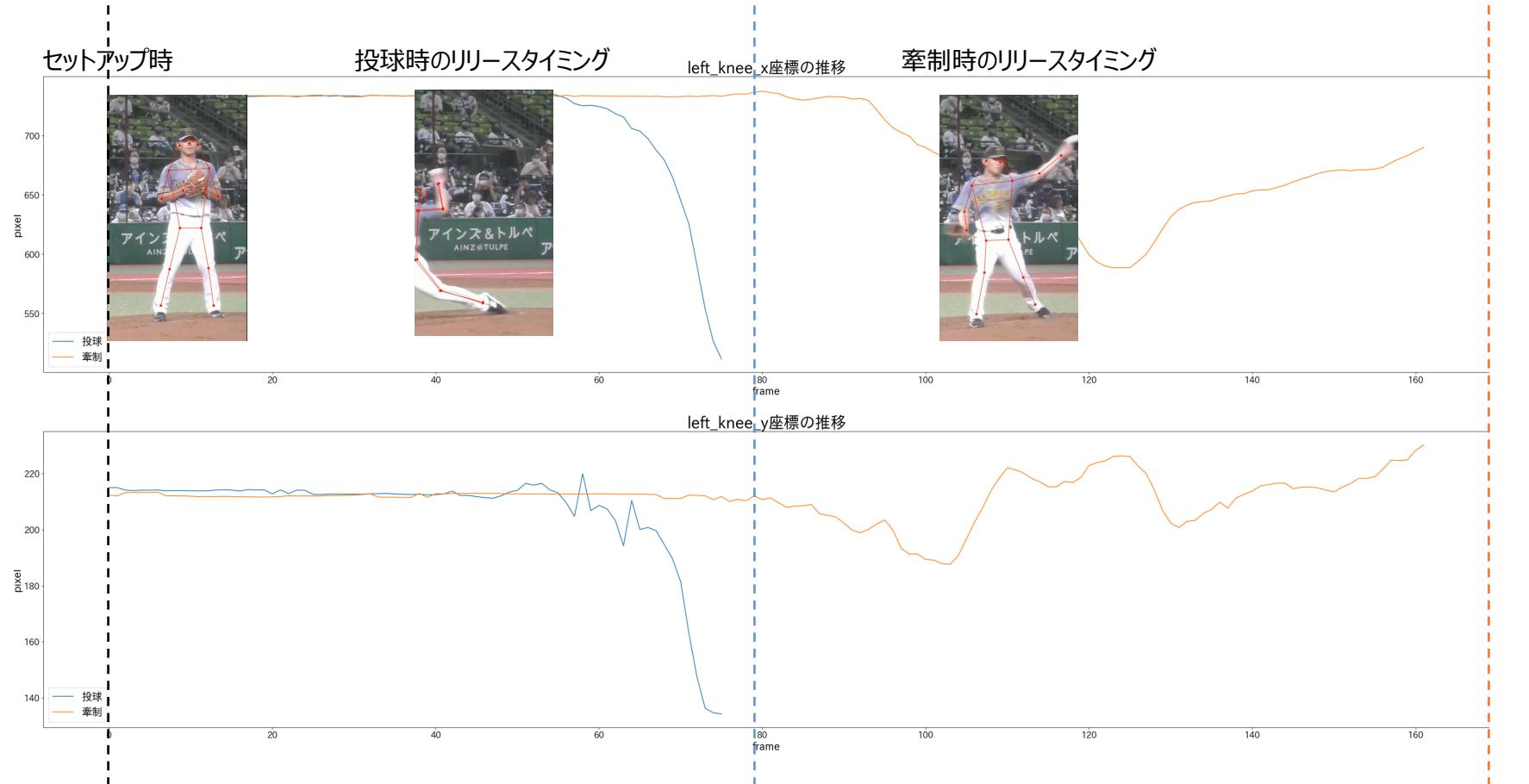
データの選択



分析結果



Data/Domain知識



問題設定

- 2009年から2022年までの日付や対戦相手など5-6の指標(説明変数)を元にホームゲームの観客数を予測する回帰モデルを作り、重要な指標を理解する。



プロ野球Freak
For all professional baseball fans
プロ野球のニュースや選手ブログの更新情報

各 プロ野球Freak トップ | チーム一覧 | 前 試合日程・結果 | カレンダー | 貯金・順位グラフ | 観客動員数 | 予告先発投手

ニュース | 選手Twitter | 選手Instagram | 選手ブログ | 選手誕生日 | プロ野球データFreak

ヤクルト DeNA 阪神 巨人 広島 中日 オリックス ソフトバンク 西武 楽天 ロッテ 日本ハム

トップページ >> チーム >> 埼玉西武ライオンズ >> 観客動員数 >> 2020年

★おすすめコンテンツ

- 貯金・順位推移グラフ
- 観客動員数
- 選手誕生日一覧
- 予告先発投手(成績・対戦成績)

プロ野球データFreak 特殊サイト

- プロ野球選手年俸ランキング
- 先発投手一覧 セ・リーグ | パ・リーグ
- ピッチャーの打撃成績
- ルーキー成績 打者 | 投手
- 新人王有資格者成績 打者 | 投手
- ファーム選手成績 打者 | 投手

プロ野球Freak ブログバーツ

あなたのブログにプロ野球の最新ニュース一覧を掲載!

プロ野球順位表 ブログバーツ

あなたのブログに順位表を掲載!

● 最近の試合結果

6月13日 (火)	セ・リーグ	西武	●	G 巨 3 - 0 西武
6月14日 (水)	パ・リーグ	オリックス	●	G 巨 7 - 1 西武
6月15日 (木)	他	ソフトバンク	●	G 巨 3 - 2 西武
6月16日 (金)	全球団	西武	●	C 広 2 - 0 西武
6月17日 (土)	シーズン	楽天	●	C 広 6 - 4 西武
6月18日 (日)	2023年	ロッテ	○	C 広 4 - 11 西武

● 今後の試合予定

6/23(金)	楽天 - 西武	楽天モバイル	18:00
6/24(土)	楽天 - 西武	楽天モバイル	14:00
6/25(日)	楽天 - 西武	楽天モバイル	13:00

ホームゲーム観客動員数一覧

チーム	1試合平均	試合数	合計	平均試合時間
西武	6,669人	45試合	300,120人	3時間15分

<https://baseball-freak.com/audience/22/lions.html>

データの前処理と可視化

乗りものニュース » 鉄道 » 西武ドームは「ベルーナドーム」に メットライフドームから命名権更新 3月から

西武ドームは「ベルーナドーム」に メットライフドームから命名権更新 3月から

2022.01.18 乗りものニュース編集部

コメント 0  BIブックマーク 0  ツイート  @ Save  お気に入り

tags: 鉄道, 西武, 西武球場前駅, 施設

ベルーナ電車も登場するかな？

西武ドームの名称変更へ

西武ライオンズは2022年1月17日（月）、プロ野球・埼玉西武ライオンズの本拠地である西武ドーム（埼玉県所沢市、西武狭山線西武球場前駅）の名称を、3月から「ベルーナドーム」に変更すると発表しました。



西武ドーム前にある「L-train 101」（画像：西武ライオンズ）。

<https://trafficnews.jp/post/114558>

```
df['球場'].value_counts()
```

西武ドーム	410
メットライフ	315
西武プリンス	136
ベルーナドーム	70
大宮	19
県営大宮	12
前橋	6
那覇	5
新潟	2
皇子山	2
東京ドーム	1

Name: 球場, dtype: int64

```
df['球場'] = df['球場'].apply(lambda stadium: '西武ドーム' if stadium in ['メットライフ', '西武プリンス', 'ベルーナドーム'] else stadium)
df['球場'].value_counts()
```

西武ドーム	931
大宮	19
県営大宮	12
前橋	6
那覇	5
新潟	2
皇子山	2
東京ドーム	1

Name: 球場, dtype: int64

データの前処理と可視化



<https://www.data.jma.go.jp/qmd/risk/obsdl/index.php>

```
df['年月日'] = df['年'].astype(str) + '/' + df['月'].astype(str) + '/' + df['日'].astype(str)
```

```
weather_df = pd.read_csv('weather.csv')  
weather_df
```

	年月日	降水量の合計(mm)	平均気温(°C)
0	2009/3/20	2.5	13.0
1	2009/3/21	0.0	9.6
2	2009/3/22	2.5	12.4
3	2009/3/23	0.0	11.2
4	2009/3/24	0.0	7.4
...

```
df = pd.merge(df, weather_df, on='年月日')  
df
```

データの前処理と可視化

```
df['スコア(ホーム)'] = df['スコア'].str.split(' - ').apply(lambda list_: int(list_[0]))
df['スコア(相手)'] = df['スコア'].str.split(' - ').apply(lambda list_: int(list_[1]))
df['スコア(合計)'] = df['スコア'].str.split(' - ').apply(lambda list_: sum(map(int, list_)))
df['スコア(差)'] = (df['スコア(ホーム)'] - df['スコア(相手)']).astype(int)
df = df[df.columns[df.columns != 'スコア']]
df
```

観客数	勝敗	対戦相手	先発投手	試合時間	球場	年	月	日	曜日	降水量の合計(mm)	平均気温(°C)	スコア(ホーム)	スコア(相手)	スコア(合計)	スコア(差)	
年月日																
2009/4/7	24011	○	オリックス	岸	3:16	西武ドーム	2009	4	7	火	0.0	14.2	8	3	11	5
2009/4/8	10001	●	オリックス	石井一	3:05	西武ドーム	2009	4	8	水	0.0	14.1	2	10	12	-8
2009/4/9	9813	○	オリックス	西口	2:57	西武ドーム	2009	4	9	木	0.0	16.5	13	6	19	7
2009/4/17	15181	●	日本ハム	涌井	3:20	西武ドーム	2009	4	17	金	5.0	10.8	2	4	6	-2
2009/4/18	28525	●	日本ハム	帆足	3:24	西武ドーム	2009	4	18	土	0.0	13.6	4	6	10	-2
...	

```
df[['直前試合のスコア(ホーム)', '直前試合のスコア(相手)', '直前試合のスコア(合計)', '直前試合のスコア(差)']] = \
... df[['スコア(ホーム)', 'スコア(相手)', 'スコア(合計)', 'スコア(差)']].shift(1)
df
```

