STØTTET AF
Promilleafgiftsfonden for landbrug

Predicting Dry Matter in Maize fields Using Machine Learning

Contact: SEGES-datascience@seges.dk

Data Acquisition and Preprocessing

In this project, our focus is on a dataset compiled from maize yield registrations spanning seven years, from 2016 to 2022. The dataset includes the following files:

- `Majs_Udbytteregistreringer_2016_2019.xlsx`
- `Majs_Udbytteregistreringer_2020.xlsx`
- `Majs_Udbytteregistreringer_2021.xlsx`
- `Majs_Udbytteregistreringer_2022.xlsx`

Each dataset has been subjected to extensive preprocessing steps, including the rectification of duplicates, imputation of missing data, and identification and handling of outliers.

Data Processing and Analysis Workflow

Data Loading and Merging

The initial stage involved loading each raw data file into a data processing environment. Necessary transformations and formatting adjustments were made to facilitate the merging of these files into a consolidated dataset.

Data Cleaning and Quality Assurance

Following the merging process, a comprehensive data cleaning procedure was undertaken. This process comprised:

- Calculating a weighted average for fields that were divided into separate parts, followed by merging these segments into a singular field.
- Exclusion of fields lacking geographical information.
- Evaluation and management of potential outliers in dry matter registration data.

Experiment Structure

Target Variable

The target variable in this study is a floating-point value ranging from 0 to 100. It represents the percentage of dry matter measured at the time of harvest.

Temporal and Static Features

Temporal Features

The model incorporates a variety of features categorized into distinct groups based on their nature and source. Temporal features, which are measured multiple times throughout the growing season, and static features, which remain constant for each field, are the two primary categories.

Satellite Data

We employ Sentinel 2 satellite imagery, specifically the Level 1C product. This imagery includes 11 distinct spectral bands, capturing various aspects of the earth's surface:

- **B1 (Coastal Aerosol Band)**
- **B2 (Blue Band)**
- **B3 (Green Band)**
- **B4 (Red Band)**
- **B5 to B7 (Vegetation Red Edge Bands)**
- **B8 (NIR Band)**
- **B8A (Narrow NIR Band)**
- **B9 (Water Vapor Band)**
- **B11 and B12 (SWIR Bands)**

We also utilize vegetation indices such as:

- **NDVI (Normalized Difference Vegetation Index)**
- **NDRE (Normalized Difference Red Edge Index)**
- **MSAVI (Modified Soil Adjusted Vegetation Index)**

For each band and field, we extract all available images from the harvest year. Linear interpolation is used to estimate the feature value on the 1st and 15th of each month from May to November. We also calculate the relative change from May 1st throughout the season for each feature.

Climate Data

Climate data is sourced from DMI's nearest weather station to each field. The data includes:

- **Daily minimum, maximum, and mean temperature**
- **Daily mean radiation**
- **Daily precipitation**
- **Daily mean humidity**
- **Daily mean wind speed and direction**
- **Daily mean temperature 10cm below ground**
- **Daily mean leaf moisture**

Additional features derived from mean daily temperature:

- **T8**: Temperature minus 8 if above 8°C, else 0.
- **T0_8**: Temperature if between 0°C to 8°C, else 0.
- **T8_12**: Temperature if between 8°C to 12°C, else 0.
- **T12**: Temperature minus 12 if above 12°C, else 0.
- **MVE (Majsvarmeenheder)**: A specific thermal unit calculation.

For dynamic features, any measurement post-harvest is set to None.

Static Features

Terrain Data

Terrain data includes:

- **Field Height Mean**: Average height of the field above surrounding terrain in meters.
- **Field Relative Height Mean**: Comparative height of the field.
- **Gradient Slope Mean and Degrees**: Slope characteristics of the field.
- **Aspect Degrees**: Directional aspect of the field.

General Data

General field data encompasses:

- **Crop (Pre) Codes**: Field crops from the past 5 years.
- **Soil Type**: Type of soil in the field.
- **Variety Code**: Specific variety of the crop planted.

Models

Model Employed

- **Gradient Tree Boosting Model (XGBoost)**

Evaluation Structure and Metrics

- **Training Method**: The model was trained using a squared loss function.
- **Evaluation Metric**: Performance was evaluated based on the Mean Absolute Error (MAE).

Experiment

- **Feature Selection**: Initially, features were filtered by date and type. We excluded features observed after the harvest, using only those measured before September 1. Trials with post-September 1 features did not yield performance improvements.
- **Model Configurations**: Various combinations of features and hyperparameters were explored.
- **Validation Method**: K-fold cross-validation was used, with 'k' representing data from a unique year. Each fold involved training the model on data excluding fields from year 'k' and testing on that year, ensuring each year's data served once as the validation set.

- **Performance Assessment**: The Mean Absolute Error (MAE) was calculated for each year. The average MAE across all years was used to determine overall model performance.
- **Overfitting Prevention**: Early stopping based on in-sample data performance was implemented.
- **Model Selection**: The model with the lowest MAE in cross-validation was chosen as the best performer, especially effective in estimating dry matter percentage across various untrained fields.
- **Feature Importance Assessment**: The 'gain' from each fold was averaged for each feature across all folds to identify the most influential features.

Conclusion

Based on the experiment results, we conclude:

- There is a misalignment between the model's output and the available training data, as the dry matter value is only known at harvest.
- The model tends to predict the mean target value for the respective year.
- Due to weak signal and data bias towards optimal harvest dates, the model struggles to accurately detect the target value.
- Relative band features from satellite data proved less relevant than climate features where precipation shown the largest gain.

Future Work

The following areas are interesting for future work:

Data Improvement

- **Seasonal Trial Data**: Incorporating data from trials throughout the growing season is essential for improved model performance.
- **Variety Implications**: Further investigation into how variety information affects predictions could enhance model accuracy.
- **Outlier Dection**: Further assess outliers across varieties, in particular for less observed varieties, are relevant to consider.

Technical Setup

In this notebook the py39_pla_hoesttid_majs conda environment is used. It can be installed using the environment.yml file located in the GitHub repository.

