Forecast_Farming_Crop_Yield

November 11, 2020

1 1 Setting up the Notebook

1.1 1.1 User Input

- If you want this script to run "quickly", please set the number of epochs (int_epochs) to 5.
  - This will take about 5 minutes for the whole script to run.
- If you have more time, please set a higher number for int_epochs:
  - 100 epochs take about 15 minutes.
  - 500 epochs take about 30 minutes.
- The input for int_epochs will affect the performance of the neural network models but not the linear and non-linear regression models.

1.2 1.2 Formatting Extension

This extension formats the code to general programming standards - %load_ext lab_black for jupyter lab - %load_ext nb_black for jupyter notebook

```
In [1]: # %load_ext lab_black
```

1.3 1.3 Importing Libraries

1.3.1 1.3.1 General Libraries

```
In [2]: import pandas as pd
        import numpy as np
        import sys
        import warnings
```

1.3.2 1.3.2 Scikit-learn Libraries

```
In [3]: from sklearn import preprocessing
        from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
        from sklearn.svm import LinearSVR
        from sklearn.feature_selection import RFECV
        from sklearn.exceptions import DataConversionWarning, ConvergenceWarning
        from sklearn.linear_model import LinearRegression
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.svm import SVR
```

1.3.3 1.3.3 Keras Libraries

1.4 1.4 Jupyter/Lab Notebook Display Settings

In [4]:
```
pd.set_option("display.max_rows", None)
pd.set_option("display.max_columns", None)
pd.set_option("display.width", None)
pd.set_option("display.max_colwidth", None)

warnings.filterwarnings(action="ignore", category=ConvergenceWarning)
```

1.5 1.5 Class for Formatting Display Outputs

In [5]:
```
class color:
    PURPLE = "\033[95m"
    CYAN = "\033[96m"
    DARKCYAN = "\033[36m"
    BLUE = "\033[94m"
    GREEN = "\033[92m"
    YELLOW = "\033[93m"
    RED = "\033[91m"
    BOLD = "\033[1m"
    UNDERLINE = "\033[4m"
    END = "\033[0m"
```

2 2 Data Pre-processing

2.1 2.1 Loading the Dataset

In [6]:
```
df = pd.read_csv("dataset.csv")
df.head()
```

Out[6]:

	id	water	uv	area	fertilizer_usage	yield	pesticides	region	categories
0	169	5.615	65.281	3.230	0	7.977	8.969	0	b,a,c
1	476	7.044	73.319	9.081	0	23.009	7.197	0	c,a,d
2	152	5.607	60.038	2.864	2	23.019	7.424	0	d,a
3	293	9.346	64.719	2.797	2	28.066	1.256	0	d
4	10	7.969	NaN	5.407	1	29.140	0.274	0	c,d

2.2 2.2 Dataset Structure

```
In [7]: print(color.BOLD + "columns: " + color.END, df.shape[1])
        print(color.BOLD + "of rows: " + color.END, df.shape[0], "\n")

        print(color.BOLD + "columns in dataset:" + color.END)
        list(df)
```

```
columns:  9
of rows:  1000

columns in dataset:
```

```
Out[7]: ['id',
         'water',
         'uv',
         'area',
         'fertilizer_usage',
         'yield',
         'pesticides',
         'region',
         'categories']
```

2.3 2.3 Null Values

The dataset contains several null values

```
In [8]: # the dataset contains whitespaces instead of empty cells <-- replacing whiespaces wit
        df.replace(" ", np.nan, inplace=True)

        # list with a count of null values for each column in the df
        lst_count_null_values = df.count() - len(df)

        # list with column names in the df <-- if a column has null values then the count will
        lst_column_names = list(df)

        # list to put the columns with null values <--[column name, # of null values]
        lst_col_null_values = np.empty(shape=[0, 2], dtype=object)

        # loop thru each column in the df and check whether it has null values
        for i in range(len(lst_count_null_values)):
            if lst_count_null_values[i] < 0:
                lst_col_null_values = np.vstack(
                    (
                        lst_col_null_values,
                        np.array((lst_column_names[i], abs(lst_count_null_values[i]))),
                    )
                )
```

```
        print(color.BOLD + "column(s) with null values" + color.END, "\n")
        for col in range(len(lst_col_null_values)):
            print(
                color.BOLD + lst_col_null_values[col][0] + color.END,
                "with",
                color.BOLD + lst_col_null_values[col][1] + color.END,
                "null values",
            )
```

```
column(s) with null values

water with 42 null values
uv with 51 null values
```