Python

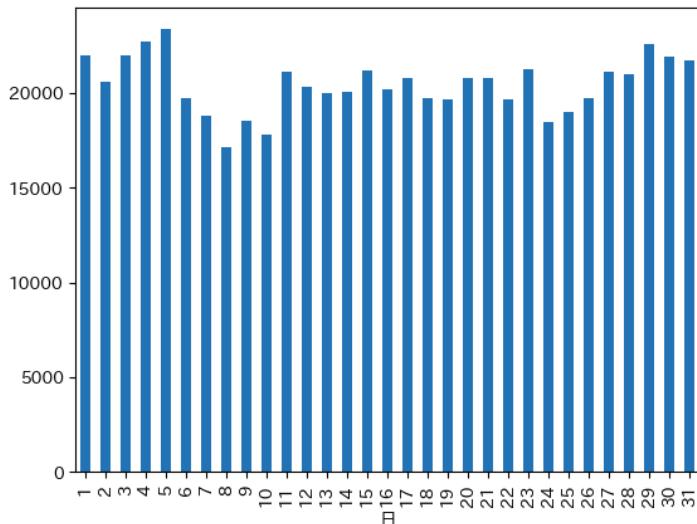
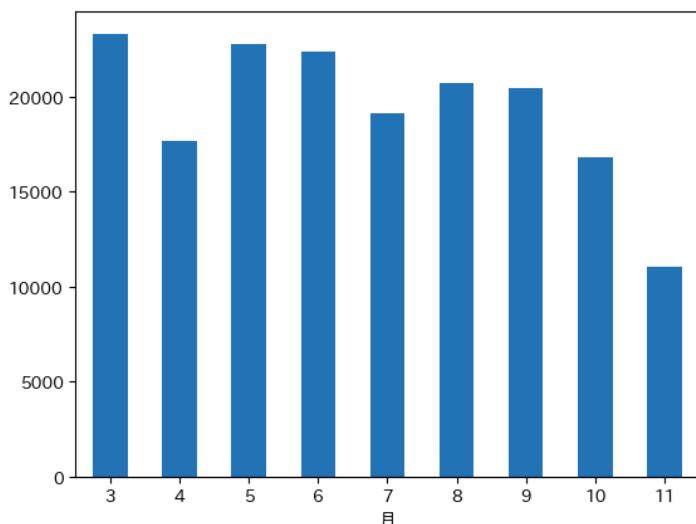
観客数	勝敗	対戦相手	先発投手	試合時間	球場	年	月	日	曜日	降水量の合計(mm)	平均気温(°C)	スコア(ホーム)	スコア(相手)	スコア(合計)	スコア(差)	直前試合のスコア(ホーム)	直前試合のスコア(相手)	直前試合のスコア(合計)	直前試合のスコア(差)	
年月日																				
2009/4/7	24011	○	オリックス	岸	3:16	西武ドーム	2009	4	7	火	0.0	14.2	8	3	11	5	NaN	NaN	NaN	NaN
2009/4/8	10001	●	オリックス	石井一	3:05	西武ドーム	2009	4	8	水	0.0	14.1	2	10	12	-8	8.0	3.0	11.0	5.0
2009/4/9	9813	○	オリックス	西口	2:57	西武ドーム	2009	4	9	木	0.0	16.5	13	6	19	7	2.0	10.0	12.0	-8.0
2009/4/17	15181	●	日本ハム	涌井	3:20	西武ドーム	2009	4	17	金	5.0	10.8	2	4	6	-2	13.0	6.0	19.0	7.0
2009/4/18	28525	●	日本ハム	帆足	3:24	西武ドーム	2009	4	18	土	0.0	13.6	4	6	10	-2	2.0	4.0	6.0	-2.0
...	

データの前処理と可視化

```
df['月別平均観客数'] = df.groupby(by='月')['観客数'].transform('mean').round(2)
df['日別平均観客数'] = df.groupby(by='日')['観客数'].transform('mean').round(2)
df
```

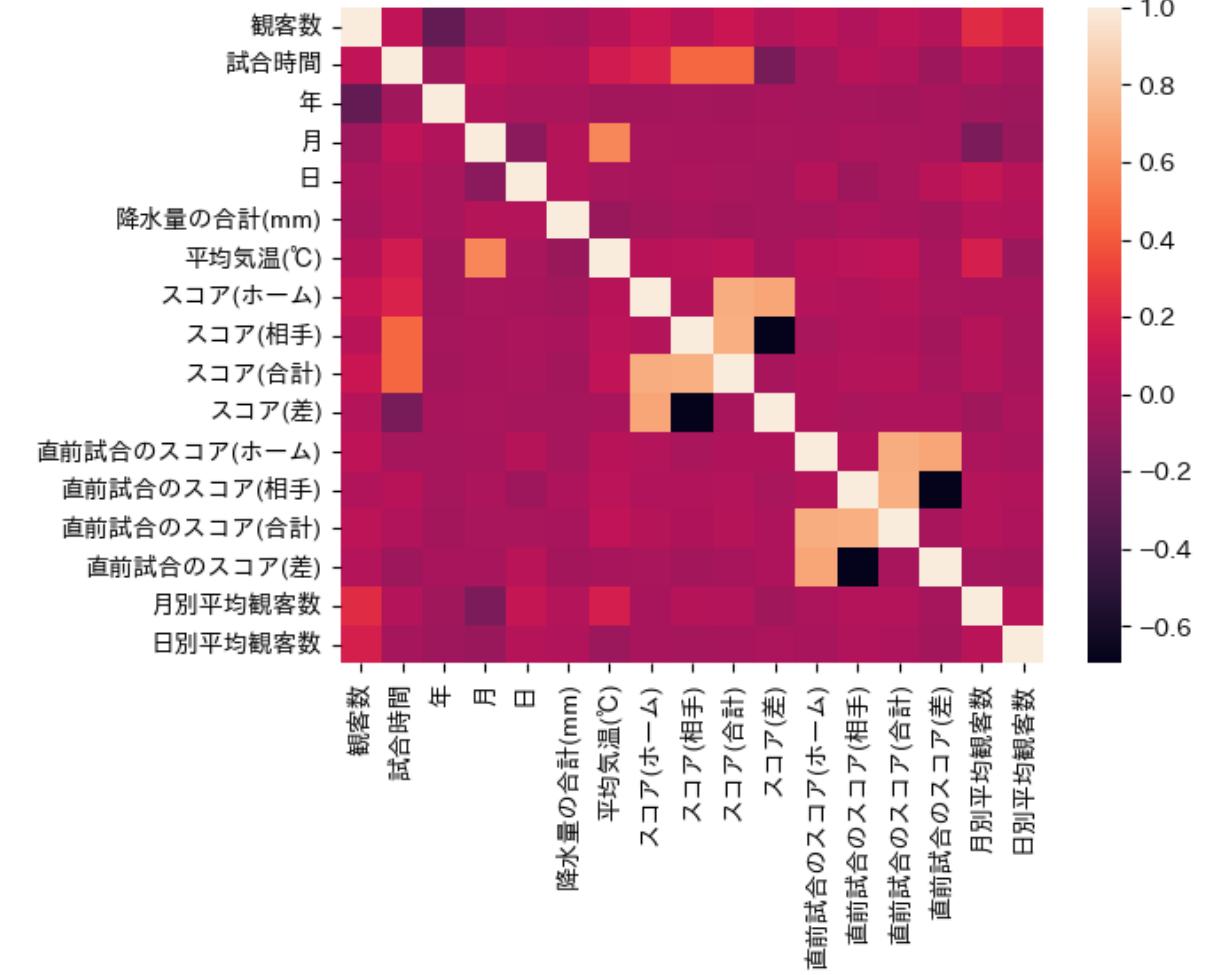
Python

観客数	勝敗	対戦相手	先発投手	試合時間	球場	年	月	日	曜日	降水量の合計(mm)	平均気温(°C)	スコア(ホーム)	スコア(相手)	スコア(合計)	スコア(差)	直前試合のスコア(ホーム)	直前試合のスコア(相手)	直前試合のスコア(合計)	直前試合のスコア(差)	直前試合の勝敗	月別平均観客数	日別平均観客数	
年月日																							
2009/4/7	24011	○	オリックス	岸	196	西武ドーム	2009	4	7	火	0.0	14.2	8	3	11	5	3	2	7	1	△	17693.64	18786.29
2009/4/8	10001	●	オリックス	石井一	185	西武ドーム	2009	4	8	水	0.0	14.1	2	10	12	-8	8	3	11	5	○	17693.64	17143.54
2009/4/9	9813	○	オリックス	西口	177	西武ドーム	2009	4	9	木	0.0	16.5	13	6	19	7	2	10	12	-8	●	17693.64	18510.93
2009/4/17	15181	●	日本ハム	涌井	200	西武ドーム	2009	4	17	金	5.0	10.8	2	4	6	-2	13	6	19	7	○	17693.64	20806.97
2009/4/18	28525	●	日本ハム	帆足	204	西武ドーム	2009	4	18	土	0.0	13.6	4	6	10	-2	2	4	6	-2	●	17693.64	19713.64
...