Implementation and Analysis

In the following we show the final analysis and overview of the data used.

Notebook Configuration and Initial Setup

In [ ]:
```
import re
import warnings
import itertools
import pprint
import numpy as np
import pandas as pd
import geopandas as gpd
from pathlib import Path
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
import pprint
import sklearn
import xgboost as xgb
from collections import defaultdict
```

In [ ]:
```
# Set figure size for the notebook
plt.rcParams["figure.figsize"]=(10,6)
plt.rcParams['figure.dpi']=100

# set seaborn whitegrid theme
sns.set(style="whitegrid")
palette = "Set1"

# ignore future warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

Exploratory Data Analysis (EDA)

In [ ]:
```
data_folder = Path('data/final-data/') #local data folder
```

In [ ]:
```
df_training_data = pd.read_parquet(data_folder / 'dataset_full_with_nans.parquet')
```

In [ ]:
```
print('The training dataset consists of {} field-ids across {} farms'.format(
     df_training_data.shape[0], df_training_data['farmId'].nunique()
    )
)
```

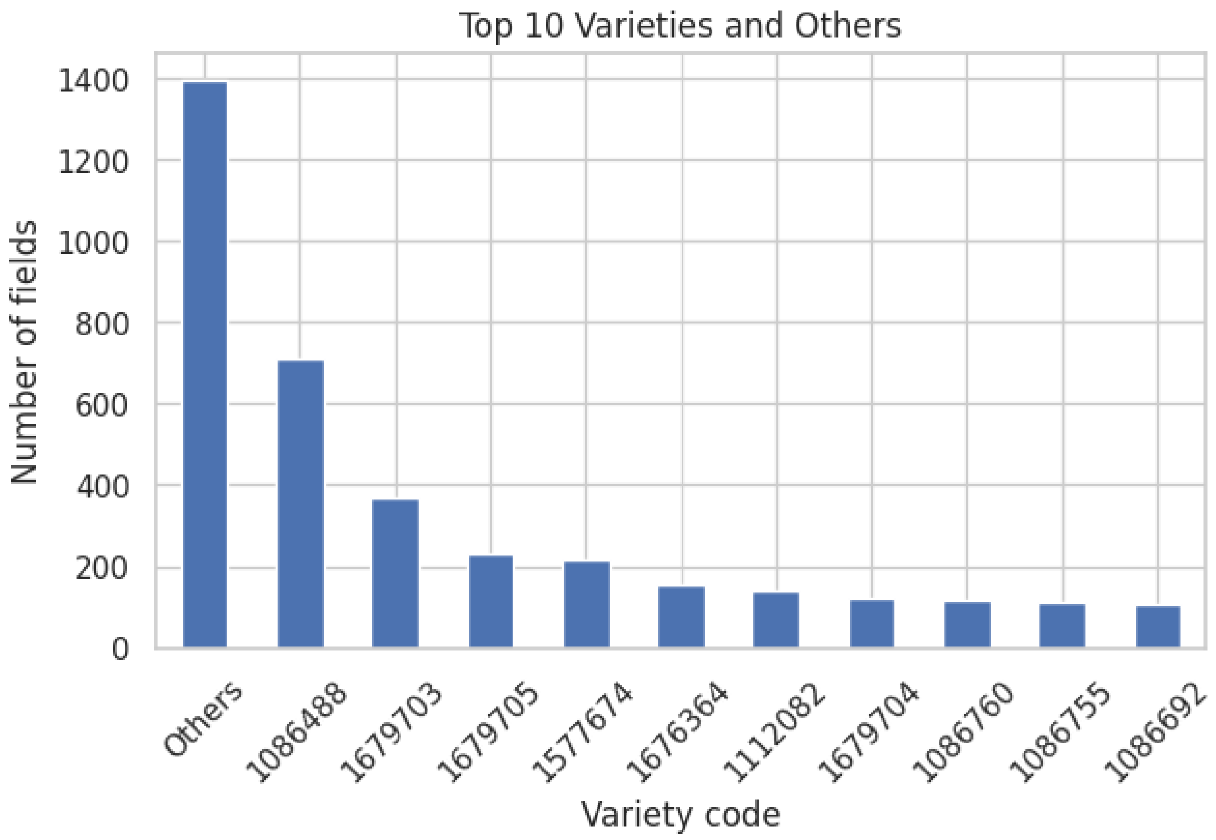
```
The training dataset consists of 3913 field-ids across 299 farms
```

In [ ]:
```
print('The training dataset contains {} unique varieties'.format(
    df_training_data['variety_code'].nunique()
    )
)
```

```
The training dataset contains 118 unique varieties
```

In [ ]:
```
# Display number of fields for variety-groups. Display only groups with above 100 fie

# Copy training data and remove NaNs for variety code
df_variety_count = df_training_data.copy()
df_variety_count = df_variety_count[~df_variety_count['variety_code'].isna()]
df_variety_count['variety_code'] = df_variety_count['variety_code'].astype(int)
# Number of top varieties to display
```

```
N = 10

# Count the occurrences of each variety
variety_counts = df_variety_count['variety_code'].value_counts()

# Identify the top N varieties
top_varieties = variety_counts.head(N).index

# Label the varieties not in the top N as 'Others'
df_variety_count['variety_grouped'] = df_variety_count['variety_code'].apply(
    lambda x: x if x in top_varieties else 'Others'
    )

# Aggregate the counts again, now with 'Others' included
grouped_counts = df_variety_count['variety_grouped'].value_counts()