2.3.1 2.3.1 Null Values Details

```
In [9]: int_count_total_null_values = 0
        for i in range(len(lst_col_null_values)):
            int_count_total_null_values += int(lst_col_null_values[i][1])

        int_count_total_values = df.shape[1] * df.shape[0]

        print(
            color.BOLD + "total null values in dataset:" + color.END,
            int_count_total_null_values,
        )
        print(
            color.BOLD + "total size of dataset:" + color.END,
            df.shape[1],
            "columns",
            "*",
            df.shape[0],
            "rows",
            "=",
            int_count_total_values,
            "values",
        )
        print(
            color.BOLD + "percentage of null values in dataset:" + color.END,
            "%.4f" % ((int_count_total_null_values / int_count_total_values) * 100),
            "%",
        )
```

```
total null values in dataset: 93
total size of dataset: 9 columns * 1000 rows = 9000 values
percentage of null values in dataset: 1.0333 %
```

2.3.2 2.3.2 Null Value Replacement Options

Columns with missing values - Water - the average amount of water received by hectare - UV - the average amount of light received by hectare

Option 1: Replace all null values with 0 - This wouldn't make sense as it's impossible to have 0 water and 0 uv. - The minimum water for any farm was 0.072 and for water 45.254.

Option 2: Replace all null values with a constant value - This wouldn't make sense as water or uv isn't a constant.

Option 3: Forecast replacement values using machine learning - Without investigating all the variables further, it's clear that the following variables have no affect on either water or uv: area, fertilizer_usage, yield, pesticides usage, pesticides used. - This leaves only the variable region to have a direct effect on water and uv. - Since there's only 1 variable (region) to use, any regression or oder ML model wouldn't have much data to work with.

Option 4: Replace null values with an average/median **<– chosen option** - I could assign the average/median values of uv and water of the whole data set to the null values. - However, since the region dirrectly affects the water and uv, it would make more sense to use the average/median per region to fill the null values. - To be sure of this correlation, I'd have to additionally look at the correlation between region and water/uv - The average can be heavily influenced by extreme outliers. Therefore, I decided to use the median which better caputures the overall water and uv by region.

2.3.3 2.3.3 Null Value Replacement with Median per Region

```
In [10]: df["water"] = df["water"].fillna(df.groupby("region")["water"].transform("median"))
         df["uv"] = df["uv"].fillna(df.groupby("region")["uv"].transform("median"))
         # df.head()
```

2.4 2.4 Outlier Detection

- Looking at the table below, it's very clear that the maximum for water (5,340) must be an error in the data.
- The maximum numbers of the other variables seem to be correct but could still include outliers.

```
In [11]: df.describe()
```

```
Out[11]:                 id        water           uv         area  fertilizer_usage  \
         count  1000.000000  1000.000000  1000.000000  1000.000000        1000.00000
         mean    499.500000    11.981543    73.939665     8.098848           2.12300
         std     288.819436   168.677972     9.650628     2.692632           1.52256
         min       0.000000     0.072000    45.264000     0.263000           0.00000
         25%     249.750000     4.695500    66.931500     6.297000           1.00000
         50%     499.500000     6.452000    73.713500     7.987500           2.00000
         75%     749.250000     8.611000    80.220250     9.900250           3.00000
         max     999.000000  5340.000000   106.310000    18.311000           5.00000

                       yield   pesticides       region
         count  1000.000000  1000.000000  1000.000000
         mean     58.758571     3.452301     3.039000
```

```
std      24.563683     2.076921     1.883886
min       2.843000     0.014000     0.000000
25%      40.698000     1.804500     2.000000
50%      55.602500     3.275500     2.000000
75%      73.645500     4.916000     5.000000
max     148.845000     9.532000     6.000000
```

2.5 2.4.1 Correct Value in 'water'

I'll check whether there's more erroneous values in 'water' by sorting the DataFrame by water

```
In [12]: df.sort_values(["water"], ascending=(False)).head()
```

```
Out[12]:       id     water      uv    area  fertilizer_usage    yield  pesticides  \
         36   586  5340.000  91.224   8.429                 2   67.321       2.933
         182  594    15.214  66.904   8.438                 3   86.742       1.910
         412  756    14.217  65.374   7.549                 3   64.370       0.769
         739  434    13.832  85.961  11.295                 4  140.702       3.091
         260  886    13.529  86.763   3.507                 1   35.012       3.844

              region categories
         36        0        c,a
         182       1    a,b,c,d
         412       2          c
         739       4    a,d,c,b
         260       2        a,c
```

2.5.1 2.4.1.1 Finding Wrong Value

Looking at the above table, there seems to be only one erroneous value in 'water': 5,340

2.5.2 2.4.1.2 Replacing Wrong Value

I'm replacing the wrong value with the median 'water' by 'region

```
In [13]: df["water"] = df["water"].replace([df["water"].max()], np.nan)
         df["water"] = df["water"].fillna(df.groupby("region")["water"].transform("median"))
```