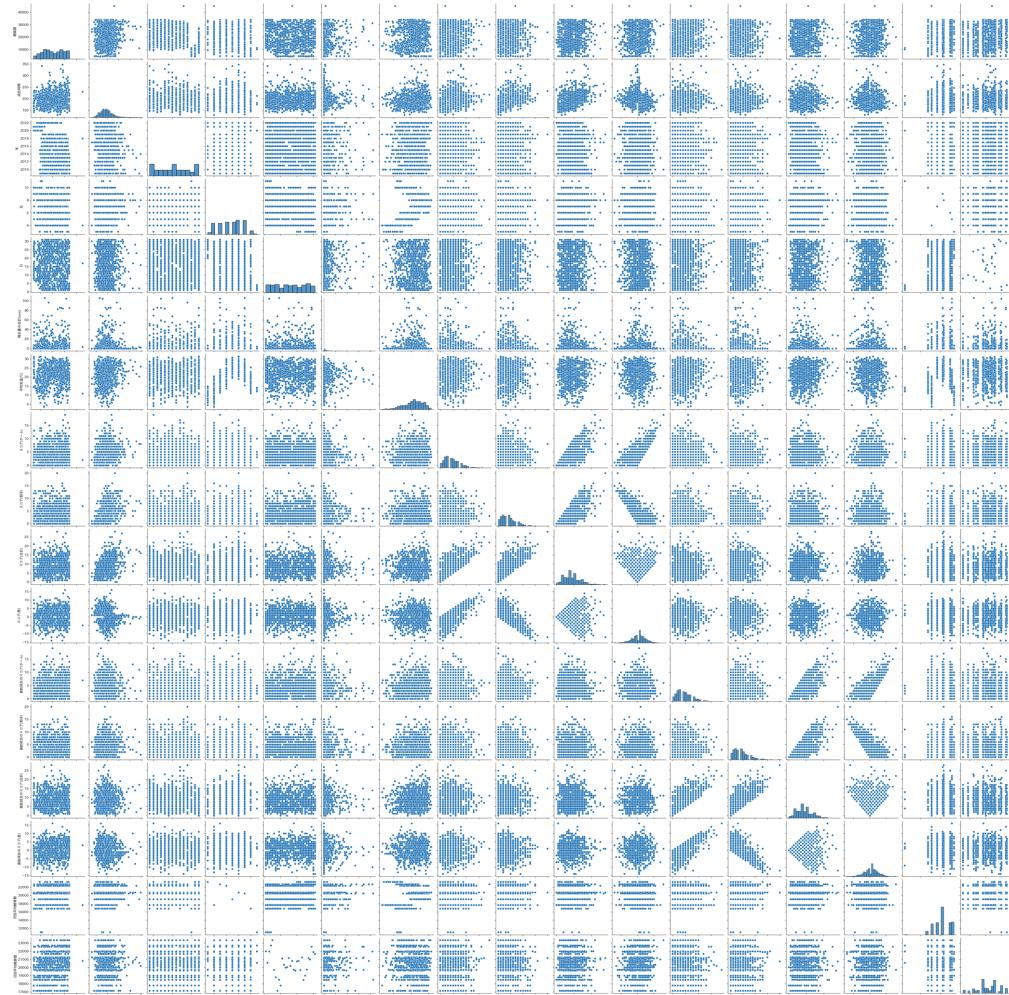


データの前処理と可視化

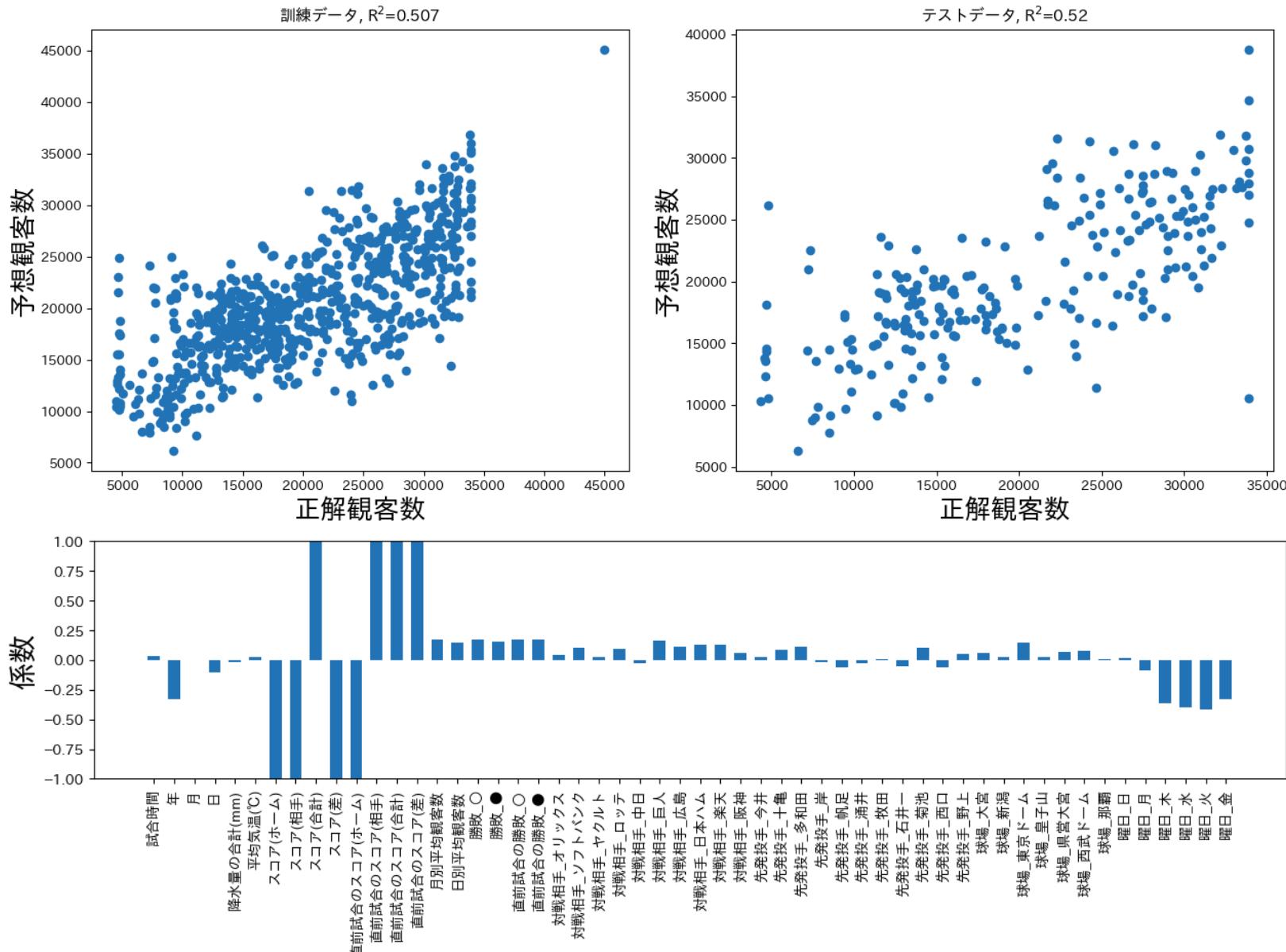
HeatMap ヒートマップ



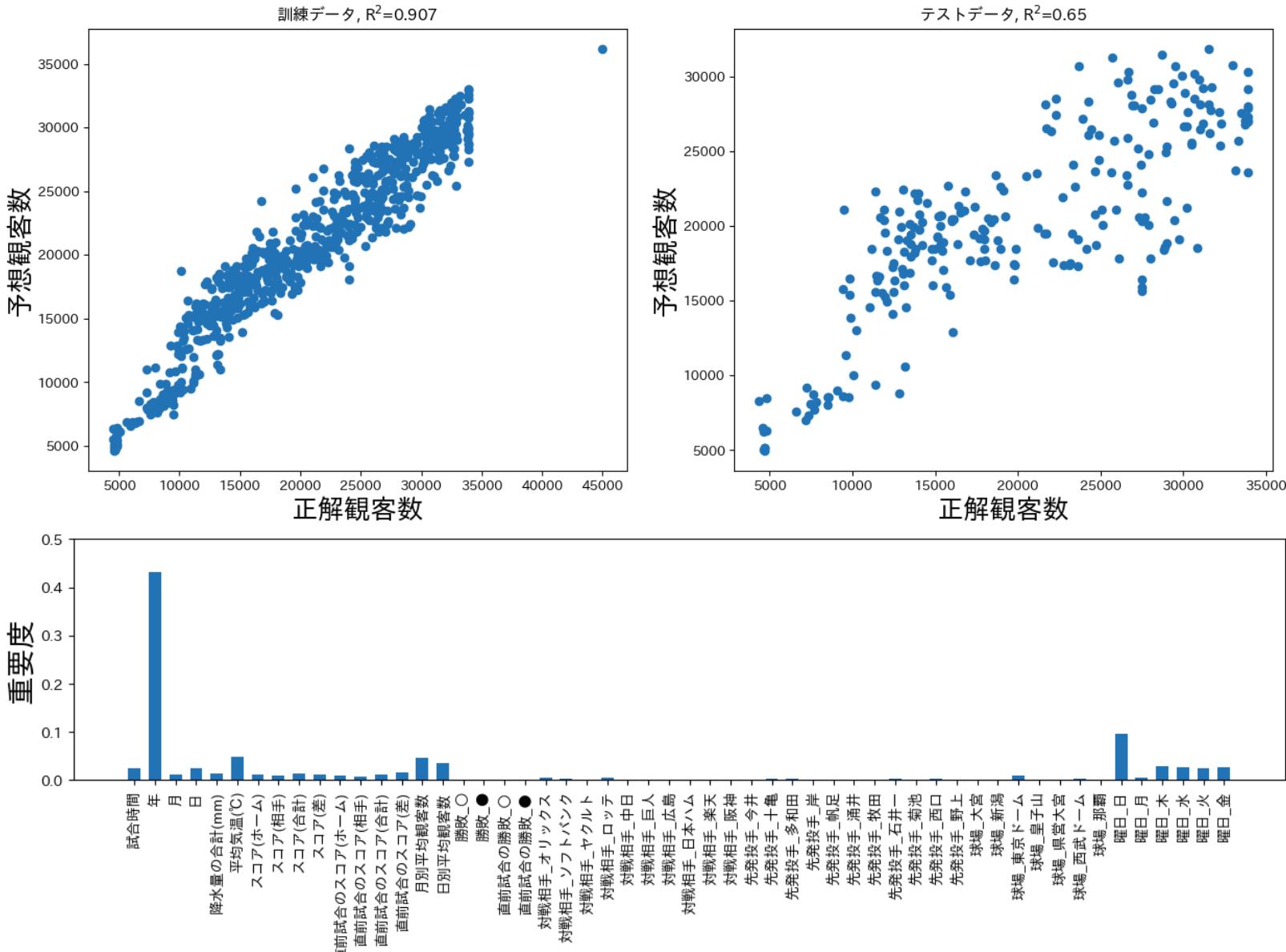
PairPlot 散布図行列



分析と重要指標理解 (Linear Regression as a Base Model)