# Create the bar plot
plt.figure(figsize=(7, 4))
grouped_counts.plot(kind='bar')
plt.xlabel('Variety code')
plt.ylabel('Number of fields')
plt.title(f'Top {N} Varieties and Others')
plt.xticks(rotation=45)  # Rotate labels for better readability
plt.show()
```



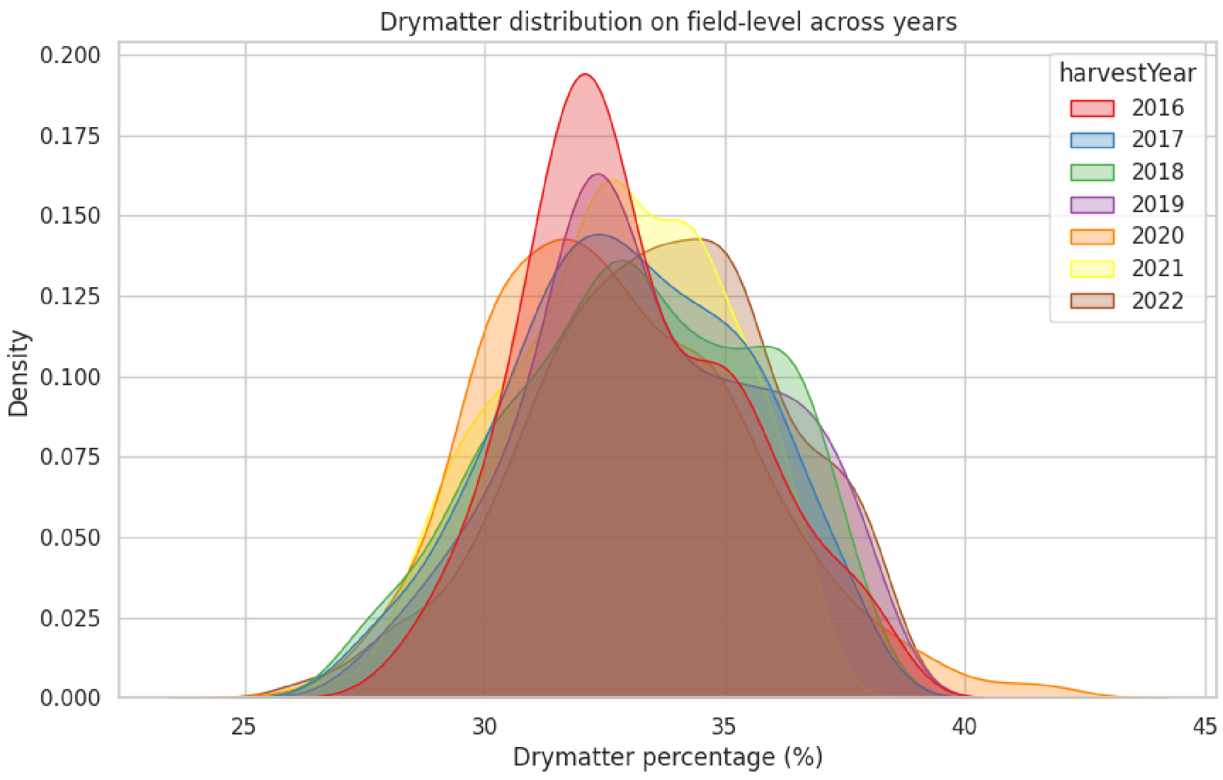
Remove drymatter-outliers across varieties

We use IQR for outlier-detection but only for varieties with +100 fields observed.

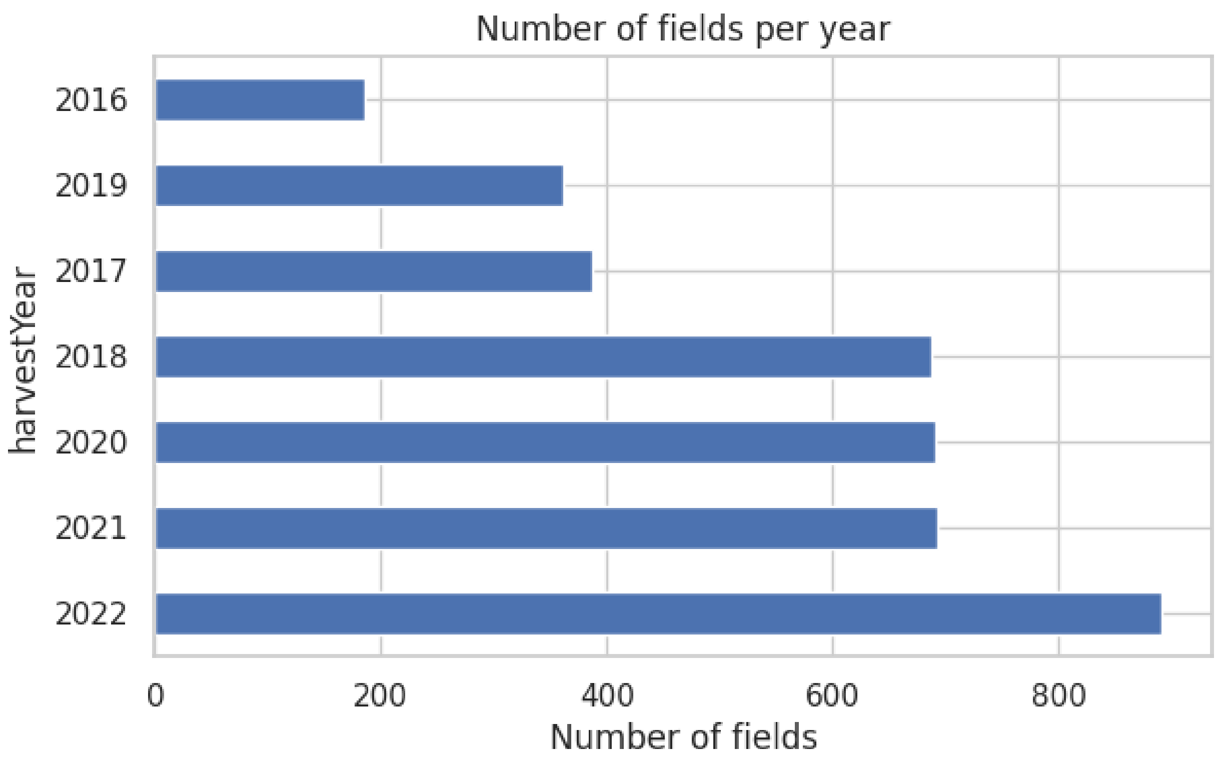
In [ ]:
```
# Function to remove outliers based on IQR
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
```