2.5.3 2.4.1.3 Checking Maximum again

Now the maximum value for 'water' seems to be within a correct range as shown in the table below

```
In [14]: df.describe()
```

```
Out[14]:                 id        water           uv         area  fertilizer_usage  \
         count  1000.000000  1000.000000  1000.000000  1000.000000        1000.00000
         mean    499.500000     6.647815    73.939665     8.098848           2.12300
         std     288.819436     2.759887     9.650628     2.692632           1.52256
```

```
min       0.000000     0.072000    45.264000     0.263000          0.00000
25%     249.750000     4.695500    66.931500     6.297000          1.00000
50%     499.500000     6.439500    73.713500     7.987500          2.00000
75%     749.250000     8.609250    80.220250     9.900250          3.00000
max     999.000000    15.214000   106.310000    18.311000          5.00000

              yield   pesticides       region
count   1000.000000  1000.000000  1000.000000
mean      58.758571     3.452301     3.039000
std       24.563683     2.076921     1.883886
min        2.843000     0.014000     0.000000
25%       40.698000     1.804500     2.000000
50%       55.602500     3.275500     2.000000
75%       73.645500     4.916000     5.000000
max      148.845000     9.532000     6.000000
```

2.5.4 2.4.2 Removal of Outliers

- I'm using the IQR, which is the middle 50% of the data, to identify outliers.
- If a value is 3x outside the IQR, I'm removing the entire row from the dataset.
- To compare the performance between the dataset with and without the outliers, I'm creating 2 separate DataFrames:
  - with outliers: df_w_outliers
  - without outliers: df_wo_outliers
- The 2 DataFrames are stored within a dictionary (dict_df) for easy looping through later on.

```
In [15]: df_w_outliers = df.copy()
         df_wo_outliers = df.copy()

         # setting up percentiles
         Q1 = df_wo_outliers.quantile(0.25)
         Q3 = df_wo_outliers.quantile(0.75)
         IQR = Q3 - Q1

         # removing rows with outliers from the df
         df_wo_outliers = df_wo_outliers[
             ~((df_wo_outliers < (Q1 - 1.5 * IQR)) | (df_wo_outliers > (Q3 + 1.5 * IQR))).any(
                 axis=1
             )
         ]

         # creating a dictionary with both dfs
         dict_df = {
             "df_w_outliers": ["with outliers", df_w_outliers],
             "df_wo_outliers": ["without outliers", df_wo_outliers],
         }
```

```
        # creating a list
        list_dfs = list(dict_df.keys())

        # deleting the duplicate DataFrames
        df_w_outliers = pd.DataFrame()
        df_wo_outliers = pd.DataFrame()
        df = pd.DataFrame()

        print(
            color.BOLD + "rows with outliers removed:" + color.END,
            dict_df["df_w_outliers"][1].shape[0] - dict_df["df_wo_outliers"][1].shape[0],
        )
```

```
rows with outliers removed: 32
```

2.6 2.4 Shuffling the DataFrame

The dataset seems to be ordered by region Because I'll later split the data into train and test data, I need to shuffle the data into a random order.

```
In [16]: for iter_df in list_dfs:
             dict_df[iter_df][1] = dict_df[iter_df][1].sample(frac=1).reset_index(drop=True)
         # dict_df[iter_df][1].head()
```

2.7 2.5 Creating Dummy Variables

2.7.1 2.5.1 Pesticides

2.5.1.1 Splitting 'categories' (used pesticides) into multiple boolean columns Since the pesticides used are comma separated in the column 'categories', I need to split these values up into multiple columns with boolean values. Splitting into multiple columns is required for data input of the model as well as the correlation analysis.

```
In [17]: # List of different kinds of pesticides
         lst_pesticide_categories = ["a", "b", "c", "d"]

         for iter_df in list_dfs:

             # Creating a new column stating whether a certain pesticide was used
             for str_pesticide_category in lst_pesticide_categories:
                 result = dict_df[iter_df][1].categories.str.contains(pat=str_pesticide_categor
                 dict_df[iter_df][1]["pesticide_contains_" + str_pesticide_category] = result

         dict_df[iter_df][1].head()
```