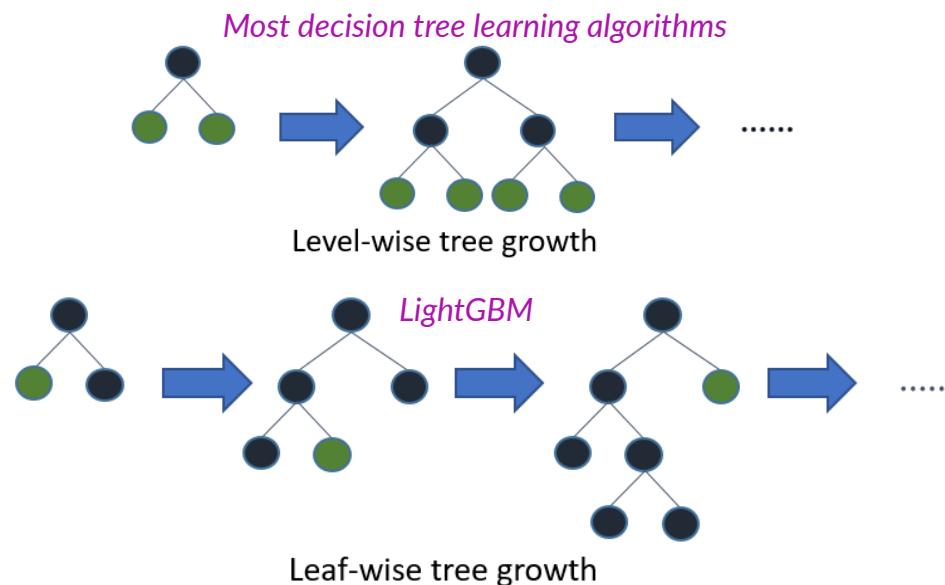
分析と重要指標理解 (RandomForest with GridSearchCV)



分析と重要指標理解 (LightGBM with Optuna)

LightGBM: A gradient boosting framework that uses tree-based learning algorithms

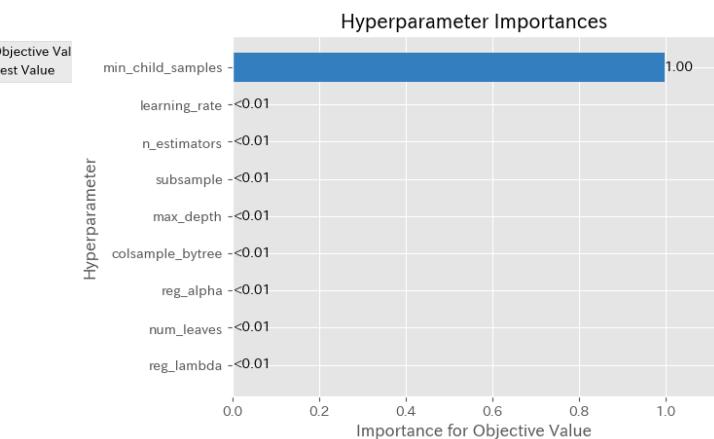
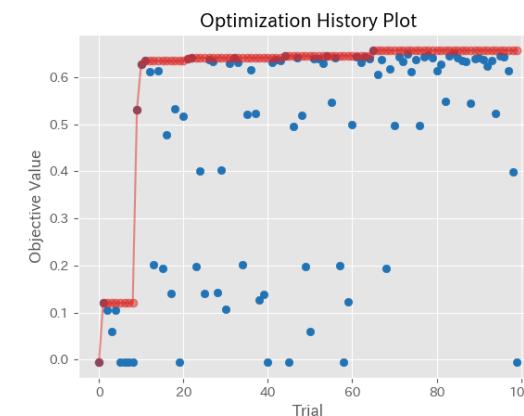
- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy
- Support of parallel, distributed, and GPU learning
- Capable of handling large-scale data



<https://lightgbm.readthedocs.io/en/stable/Features.html>

Optuna: A hyperparameter optimization framework that employs Bayesian optimization algorithms

- Lightweight, versatile, and platform agnostic architecture
- Pythonic search spaces
- Efficient optimization algorithms
- Easy parallelization
- Quick visualization



<https://optuna.readthedocs.io/en/stable/index.html>
RIKKYO UNIVERSITY
Graduate School of Artificial Intelligence and Science

分析と重要指標理解 (LightGBM with Optuna)

```
def objective(trial):
    param = {
        'metric': 'rmse',
        'n_estimators': trial.suggest_int('n_estimators', 100, 10000),
        'reg_alpha': trial.suggest_float('reg_alpha', 1e-3, 10.0, log=True),
        'reg_lambda': trial.suggest_float('reg_lambda', 1e-3, 10.0, log=True),
        'learning_rate': trial.suggest_float('learning_rate', 1e-5, 1e-2, log=True),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.3, 1.0),
        'subsample': trial.suggest_float('subsample', 0.3, 1.0),
        'max_depth': trial.suggest_categorical('max_depth', [10, 20, 50, 80, 100]),
        'num_leaves' : trial.suggest_int('num_leaves', 10, 1000),
        'min_child_samples': trial.suggest_int('min_child_samples', 1, 500)
    }
    lgbm = LGBMRegressor(n_jobs=-1, **param)
    callbacks=[lgb.early_stopping(100, verbose=False), lgb.log_evaluation(period=0)]
    lgbm.fit(X_train_opt, y_train_opt, eval_set=(X_valid_opt, y_valid_opt), callbacks=callbacks)
    y_val_opt_preds = lgbm.predict(X_valid_opt)
    return r2_score(y_valid_opt, y_val_opt_preds)
```