```
In [ ]: # Find the top 10 'variety_code' groups
        top_varieties = df_training_data['variety_code'].value_counts().head(10).index

        # Separate the top groups and the rest
        top_groups = df_training_data[df_training_data['variety_code'].isin(top_varieties)]
        rest_groups = df_training_data[~df_training_data['variety_code'].isin(top_varieties)

        # Apply the outlier removal to the top groups
        top_groups_filtered = top_groups.groupby('variety_code').apply(
            lambda x: remove_outliers(x, 'drymatter_per')).reset_index(drop=True)

        # Combine the results
        df_training_data_filtered = pd.concat([top_groups_filtered, rest_groups]).reset_inde
```

```
In [ ]: sns.kdeplot(df_training_data_filtered, x='drymatter_per', hue='harvestYear', fill=Tr
                        common_norm=False, alpha=0.3, bw_adjust=1, palette=palette)
        plt.title('Drymatter distribution on field-level across years')
        _ = plt.xlabel('Drymatter percentage (%)')
```

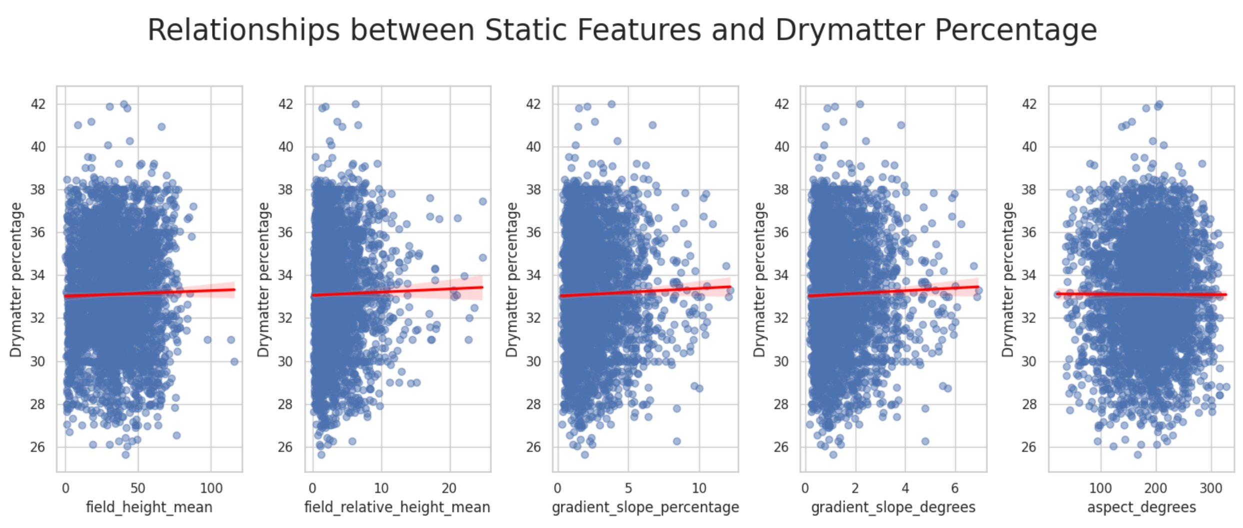


```
In [ ]: df_training_data_filtered['harvestYear'].value_counts().plot(kind='barh', figsize=(7
        plt.title('Number of fields per year')
        _ = plt.xlabel('Number of fields')
```



Static features

```
In [ ]: y_var = 'drymatter_per'
        x_vars = ['field_height_mean', 'field_relative_height_mean',
                  'gradient_slope_percentage', 'gradient_slope_degrees',
                  'aspect_degrees',]

        fig, axes = plt.subplots(1, 5, figsize=(15, 6))
        axes = axes.flatten()

        for i, x_var in enumerate(x_vars):
            sns.regplot(data=df_training_data_filtered, x=x_var, y=y_var, ax=axes[i],
                        scatter_kws={'alpha':0.5}, line_kws={'color':'red'})
            axes[i].set_xlabel(x_var)
            axes[i].set_ylabel("Drymatter percentage")

        fig.suptitle('Relationships between Static Features and Drymatter Percentage',
                     fontsize=25, y=1.02)