Out[17]:

	id	water	uv	area	fertilizer_usage	yield	pesticides	region	\
0	792	9.031	73.7135	7.764	2	63.153	2.561	2	
1	112	10.110	79.7420	5.594	3	64.496	4.515	2	

```
2  343   5.141  75.3150  5.833                1  50.170       2.458       0
3  955   5.695  70.5400  5.017                3  35.066       4.605       5
4  538   2.925  60.8330  8.633                1  39.785       0.221       2

  categories  pesticide_contains_a  pesticide_contains_b  \
0      c,d,b                 False                  True
1      c,b,d                 False                  True
2          c                 False                 False
3      d,b,c                 False                  True
4          d                 False                 False

   pesticide_contains_c  pesticide_contains_d
0                  True                  True
1                  True                  True
2                  True                 False
3                  True                  True
4                 False                  True
```

2.5.1.2 Convert Boolean Values to Integers As some models can only work with numerical data, I'm converting the boolean values to integers (0/1) I could have done this in one step but I wanted to do it in seperate steps to clearly show my work.

```
In [18]: for iter_df in list_dfs:

             for str_pesticide_category in lst_pesticide_categories:
                 dict_df[iter_df][1]["pesticide_contains_" + str_pesticide_category] = dict_df
                     iter_df
                 ][1]["pesticide_contains_" + str_pesticide_category].astype(int)
         dict_df[iter_df][1].head()
```

```
Out[18]:     id   water       uv   area  fertilizer_usage   yield  pesticides  region  \
         0  792   9.031  73.7135  7.764                 2  63.153       2.561       2
         1  112  10.110  79.7420  5.594                 3  64.496       4.515       2
         2  343   5.141  75.3150  5.833                 1  50.170       2.458       0
         3  955   5.695  70.5400  5.017                 3  35.066       4.605       5
         4  538   2.925  60.8330  8.633                 1  39.785       0.221       2

           categories  pesticide_contains_a  pesticide_contains_b  \
         0      c,d,b                     0                     1
         1      c,b,d                     0                     1
         2          c                     0                     0
         3      d,b,c                     0                     1
         4          d                     0                     0

            pesticide_contains_c  pesticide_contains_d
         0                     1                     1
         1                     1                     1
         2                     1                     0
```

```
3                    1                    1
4                    0                    1
```

2.5.1.3 Sorting the Pesticides within the 'categories' Column To get the unique combinaton of categories (pesticides) used together, I need to sort the categories alphabetically.

```
In [19]: for iter_df in list_dfs:
             dict_df[iter_df][1]["categories_sorted"] = np.nan
             for str_pesticide_category in lst_pesticide_categories:
                 dict_df[iter_df][1]["categories_sorted"] = np.where(
                     dict_df[iter_df][1]["pesticide_contains_" + str_pesticide_category] == 1,
                     dict_df[iter_df][1]["categories_sorted"].fillna("")
                     + str_pesticide_category,
                     dict_df[iter_df][1]["categories_sorted"],
                 )
         dict_df[iter_df][1].head()
```

```
Out[19]:     id   water       uv   area  fertilizer_usage   yield  pesticides  region  \
         0  792   9.031  73.7135  7.764                 2  63.153       2.561       2
         1  112  10.110  79.7420  5.594                 3  64.496       4.515       2
         2  343   5.141  75.3150  5.833                 1  50.170       2.458       0
         3  955   5.695  70.5400  5.017                 3  35.066       4.605       5
         4  538   2.925  60.8330  8.633                 1  39.785       0.221       2

           categories  pesticide_contains_a  pesticide_contains_b  \
         0      c,d,b                     0                     1
         1      c,b,d                     0                     1
         2          c                     0                     0
         3      d,b,c                     0                     1
         4          d                     0                     0

            pesticide_contains_c  pesticide_contains_d categories_sorted
         0                     1                     1               bcd
         1                     1                     1               bcd
         2                     1                     0                 c
         3                     1                     1               bcd
         4                     0                     1                 d
```

2.5.1.4 Splitting the Sorted Categories into multiple boolean Columns Splitting into multiple columns is required for data input of the model as well as the correlation analysis.

```
In [20]: for iter_df in list_dfs:
             for str_category_combination in dict_df[iter_df][1].categories_sorted.unique():
                 dict_df[iter_df][1]["pesticide_" + str_category_combination] = np.where(
                     dict_df[iter_df][1]["categories_sorted"] == str_category_combination,
                     True,
                     False,
                 )
         dict_df[iter_df][1].head()
```