✓ 0.0s

Python

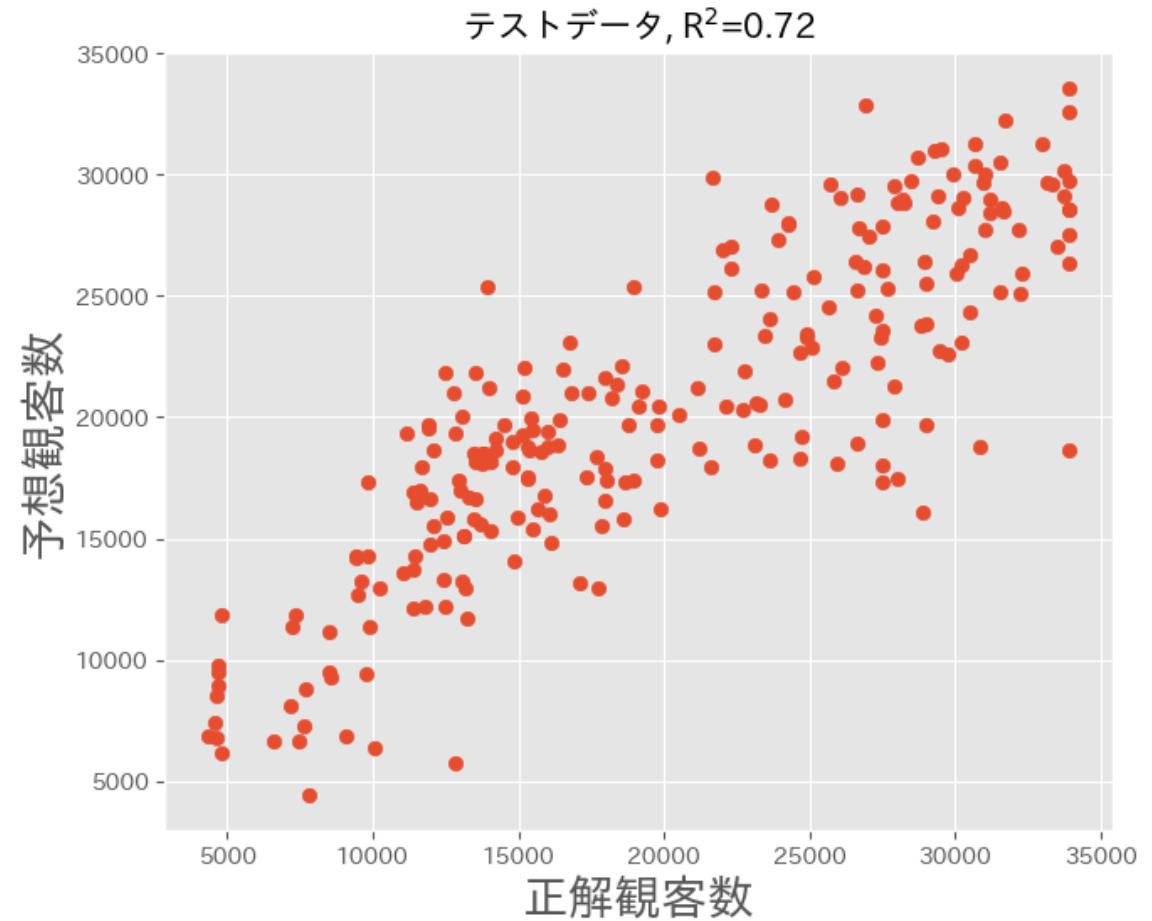
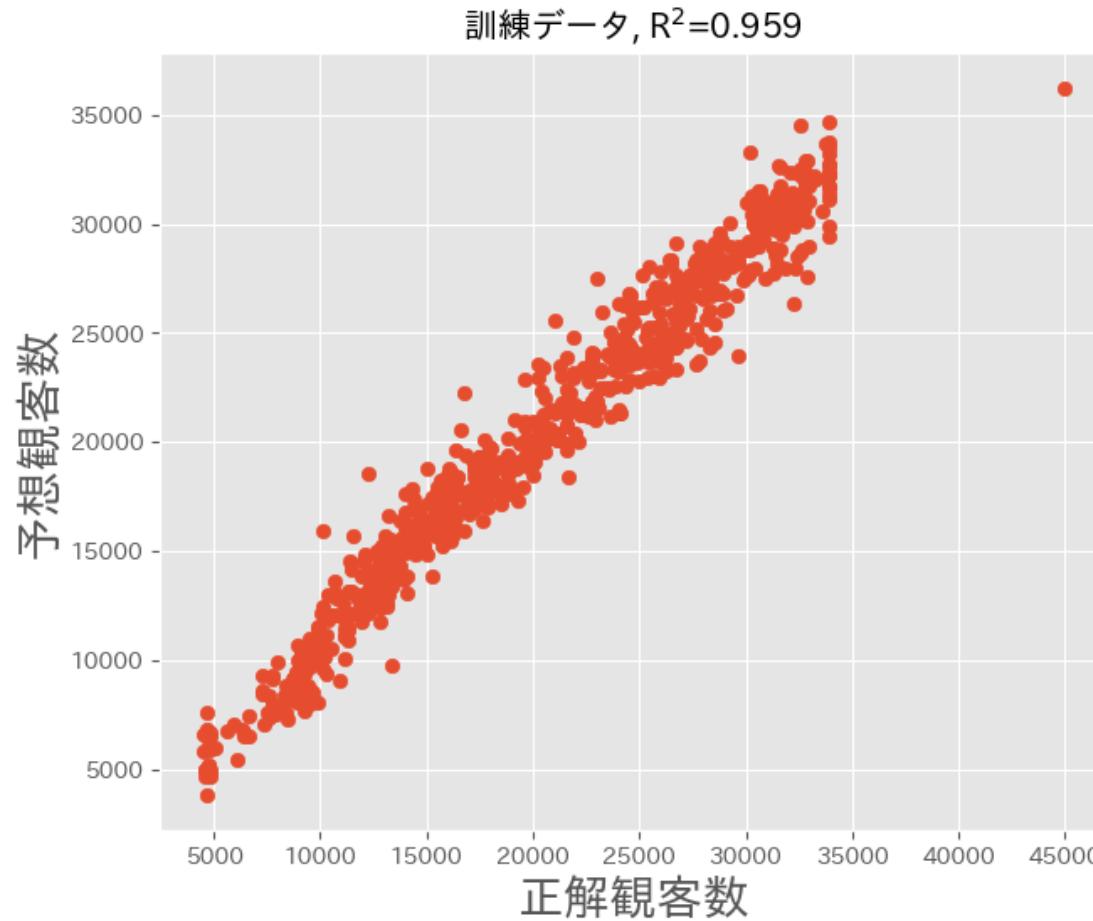
```
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

✓ 5m 39.0s

Python

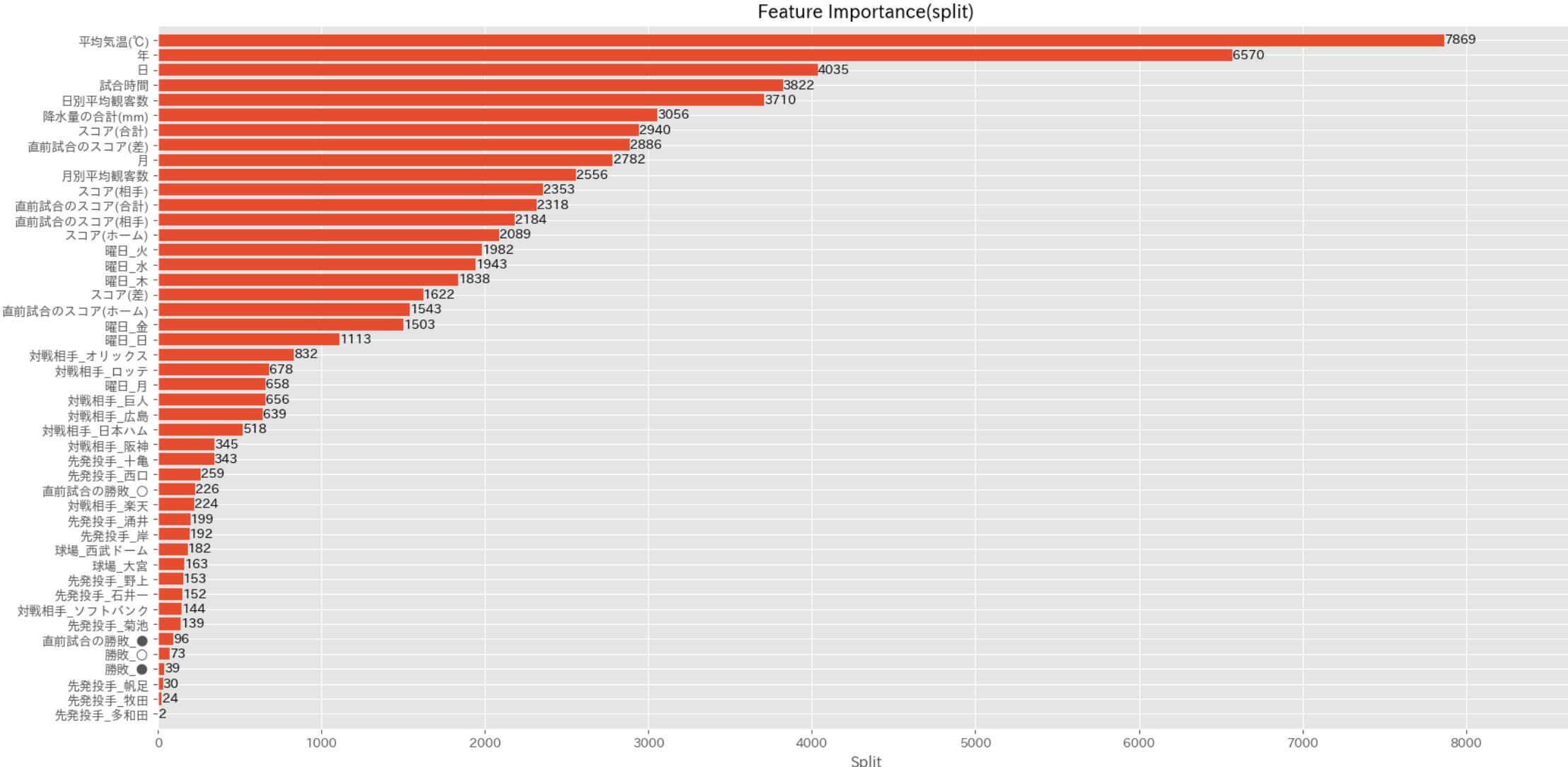
```
[I 2023-06-23 15:59:45,654] Trial 70 finished with value: 0.4974501050055445 and parameters: {'n_estimators': 6021, 'reg_alpha': 0.1494622501575405, 'reg_lambda': 0.003601407030873869, 'learning_rate': 0.0045229}
[I 2023-06-23 15:59:46,978] Trial 71 finished with value: 0.6428820865665401 and parameters: {'n_estimators': 7508, 'reg_alpha': 0.27406432883908605, 'reg_lambda': 0.003601407030873869, 'learning_rate': 0.0045229}
[I 2023-06-23 15:59:48,680] Trial 72 finished with value: 0.6337434044020472 and parameters: {'n_estimators': 6838, 'reg_alpha': 0.2668755073838498, 'reg_lambda': 0.003722942303338194, 'learning_rate': 0.00447126}
[I 2023-06-23 15:59:50,253] Trial 73 finished with value: 0.6479510691367751 and parameters: {'n_estimators': 7621, 'reg_alpha': 0.4518603998700136, 'reg_lambda': 0.001055678867438189, 'learning_rate': 0.00544759}
[I 2023-06-23 15:59:55,608] Trial 74 finished with value: 0.6108263316801865 and parameters: {'n_estimators': 7600, 'reg_alpha': 0.404133962701875, 'reg_lambda': 0.0010815983364903276, 'learning_rate': 0.006004365}
[I 2023-06-23 15:59:57,409] Trial 75 finished with value: 0.6379283552308659 and parameters: {'n_estimators': 8021, 'reg_alpha': 0.8001181927889423, 'reg_lambda': 0.001310500833664901, 'learning_rate': 0.007486401}