        plt.tight_layout()
        plt.show()
```



Dynamic features

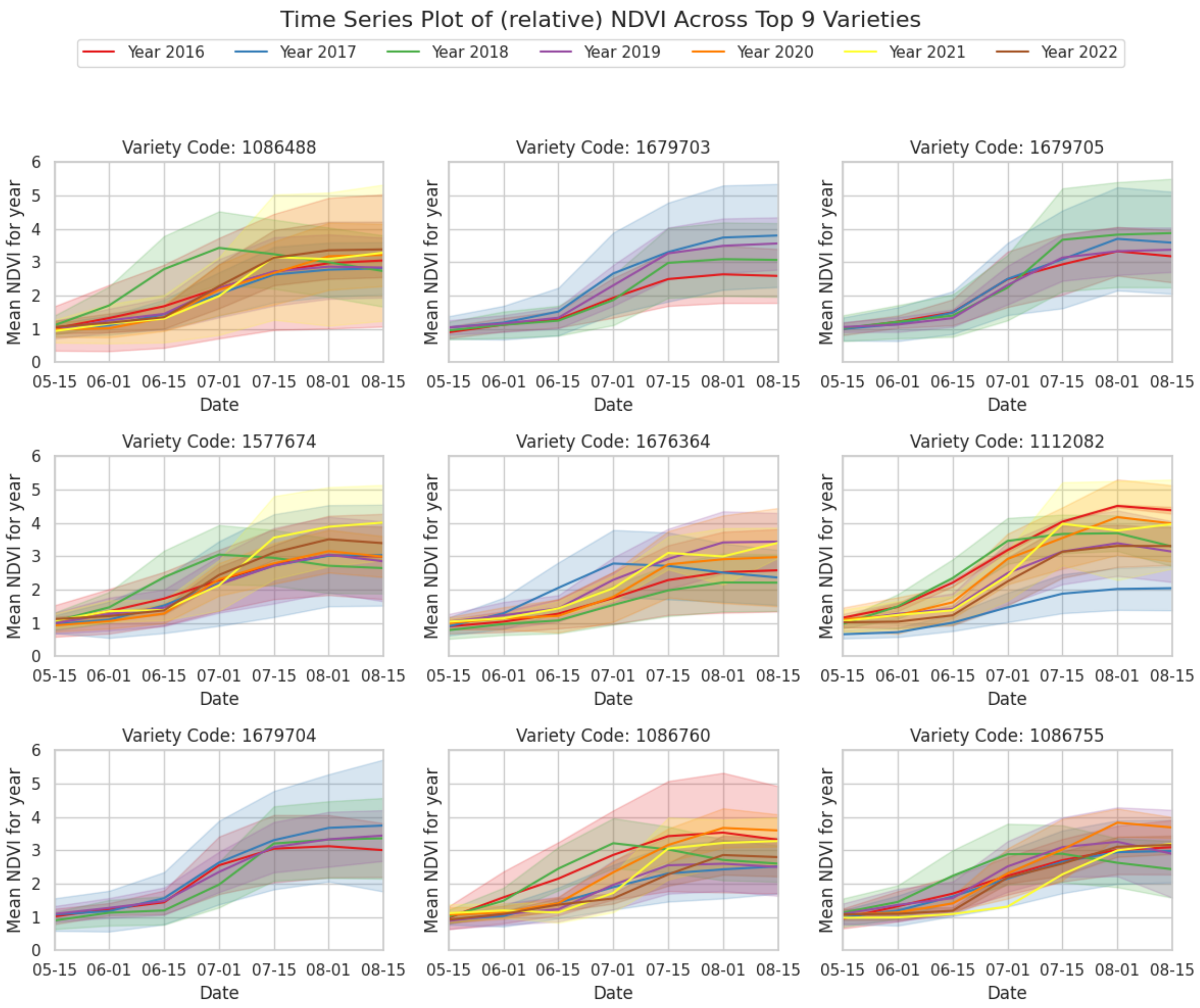
We study if the relative NDRE/NDVI is significantly different across varieties.

```
In [ ]: def filter_relative_columns(df, additional_columns):
            filtered_columns = [col for col in df.columns if 'relative' in col]
            return df[filtered_columns + additional_columns].copy()

        def preprocess_data(df):
            df = df.dropna(subset=['variety_code'])
            df['variety_code'] = df['variety_code'].astype(int).astype(str)
            df['harvestYear'] = df['harvestYear'].astype(int).astype(str)
            return df

        def extract_time_series(df, var, variety_code):
            filtered_df = df[df['variety_code'] == variety_code]
            date_columns = [col for col in filtered_df.columns if var in col and re.search(r

            time_series_data = []
            for index, row in filtered_df.iterrows():
                harvestYear = row['harvestYear']
                for col in date_columns:
                    date = re.findall(r'\d{2}-\d{2}', col)[0]
                    value = row[col]
                    time_series_data.append({'Date': date, var: value, 'harvestYear': harves

            return pd.DataFrame(time_series_data)
```

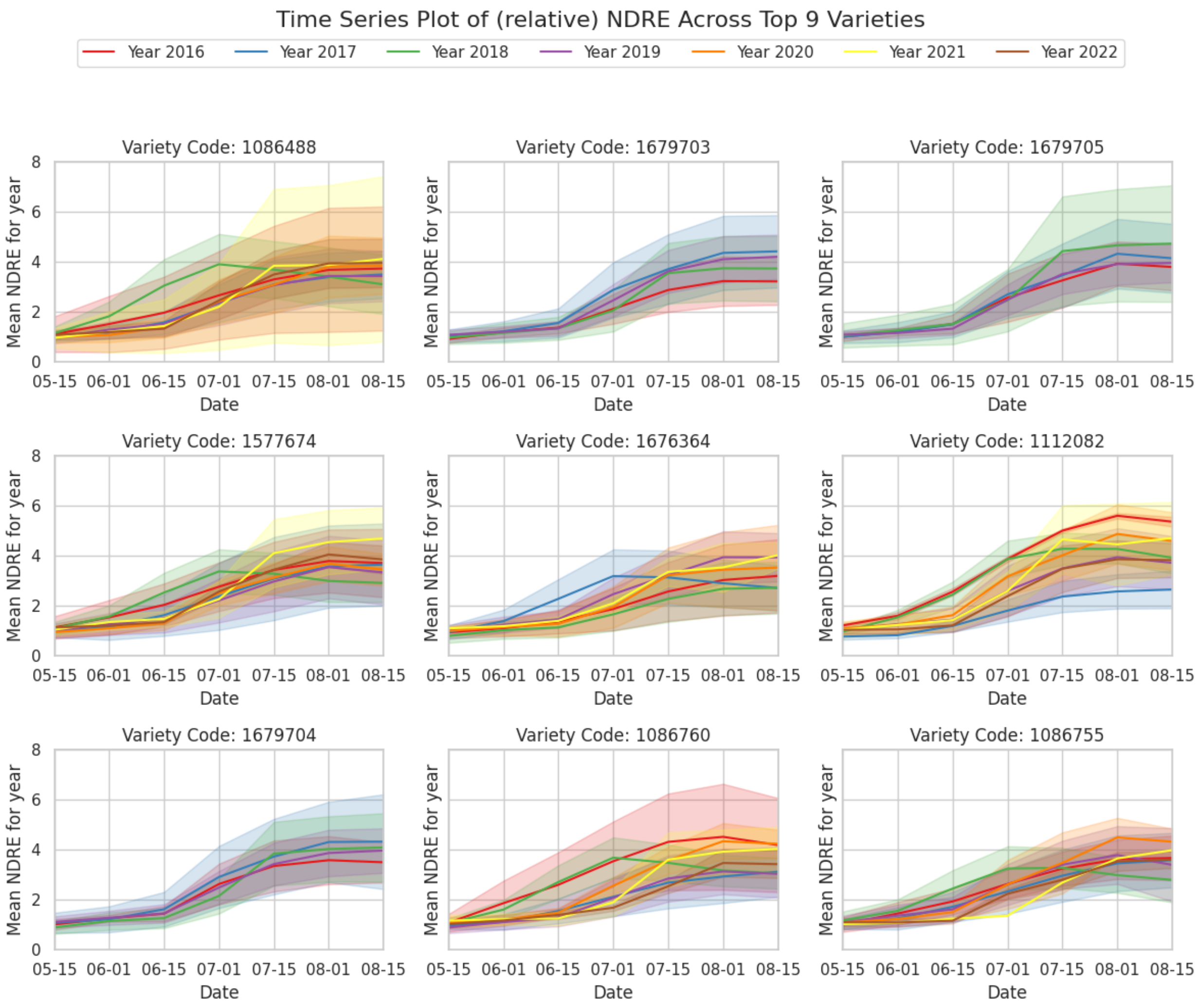
```
In [ ]: def plot_time_series(df, var, top_varieties, title, ylim):
            """Plot time series data for top varieties of a given variable."""
            palette = sns.color_palette("Set1", 7)
            fig, axs = plt.subplots(3, 3, figsize=(12, 10), sharey=True)
            axs = axs.flatten()