Out[20]:

	id	water	uv	area	fertilizer_usage	yield	pesticides	region	\
0	792	9.031	73.7135	7.764	2	63.153	2.561	2	
1	112	10.110	79.7420	5.594	3	64.496	4.515	2	
2	343	5.141	75.3150	5.833	1	50.170	2.458	0	
3	955	5.695	70.5400	5.017	3	35.066	4.605	5	
4	538	2.925	60.8330	8.633	1	39.785	0.221	2	

	categories	pesticide_contains_a	pesticide_contains_b	\
0	c,d,b	0	1	
1	c,b,d	0	1	
2	c	0	0	
3	d,b,c	0	1	
4	d	0	0	

	pesticide_contains_c	pesticide_contains_d	categories_sorted	\
0	1	1	bcd	
1	1	1	bcd	
2	1	0	c	
3	1	1	bcd	
4	0	1	d	

	pesticide_bcd	pesticide_c	pesticide_d	pesticide_abcd	pesticide_bc	\
0	True	False	False	False	False	
1	True	False	False	False	False	
2	False	True	False	False	False	
3	True	False	False	False	False	
4	False	False	True	False	False	

	pesticide_abc	pesticide_b	pesticide_a	pesticide_ad	pesticide_acd	\
0	False	False	False	False	False	
1	False	False	False	False	False	
2	False	False	False	False	False	
3	False	False	False	False	False	
4	False	False	False	False	False	

	pesticide_bd	pesticide_ac	pesticide_abd	pesticide_ab	pesticide_cd
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False

2.5.1.5 Convert boolean Values to Integers As some models can only work with numerical data, I'm converting the boolean values to integers (0/1).

```
In [21]: for iter_df in list_dfs:
             for str_category_combination in dict_df[iter_df][1].categories_sorted.unique():
                 dict_df[iter_df][1]["pesticide_" + str_category_combination] = dict_df[iter_df
```

```
                1
        ]["pesticide_" + str_category_combination].astype(int)
dict_df[iter_df][1].head()
```

Out[21]:

	id	water	uv	area	fertilizer_usage	yield	pesticides	region
0	792	9.031	73.7135	7.764	2	63.153	2.561	2
1	112	10.110	79.7420	5.594	3	64.496	4.515	2
2	343	5.141	75.3150	5.833	1	50.170	2.458	0
3	955	5.695	70.5400	5.017	3	35.066	4.605	5
4	538	2.925	60.8330	8.633	1	39.785	0.221	2

	categories	pesticide_contains_a	pesticide_contains_b
0	c,d,b	0	1
1	c,b,d	0	1
2	c	0	0
3	d,b,c	0	1
4	d	0	0

	pesticide_contains_c	pesticide_contains_d	categories_sorted
0	1	1	bcd
1	1	1	bcd
2	1	0	c
3	1	1	bcd
4	0	1	d

	pesticide_bcd	pesticide_c	pesticide_d	pesticide_abcd	pesticide_bc
0	1	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	1	0	0	0	0
4	0	0	1	0	0

	pesticide_abc	pesticide_b	pesticide_a	pesticide_ad	pesticide_acd
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

	pesticide_bd	pesticide_ac	pesticide_abd	pesticide_ab	pesticide_cd
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

2.7.2 2.5.2 Regions

2.5.2.1 Creating a separate Column for each Region Depending on the model, splitting into multiple columns is required for data input as well as the correlation analysis. Unlike the previous method where I used a loop, I'm using a cleaner built-in function to get dummy columns.

```
In [22]: for iter_df in list_dfs:
             dict_df[iter_df][1]["region_temp"] = "region_" + dict_df[iter_df][1][
                 "region"
             ].astype(str)
             df_dummies = pd.get_dummies(dict_df[iter_df][1]["region_temp"])
             dict_df[iter_df][1] = pd.concat([dict_df[iter_df][1], df_dummies], axis=1)
             df_dummies = pd.DataFrame()
         dict_df[iter_df][1].head()
```

Out[22]:

	id	water	uv	area	fertilizer_usage	yield	pesticides	region
0	792	9.031	73.7135	7.764	2	63.153	2.561	2
1	112	10.110	79.7420	5.594	3	64.496	4.515	2
2	343	5.141	75.3150	5.833	1	50.170	2.458	0
3	955	5.695	70.5400	5.017	3	35.066	4.605	5
4	538	2.925	60.8330	8.633	1	39.785	0.221	2

	categories	pesticide_contains_a	pesticide_contains_b
0	c,d,b	0	1
1	c,b,d	0	1
2	c	0	0
3	d,b,c	0	1
4	d	0	0

	pesticide_contains_c	pesticide_contains_d	categories_sorted
0	1	1	bcd
1	1	1	bcd
2	1	0	c
3	1	1	bcd
4	0	1	d

	pesticide_bcd	pesticide_c	pesticide_d	pesticide_abcd	pesticide_bc
0	1	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	1	0	0	0	0
4	0	0	1	0	0

	pesticide_abc	pesticide_b	pesticide_a	pesticide_ad	pesticide_acd
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

```
   pesticide_bd  pesticide_ac  pesticide_abd  pesticide_ab  pesticide_cd  \
0             0             0              0             0             0
1             0             0              0             0             0
2             0             0              0             0             0
3             0             0              0             0             0
4             0             0              0             0             0

  region_temp  region_0  region_1  region_2  region_3  region_4  region_5  \
0    region_2         0         0         1         0         0         0
1    region_2         0         0         1         0         0         0
2    region_0         1         0         0         0         0         0
3    region_5         0         0         0         0         0         1
4    region_2         0         0         1         0         0         0

   region_6
0         0
1         0
2         0
3         0
4         0
```