[I 2023-06-23 15:59:59,440] Trial 76 finished with value: 0.4968538278078716 and parameters: {'n_estimators': 5121, 'reg_alpha': 0.19984728541983465, 'reg_lambda': 0.002028743413766415, 'learning_rate': 0.00325991}
[I 2023-06-23 16:00:01,098] Trial 77 finished with value: 0.6423828385717241 and parameters: {'n_estimators': 6003, 'reg_alpha': 0.2605180062134029, 'reg_lambda': 0.001030557721974031, 'learning_rate': 0.004884538}
[I 2023-06-23 16:00:02,822] Trial 78 finished with value: 0.6467107487701773 and parameters: {'n_estimators': 5862, 'reg_alpha': 0.2684699068380657, 'reg_lambda': 0.0014167507397810084, 'learning_rate': 0.0050065}
[I 2023-06-23 16:00:04,087] Trial 79 finished with value: 0.6409871387667284 and parameters: {'n_estimators': 5934, 'reg_alpha': 0.12243677821944854, 'reg_lambda': 0.0010442669160556483, 'learning_rate': 0.0053394}
[I 2023-06-23 16:00:06,483] Trial 80 finished with value: 0.6124238016906647 and parameters: {'n_estimators': 6358, 'reg_alpha': 0.45801845854464984, 'reg_lambda': 0.0014146098324986478, 'learning_rate': 0.0087643}
[I 2023-06-23 16:00:07,906] Trial 81 finished with value: 0.6280498796762823 and parameters: {'n_estimators': 7042, 'reg_alpha': 0.2391785706474507, 'reg_lambda': 0.0010143473742818129, 'learning_rate': 0.00505583}
[I 2023-06-23 16:00:34,478] Trial 82 finished with value: 0.5490338182212506 and parameters: {'n_estimators': 6586, 'reg_alpha': 0.3164571157375383, 'reg_lambda': 0.001614232152540878, 'learning_rate': 0.005989761}
[I 2023-06-23 16:00:36,186] Trial 83 finished with value: 0.6445050491094573 and parameters: {'n_estimators': 6275, 'reg_alpha': 0.16236015445623417, 'reg_lambda': 0.003011420004033353, 'learning_rate': 0.00436133}