            for i, variety_code in enumerate(top_varieties):
                ts_data = extract_time_series(df, var, variety_code)
                unique_years = ts_data['harvestYear'].unique()
                year_color = {year: palette[i % 7] for i, year in enumerate(unique_years)}

                for year in unique_years:
                    yearly_data = ts_data[ts_data['harvestYear'] == year]
                    mean_values = yearly_data.groupby('Date')[var].mean()
                    std_values = yearly_data.groupby('Date')[var].std()

                    axs[i].plot(mean_values.index, mean_values, marker='', color=year_color[
                    axs[i].fill_between(mean_values.index, mean_values - std_values, mean_va
                                        color=year_color[year], alpha=0.2)

                axs[i].set_title(f'Variety Code: {variety_code}')
                axs[i].set_ylim(0, ylim)
                axs[i].set_xlim(0, 6)
                axs[i].set_xlabel('Date')
                axs[i].set_ylabel(f'Mean {var} for year')

            fig.suptitle(title, fontsize=16, y=1.02)
            handles, labels = axs[0].get_legend_handles_labels()
            fig.legend(handles, labels, loc='upper center', ncol=len(unique_years), bbox_to_
            plt.tight_layout(rect=[0, 0.03, 1, 0.95])
            plt.show()
```

In [ ]:
```
df_training_data_filtered_to_plot = filter_relative_columns(df_training_data_filtere
df_training_data_filtered_to_plot = preprocess_data(df_training_data_filtered_to_plo

top_varieties = df_training_data_filtered_to_plot['variety_code'].value_counts().hea
```

In [ ]:
```
# Plot for NDVI
plot_time_series(df_training_data_filtered_to_plot, 'NDVI', top_varieties,
 'Time Series Plot of (relative) NDVI Across Top 9 Varieties', ylim=6)
```



```
In [ ]: # Plot for NDRE
        plot_time_series(df_training_data_filtered_to_plot, 'NDRE', top_varieties,
         'Time Series Plot of (relative) NDRE Across Top 9 Varieties', ylim=8)
```



Feature Engineering and Extractions

```
In [ ]: def extract_date(col_name):
            match = re.search(r'(\d{2}-\d{2})', col_name)
            return match.group(0) if match else None

        def extract_name(name, feature_type):
            if feature_type == 'S2':
                match = re.search('NDVI|NDRE', name)
                matchb = re.search('relative', name)
                return True if (match is not None and matchb is not None) else False

            if feature_type == 'climate':
                match = re.search('acc_precip', name)
                return True if match else False

        def feature_selection_on_type(df, feature_type):
            if feature_type == 'S2':
                select_feature = [extract_name(col, feature_type) for col in df.columns]
            if feature_type == 'climate':
                select_feature = [extract_name(col, feature_type) for col in df.columns]
```

```
    return df.loc[:,select_feature]

def feature_selection_on_date(df, cut_off_date):
    feature_dates = [extract_date(col) for col in df]

    result_list = [datetime.strptime(cut_off_date, '%m-%d') > datetime.strptime(date
                    if date is not None else True for date in feature_dates]

    return df.loc[:,result_list]
```

In [ ]:
```
# Filter out irrelevant climate data
df_training_data_filtered = df_training_data_filtered.iloc[:,:18].merge(
    feature_selection_on_type(df_training_data_filtered, 'S2'), left_index=True, rig
        feature_selection_on_type(df_training_data_filtered, 'climate'), left_index=

# Filter out features measured after 30 August
df_training_data_filtered = feature_selection_on_date(df_training_data_filtered, '08
```

In [ ]:
```
# Time between 1. january until time
df_training_data_filtered['theDate'] = pd.to_datetime(df_training_data_filtered['the
df_training_data_filtered['time_since_jan'] = df_training_data_filtered['theDate'].a
    lambda x: (x - pd.Timestamp(f'{x.year}-01-01')).days
    )
```

Machine Learning Model Training

In [ ]:
```
target = 'drymatter_per'
```

In [ ]:
```
cv_col = 'harvestYear'
```

In [ ]:
```
dict_model_parameters = {
    'objective': 'reg:squarederror',
    'tree_method': 'approx',
    'eval_metric': 'mae',
    'enable_categorical': True,
    'early_stopping_rounds': 200,
    'n_estimators': 3000,
    'learning_rate': 0.1,
    'reg_lambda': 1000,
    'reg_alpha': 100,
    'max_depth': 6,
}
```

In [ ]:
```
features_to_include = df_training_data_filtered.columns[6:].tolist()
```

In [ ]:
```
# Exclude irrelevant columns
df_training_data_cv = df_training_data_filtered.copy()
df_training_data_cv = df_training_data_cv[features_to_include + [target, 'harvestYea
```

In [ ]:
```
# Run k-fold cross-validation
test_sets = []
feature_sets = []
for fold in df_training_data_cv[cv_col].unique():
    # Define train/test dataset
    df_test_data = df_training_data_cv.copy()
    df_test_data = df_test_data[df_test_data[cv_col] == fold]
    df_train_data = df_training_data_cv.copy()
    df_train_data = df_train_data[~(df_train_data[cv_col] == fold)]
    # Seperate features and target
    X_train = df_train_data.drop([target], axis=1)
    X_test = df_test_data.drop([target], axis=1)
    y_train = df_train_data[target]
    y_test = df_test_data[target]
    # Extract features to include
    X_train = X_train[features_to_include]
    X_test = X_test[features_to_include]
    column_list = list(X_train.columns)