2.8 2.6 Removing unused Columns

```
In [23]: for iter_df in list_dfs:
             dict_df[iter_df][1].drop(
                 columns=["id", "categories", "categories_sorted", "region_temp", "region"],
                 inplace=True,
             )
         dict_df[iter_df][1].head()
```

```
Out[23]:     water       uv   area  fertilizer_usage   yield  pesticides  \
         0   9.031  73.7135  7.764                 2  63.153       2.561
         1  10.110  79.7420  5.594                 3  64.496       4.515
         2   5.141  75.3150  5.833                 1  50.170       2.458
         3   5.695  70.5400  5.017                 3  35.066       4.605
         4   2.925  60.8330  8.633                 1  39.785       0.221

            pesticide_contains_a  pesticide_contains_b  pesticide_contains_c  \
         0                     0                     1                     1
         1                     0                     1                     1
         2                     0                     0                     1
         3                     0                     1                     1
         4                     0                     0                     0

            pesticide_contains_d  pesticide_bcd  pesticide_c  pesticide_d  \
         0                     1              1            0            0
         1                     1              1            0            0
         2                     0              0            1            0
```

```
3                    1             1            0            0
4                    1             0            0            1

   pesticide_abcd  pesticide_bc  pesticide_abc  pesticide_b  pesticide_a  \
0               0             0              0            0            0
1               0             0              0            0            0
2               0             0              0            0            0
3               0             0              0            0            0
4               0             0              0            0            0

   pesticide_ad  pesticide_acd  pesticide_bd  pesticide_ac  pesticide_abd  \
0             0              0             0             0              0
1             0              0             0             0              0
2             0              0             0             0              0
3             0              0             0             0              0
4             0              0             0             0              0

   pesticide_ab  pesticide_cd  region_0  region_1  region_2  region_3  \
0             0             0         0         0         1         0
1             0             0         0         0         1         0
2             0             0         1         0         0         0
3             0             0         0         0         0         0
4             0             0         0         0         1         0

   region_4  region_5  region_6
0         0         0         0
1         0         0         0
2         0         0         0
3         0         1         0
4         0         0         0
```

3 3 Exploratory Data Analysis

3.1 3.1 Calculating the Correlation between all Features and the Target Feature 'yield'

I'm using the Spearman method as it's better able to catch non-linear relationships. **Spearman Correlation Coefficient Range** - .00 - .19: very weak - .20 - .39: weak - .40 - .59: moderate - .60 - .79: strong - .80 - 1.0: very strong

```
In [24]: df_correlations = pd.DataFrame(columns=["feature", "correlation"])

         for str_column in dict_df["df_w_outliers"][1].columns:
             corr = dict_df["df_w_outliers"][1][str_column].corr(
                 dict_df["df_w_outliers"][1]["yield"], method="spearman"
             )  # pearson
             df_correlations.loc[len(df_correlations)] = [str_column, corr]

         df_correlations = df_correlations.reindex(
```

```
            df_correlations.correlation.abs().sort_values(ascending=False).index
        ).reset_index(drop=True)

        print(color.BOLD + "Spearman correlation with", "yield" + color.END, "\n")
        display(df_correlations)
```

Spearman correlation with yield

	feature	correlation
0	yield	1.000000
1	area	0.474607
2	fertilizer_usage	0.459740
3	region_4	0.237172
4	water	0.225665
5	region_6	-0.107846
6	region_5	-0.104509
7	pesticides	0.092768
8	region_2	-0.076822
9	pesticide_d	0.073817
10	pesticide_bcd	-0.067945
11	pesticide_contains_c	-0.060950
12	region_1	0.057477
13	pesticide_c	-0.053120
14	region_3	0.044820
15	uv	0.039300
16	pesticide_contains_b	-0.031860
17	pesticide_cd	0.028711
18	pesticide_abd	-0.025466
19	pesticide_a	0.025348
20	pesticide_b	0.023533
21	pesticide_ac	-0.017978
22	region_0	-0.015709
23	pesticide_ad	0.014146
24	pesticide_acd	0.010759
25	pesticide_bc	0.009510
26	pesticide_ab	0.007755
27	pesticide_contains_d	0.006533
28	pesticide_contains_a	0.005487
29	pesticide_bd	-0.005125
30	pesticide_abcd	-0.001571
31	pesticide_abc	0.000918