[I 2023-06-23 16:00:37,851] Trial 84 finished with value: 0.6506051483103439 and parameters: {'n_estimators': 5655, 'reg_alpha': 0.16186741463126192, 'reg_lambda': 0.0030799220963698636, 'learning_rate': 0.0042837}
[I 2023-06-23 16:00:39,498] Trial 85 finished with value: 0.6417882697545344 and parameters: {'n_estimators': 5728, 'reg_alpha': 0.16252926079588217, 'reg_lambda': 0.0033538030411901973, 'learning_rate': 0.0039218}
[I 2023-06-23 16:00:40,510] Trial 86 finished with value: 0.6350052184896491 and parameters: {'n_estimators': 4947, 'reg_alpha': 0.10764691683037311, 'reg_lambda': 0.0019988165371062133, 'learning_rate': 0.0084068}
```

分析と重要指標理解 (LightGBM with Optuna)



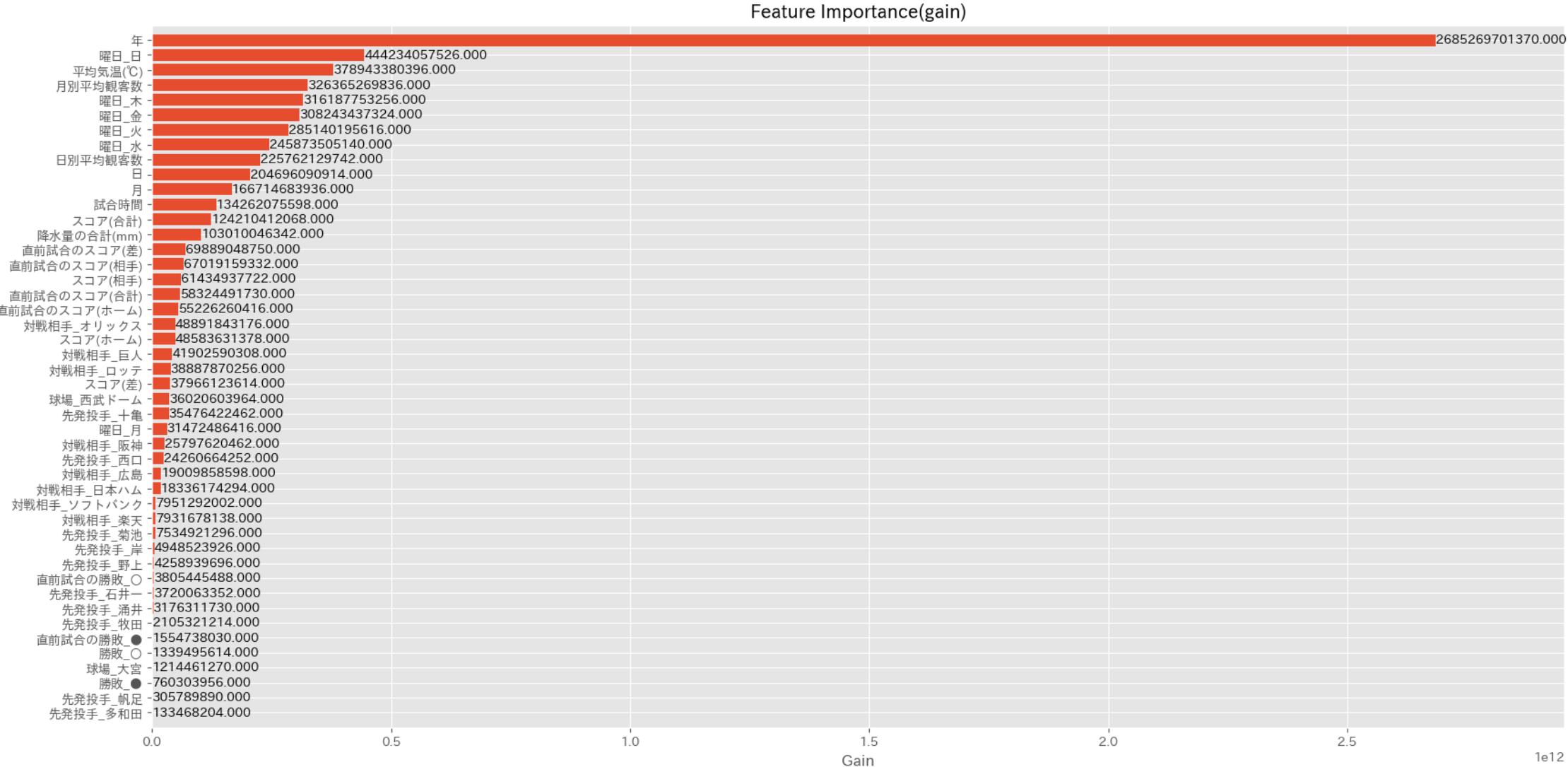
分析と重要指標理解 (LightGBM with Optuna)

Split: result contains numbers of times the feature is used to split the data across all trees.



分析と重要指標理解 (LightGBM with Optuna)

Gain: result contains the average training loss reduction gained when using a feature for splitting.



分析と重要指標理解 (SHAP)

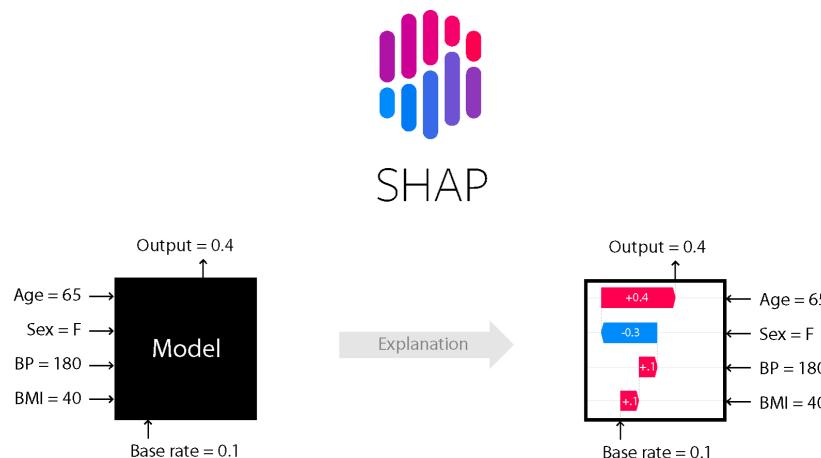
What makes a measure of feature importance good or bad?

1. Consistency

- Whenever we change a model such that it relies more on a feature, then the attributed importance for that feature should not decrease.

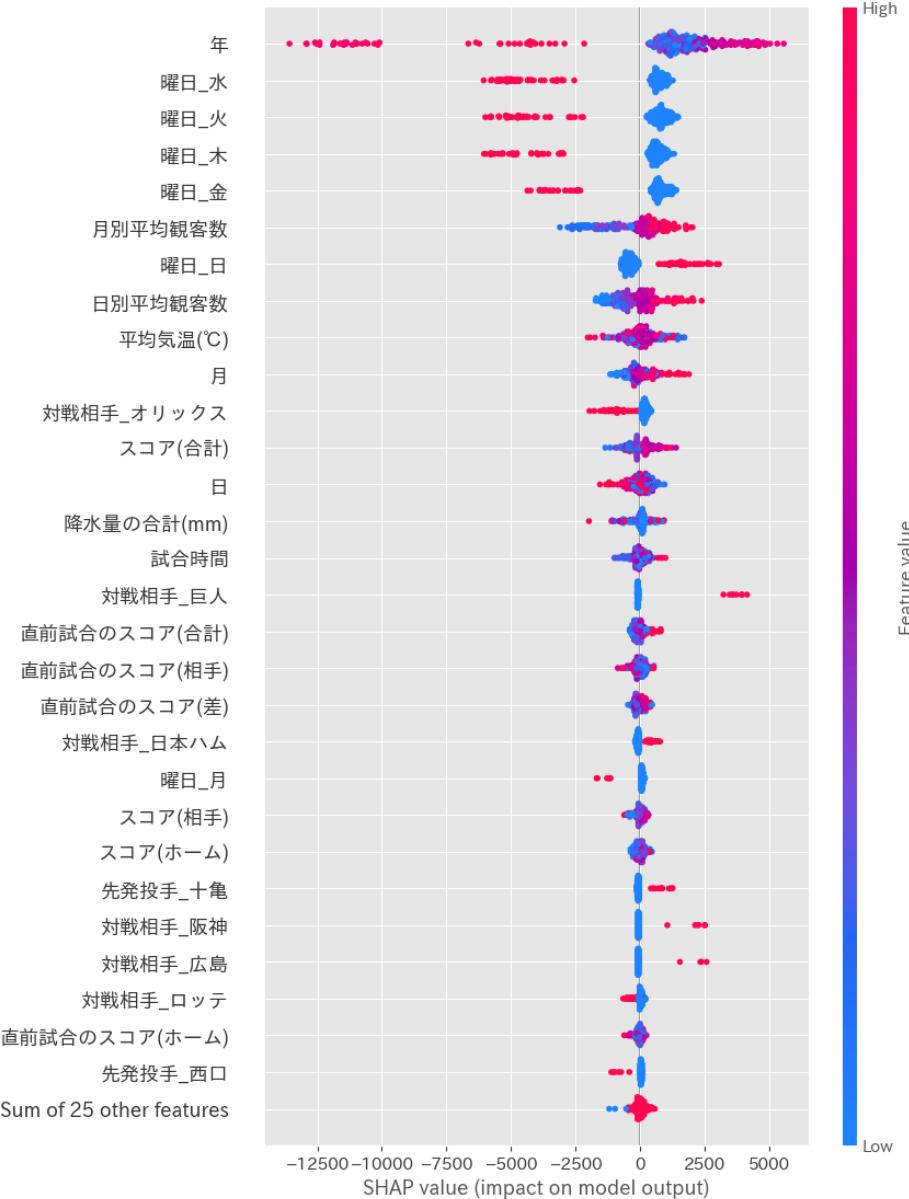
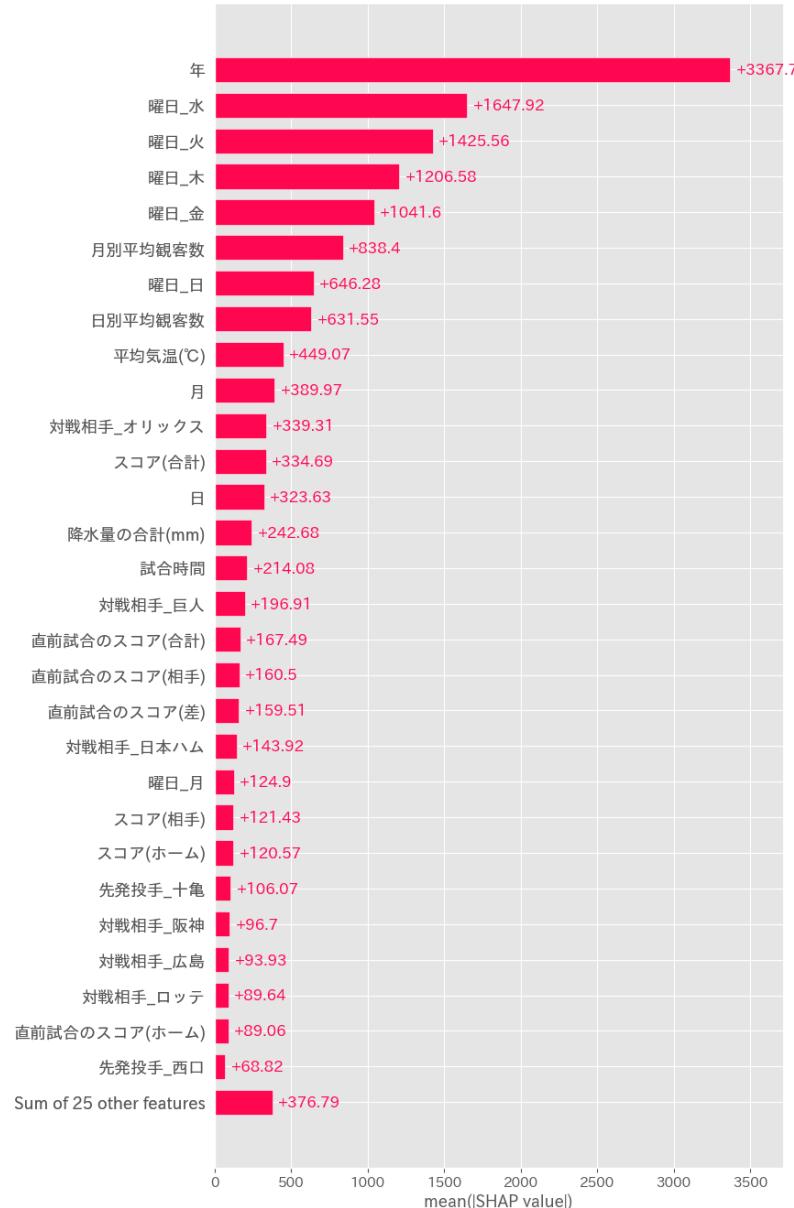
2. Accuracy

- The sum of all the feature importances should sum up to the total importance of the model. (For example if importance is measured by the R^2 value then the attribution to each feature should sum to the R^2 of the full model)