    # Initiate model
    model = xgb.XGBRegressor(**dict_model_parameters)

    # Fit model
    model.fit(
        X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
        verbose=True
    )

    # Force negative predictions to zero
    y_pred = model.predict(X_test)
    test_set = X_test.copy()
    test_set['pred'] = y_pred
    test_set['target'] = y_test
    test_set[cv_col] = fold
    test_set['best_iteration'] = model.best_iteration

    # Append fold result to output list
    test_sets.append(test_set)
    feature_sets.append(model.get_booster().get_score(importance_type='gain'))
    del X_train
    del X_test
    del df_train_data
    del df_test_data

# Concatenate results into dataframe and aggregate to field-level
df_predictions = pd.concat(test_sets)
```

Evaluate results

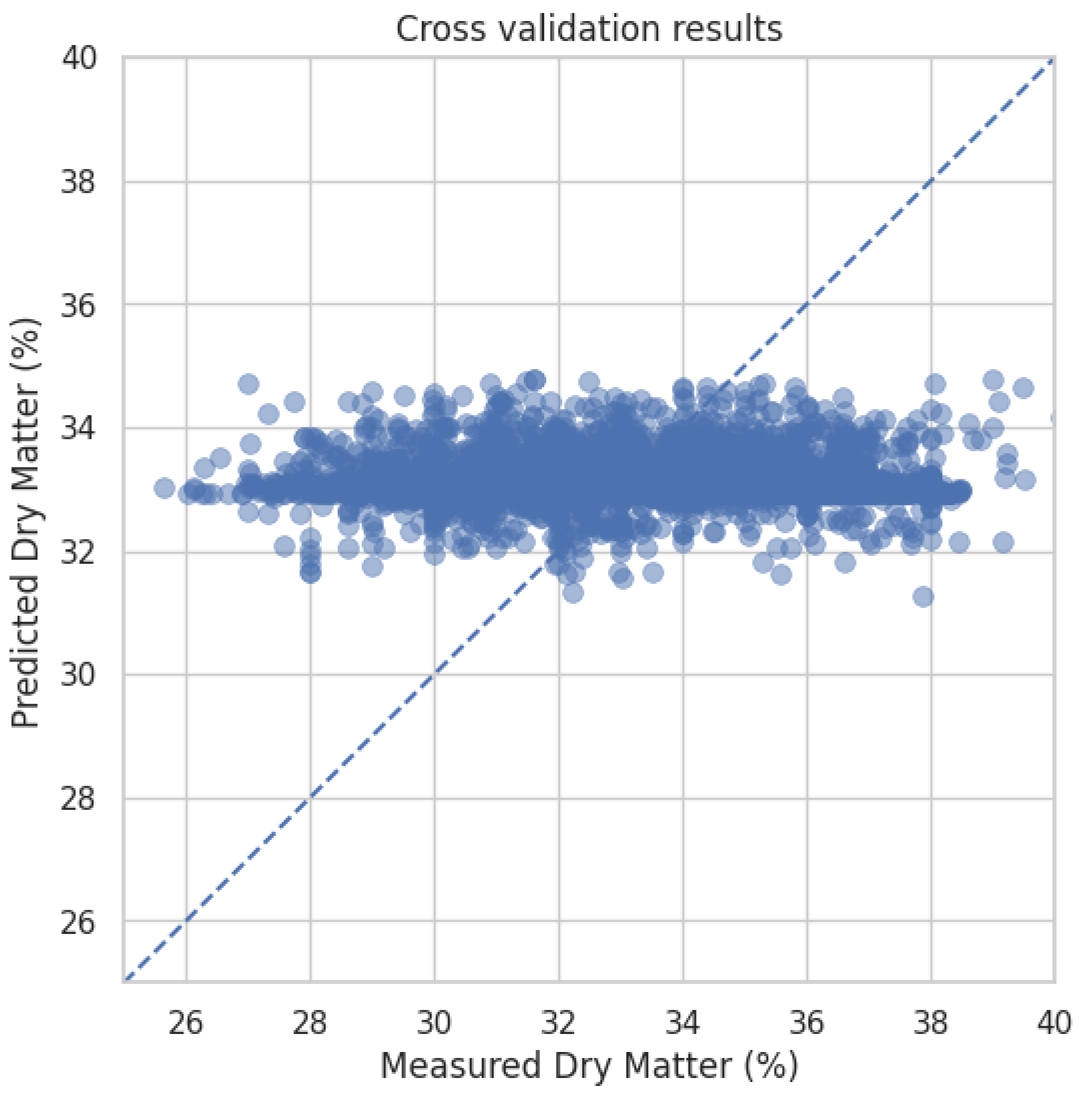
In [ ]:
```
# Dictionary to store evaluation output
eval_results = {}

#MAE
eval_results['MAE'] = sklearn.metrics.mean_absolute_error(
    df_predictions['target'], df_predictions['pred']
    )
#RMSE
eval_results['RMSE'] = np.sqrt(sklearn.metrics.mean_squared_error(
    df_predictions['target'], df_predictions['pred']
    ))
#R2
```