3.2 3.2 Quick Findings from EDA

Looking at the above table with the Spearman correlation I can tell that - there's a moderate positive correlation between the area and the yield - this means the larger the area the higher the

yield - there's a moderate positive correlation between the fertilizer_usage and the yield - this means the more fertilizer is used the higher the yield - region 4 has a weak correlation with the yield - this means that having the farm in region 4 will lead to a somewhat higher yield - all the other regions have a correlation that's close to 0 - this means that having the farm in any of these areas will not really affect the yield - region 4, 5, and 6 have a negative correlation with the yield - this means that having the farm in these regions will actually lead to a smaller yield - since the correlation is so close to 0 this won't make much of a difference though - the use of pesticides has a correlation with the yield that's very close to 0 (0.048380) - this means that using more or less pesticides will not lead to a higher/smaller yield - this finding is further backed as the pesticide combination 'd' has the the largest correlation with yield of merely 0.072035 - statistically 0.072035 is irrelevant and will barely affect the yield - some pesticide combinations actually have a negative correlation with the yield - this means that using these pesticide combinations will lead to a lower yield - since the correlation of all pesticide combinations is so close to 0 this won't make much of a difference though - however, I wouldn't recommend to any farms to stop using pesticides - not a single farm in the dataset chose to not use any pesticides (lowest pesticide usage was 0.014) - we therefore do not have enough data to confidentelly say that not using any pesticide wouldn't negatively affect the yield - furthermore, not all pesticides are meant to directly affect the yield/crop - some pesticides act as a protection just in case for eg. insects or weather (freezing, etc) - this protection can be looked at as an insurance and is therefore needed to potentially protect the crop even if this won't show in the data - it is interesting to note that the correlation between uv and the yield is very close to zero with 0.053070 - this means that whether there's more or less uv will almost have no effect on the yield - however, we can not say that there's no need for uv at all - no farm in the dataset had 0 uv - the farm with the least amount of uv had 45.264 uv by the hectare - we can therefore not recommend to any farm to move their crop inside or cover up their crop outside with uv protective material - we can say however that as long as a farm gets at least 45.264 uv by the hectare the yield will not be largely affected by not having enough uv - similar to uv, water also has a correlation to the yield that is very close to zero with 0.014631 - just by going with this very low correlation, I'd say that having more water will not lead to a higher yield - however, it is not clear from the data whether the dataset only shows the natural water received through precipitation - if the weather is very dry with a lack of precipitation, I'd assume that a farm will water their crop themselves - adding water manually will throw off the data and lead to wrong conclusions - therefore, based on the data given, I wouldn't make a statement whether adding more water helps with achieving a higher yield

4 4 Data Preparation for the Model Input

4.1 4.1 Normalizing the Data

For the data input to the model I need to normalize the data.

```
In [25]: # creating a dictionary with both dfs
         dict_df_normalized = {
             "df_w_outliers": ["with outliers", df_w_outliers],
             "df_wo_outliers": ["without outliers", df_wo_outliers],
         }

         for iter_df in list_dfs:
             x = dict_df[iter_df][1].values
```

```
        min_max_scaler = preprocessing.MinMaxScaler()
        x_scaled = min_max_scaler.fit_transform(x)
        dict_df_normalized[iter_df][1] = pd.DataFrame(
            x_scaled, columns=list(dict_df[iter_df][1])
        )
    dict_df_normalized[iter_df][1].head()
```

Out[25]:

	water	uv	area	fertilizer_usage	yield	pesticides
0	0.633369	0.502709	0.478981	0.4	0.494466	0.267598
1	0.709650	0.620212	0.324632	0.6	0.506502	0.472893
2	0.358360	0.533925	0.341632	0.2	0.378118	0.256777
3	0.397526	0.440854	0.283591	0.6	0.242761	0.482349
4	0.201697	0.251652	0.540792	0.2	0.285051	0.021748

	pesticide_contains_a	pesticide_contains_b	pesticide_contains_c
0	0.0	1.0	1.0
1	0.0	1.0	1.0
2	0.0	0.0	1.0
3	0.0	1.0	1.0
4	0.0	0.0	0.0

	pesticide_contains_d	pesticide_bcd	pesticide_c	pesticide_d
0	1.0	1.0	0.0	0.0
1	1.0	1.0	0.0	0.0
2	0.0	0.0	1.0	0.0
3	1.0	1.0	0.0	0.0
4	1.0	0.0	0.0	1.0

	pesticide_abcd	pesticide_bc	pesticide_abc	pesticide_b	pesticide_a
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0

	pesticide_ad	pesticide_acd	pesticide_bd	pesticide_ac	pesticide_abd
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0

	pesticide_ab	pesticide_cd	region_0	region_1	region_2	region_3
0	0.0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0

```
   region_4  region_5  region_6
0       0.0       0.0       0.0
1       0.0       0.0       0.0
2       0.0       0.0       0.0
3       0.0       1.0       0.0
4       0.0       0.0       0.0
```

4.2 4.2 Moving the Target Variable 'yield' to the End

The target variable 'yield' needs to be at the end of the dataframe for further data processing and model input.