*SHAP is both consistent
and accurate.*

分析と重要指標理解 (SHAP)



Lightgbm without one hot encoding using categorical feature support

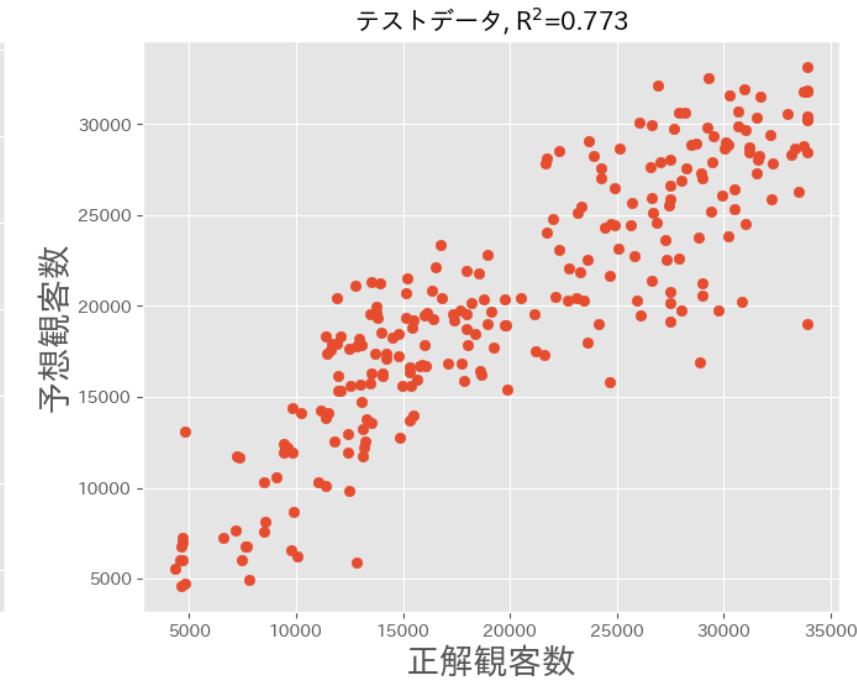
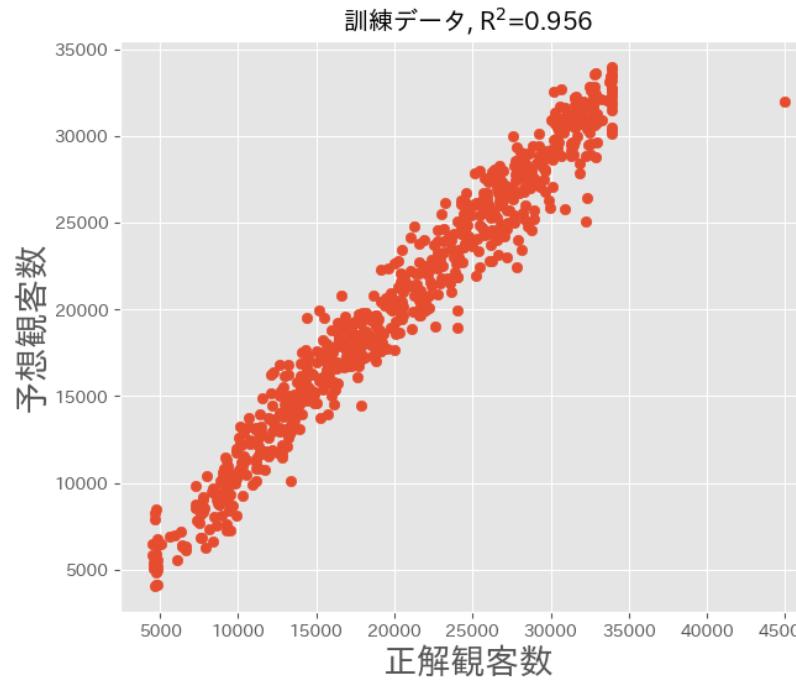
Optimal Split for Categorical Features

It is common to represent categorical features with one-hot encoding, but this approach is suboptimal for tree learners. Particularly for high-cardinality categorical features, a tree built on one-hot features tends to be unbalanced and needs to grow very deep to achieve good accuracy.

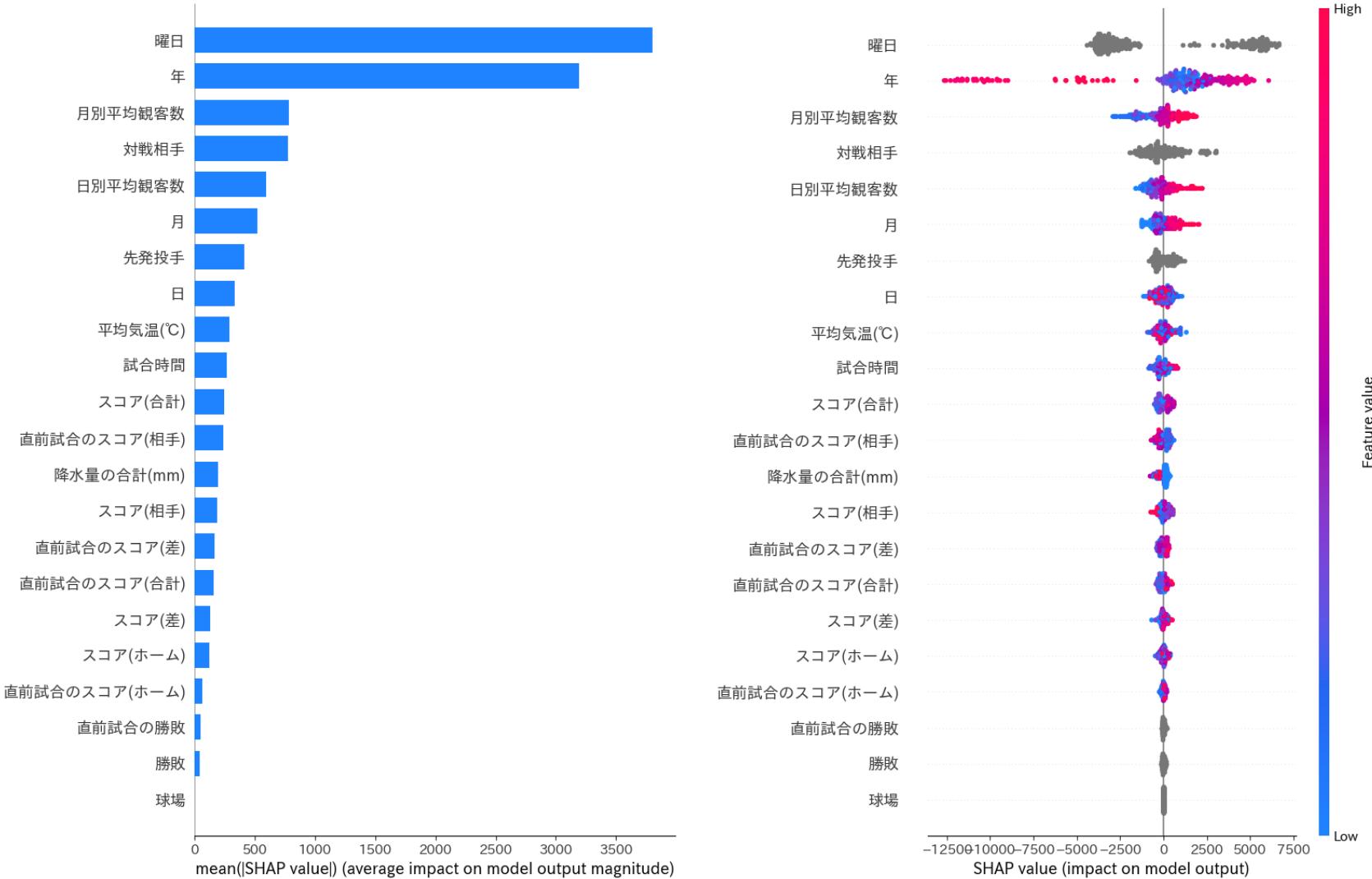
Instead of one-hot encoding, the optimal solution is to split on a categorical feature by partitioning its categories into 2 subsets. If the feature has k categories, there are $2^{k-1} - 1$ possible partitions. But there is an efficient solution for regression trees[8]. It needs about $O(k * \log(k))$ to find the optimal partition.

The basic idea is to sort the categories according to the training objective at each split. More specifically, LightGBM sorts the histogram (for a categorical feature) according to its accumulated values (`sum_gradient` / `sum_hessian`) and then finds the best split on the sorted histogram.

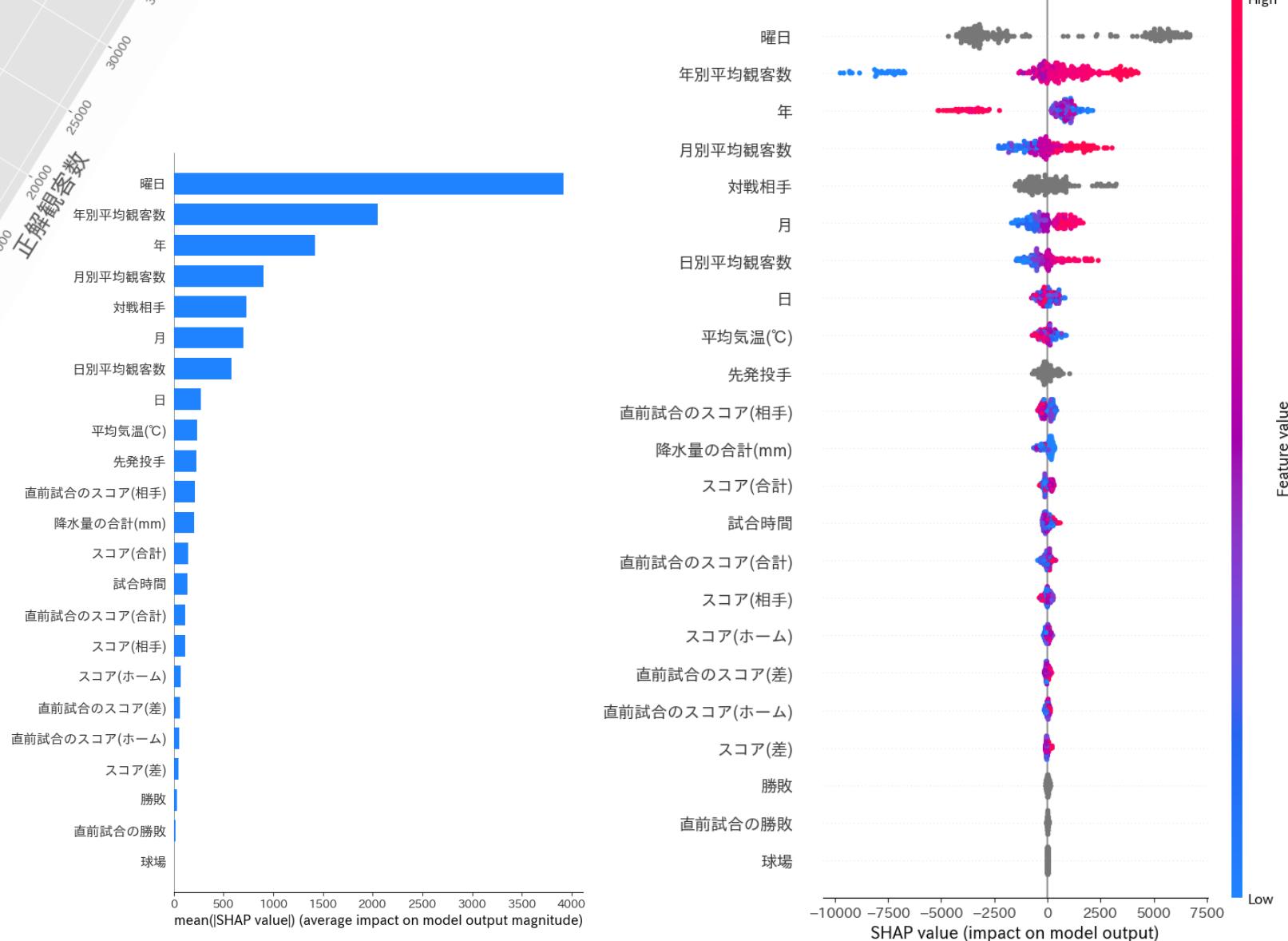
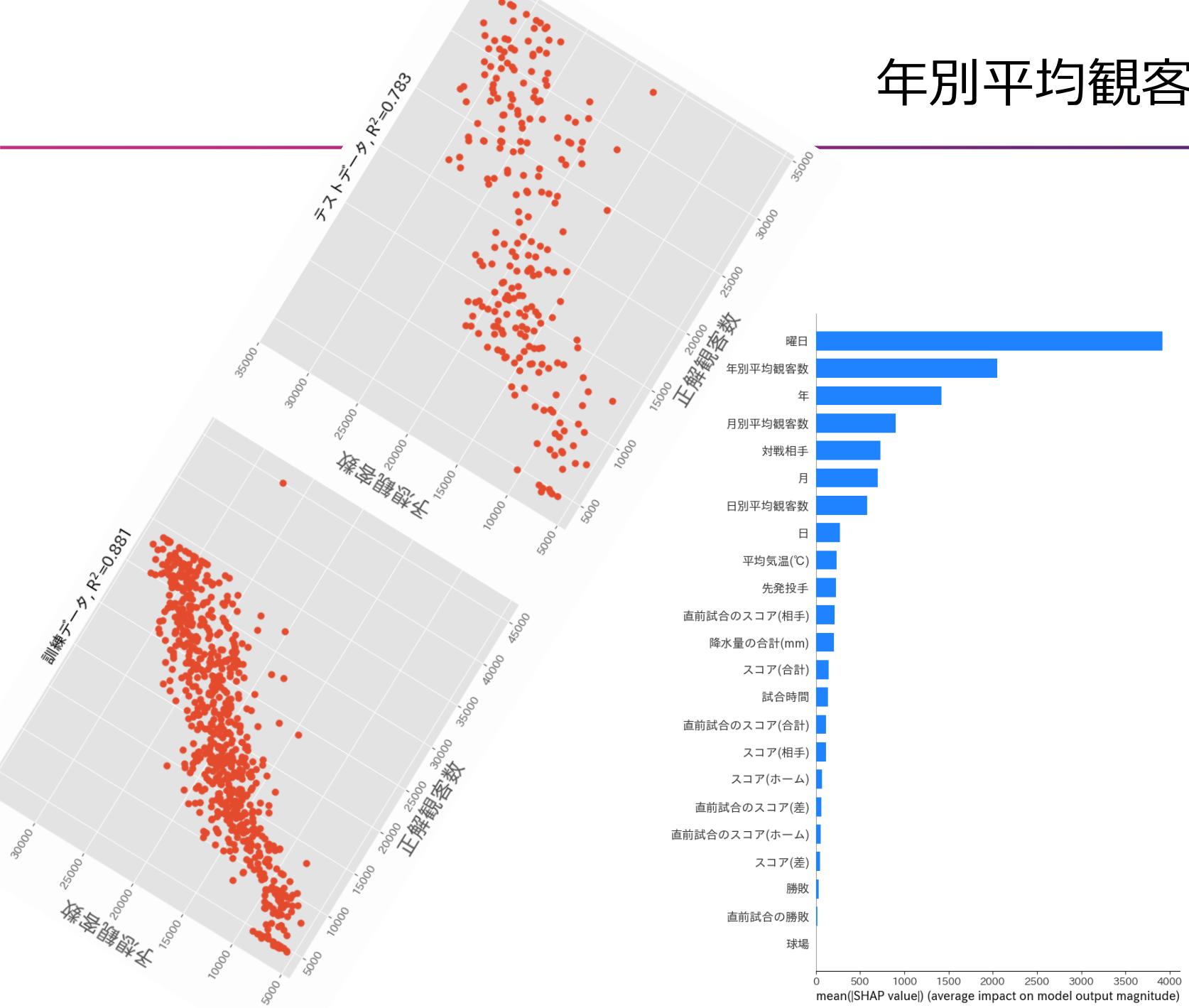
<https://lightgbm.readthedocs.io/en/latest/Features.html#optimal-split-for-categorical-features>



Lightgbm without one hot encoding using categorical feature support



年別平均観客数特徴量の追加





立教大学大学院 人工知能科学研究科

RIKKYO UNIVERSITY
Graduate School of Artificial Intelligence and Science