```
eval_results['R2'] = sklearn.metrics.r2_score(
    df_predictions['target'], df_predictions['pred']
    )

eval_results = {key : round(eval_results[key], 2) for key in eval_results}
```

```
In [ ]: pprint.pprint(eval_results)
```

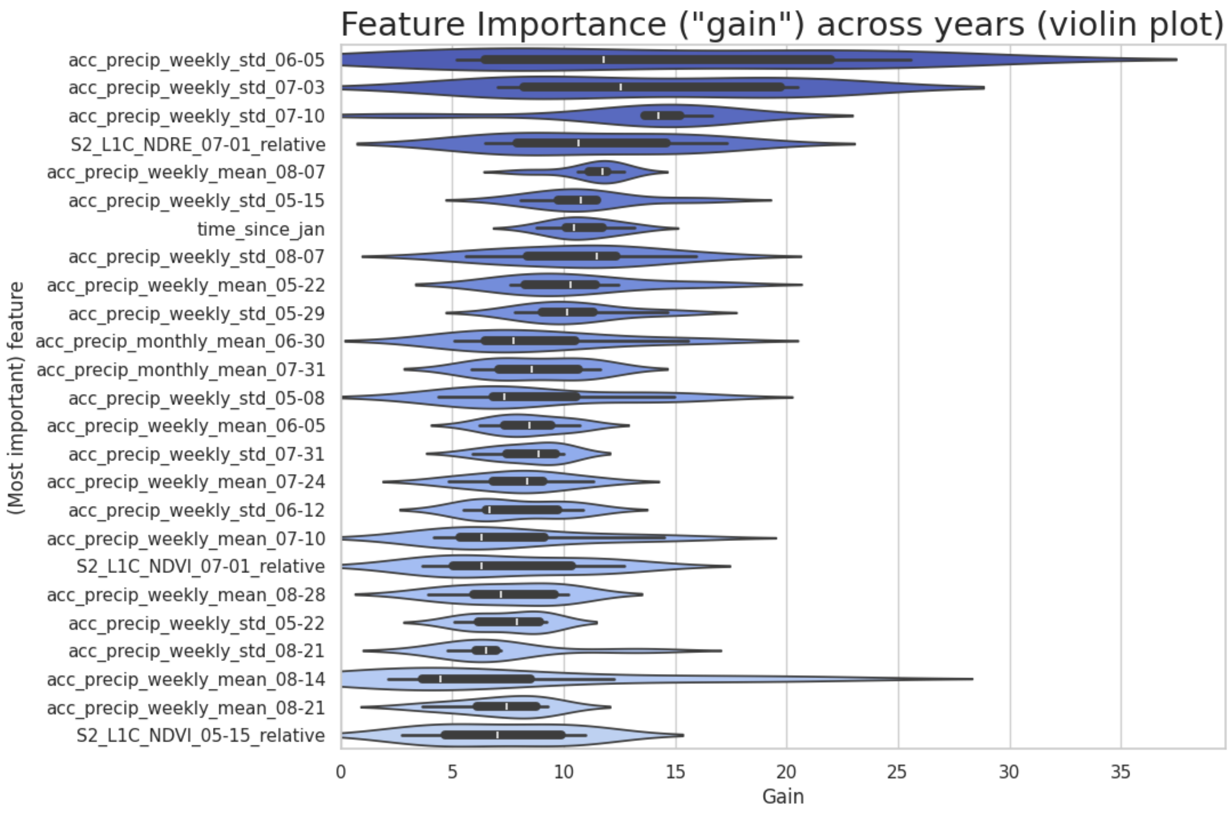
```
{'MAE': 2.09, 'R2': -0.01, 'RMSE': 2.56}
```

```
In [ ]: plt.figure(figsize=(6,6))
        sns.scatterplot(data=df_predictions, x='target', y='pred',
                        s=50, alpha=0.5, edgecolor=None)
        plt.title(f'Cross validation results')
        plt.xlabel('Measured Dry Matter (%)')
        plt.ylabel('Predicted Dry Matter (%)')
        plt.ylim(25, 40)
        plt.xlim(25, 40)
        plt.plot([0, 120], [0, 120], linestyle='dashed')
        plt.show()
```



```
In [ ]: df_folds = pd.DataFrame(feature_sets)

        # Calculate the mean importance for each feature across folds
        mean_importances = df_folds.mean().sort_values(ascending=False)
```

```
# Sort the columns of the DataFrame according to the mean importance
df_folds_sorted = df_folds[mean_importances.index]

# Create the flipped violin plot with sorted features
plt.figure(figsize=(10, 8))
sns.violinplot(data=df_folds_sorted, orient='h', order=mean_importances.index[:25],
plt.xlim(0, )
plt.xlabel('Gain')
plt.ylabel('(Most important) feature')
plt.title('Feature Importance ("gain") across years (violin plot)', fontsize=21)
plt.show()
```


Feature Importance ("gain") across years (violin plot)
acc_precip_weekly_std_06-05
acc_precip_weekly_std_07-03
acc_precip_weekly_std_07-10
S2_L1C_NDRE_07-01_relative
acc_precip_weekly_mean_08-07
acc_precip_weekly_std_05-15
time_since_jan
acc_precip_weekly_std_08-07
acc_precip_weekly_mean_05-22
acc_precip_weekly_std_05-29
acc_precip_monthly_mean_06-30
acc_precip_monthly_mean_07-31
acc_precip_weekly_std_05-08
acc_precip_weekly_mean_06-05
acc_precip_weekly_std_07-31
acc_precip_weekly_mean_07-24
acc_precip_weekly_std_06-12
acc_precip_weekly_mean_07-10
S2_L1C_NDVI_07-01_relative
acc_precip_weekly_mean_08-28
acc_precip_weekly_std_05-22
acc_precip_weekly_std_08-21
acc_precip_weekly_mean_08-14
acc_precip_weekly_mean_08-21
S2_L1C_NDVI_05-15_relative
(Most important) feature
Gain