In [26]:
```
str_target = "yield"

for iter_df in list_dfs:
    lst_columns = dict_df_normalized[iter_df][1].columns.tolist()

    # putting the target variable to the end of the list
    lst_columns.insert(
        dict_df_normalized[iter_df][1].shape[1] + 1,
        lst_columns.pop(lst_columns.index(str_target)),
    )

    dict_df_normalized[iter_df][1] = dict_df_normalized[iter_df][1].reindex(
        columns=lst_columns
    )
dict_df_normalized[iter_df][1].head()
```

Out[26]:
```
      water        uv      area  fertilizer_usage  pesticides  \
0  0.633369  0.502709  0.478981               0.4    0.267598
1  0.709650  0.620212  0.324632               0.6    0.472893
2  0.358360  0.533925  0.341632               0.2    0.256777
3  0.397526  0.440854  0.283591               0.6    0.482349
4  0.201697  0.251652  0.540792               0.2    0.021748

   pesticide_contains_a  pesticide_contains_b  pesticide_contains_c  \
0                   0.0                   1.0                   1.0
1                   0.0                   1.0                   1.0
2                   0.0                   0.0                   1.0
3                   0.0                   1.0                   1.0
4                   0.0                   0.0                   0.0

   pesticide_contains_d  pesticide_bcd  pesticide_c  pesticide_d  \
0                   1.0            1.0          0.0          0.0
1                   1.0            1.0          0.0          0.0
2                   0.0            0.0          1.0          0.0
3                   1.0            1.0          0.0          0.0
```

```
4                  1.0           0.0          0.0          1.0

   pesticide_abcd  pesticide_bc  pesticide_abc  pesticide_b  pesticide_a  \
0             0.0           0.0            0.0          0.0          0.0
1             0.0           0.0            0.0          0.0          0.0
2             0.0           0.0            0.0          0.0          0.0
3             0.0           0.0            0.0          0.0          0.0
4             0.0           0.0            0.0          0.0          0.0

   pesticide_ad  pesticide_acd  pesticide_bd  pesticide_ac  pesticide_abd  \
0           0.0            0.0           0.0           0.0            0.0
1           0.0            0.0           0.0           0.0            0.0
2           0.0            0.0           0.0           0.0            0.0
3           0.0            0.0           0.0           0.0            0.0
4           0.0            0.0           0.0           0.0            0.0

   pesticide_ab  pesticide_cd  region_0  region_1  region_2  region_3  \
0           0.0           0.0       0.0       0.0       1.0       0.0
1           0.0           0.0       0.0       0.0       1.0       0.0
2           0.0           0.0       1.0       0.0       0.0       0.0
3           0.0           0.0       0.0       0.0       0.0       0.0
4           0.0           0.0       0.0       0.0       1.0       0.0

   region_4  region_5  region_6     yield
0       0.0       0.0       0.0  0.494466
1       0.0       0.0       0.0  0.506502
2       0.0       0.0       0.0  0.378118
3       0.0       1.0       0.0  0.242761
4       0.0       0.0       0.0  0.285051
```

5 5 Forecast Model

6 5.1 Error Function

This is an error function returning the following errors: - Root Mean Squared Error (RMSE) - R-squared (r2)

```
In [27]: def get_errors(y_test, pred):
             rmse = np.sqrt(mean_squared_error(y_test, pred))
             r2 = r2_score(y_test, pred)
             #     mae = mean_absolute_error(y_test, pred)
             return rmse, r2
```

7 5.2 Feature Selection

To compare performance between different subsets of the dataset, I'll run the forecast with the following variables: - all variables in the DataFrame - only selected variabiles that have at least

somewhat of a correlation with the target feature 'yield' - "water", - "fertilizer_usage", - "uv", - "area", - "pesticides", - "region_1", - "region_2", - "region_3", - "region_4", - "region_5", - "region_6",

7.0.1 5.2.1 Setting up the 2 lists with the above columns selected

```
In [28]: lst_columns_feature_selection = [
             "water",
             "fertilizer_usage",
             "uv",
             "area",
             "pesticides",
             "region_1",
             "region_2",
             "region_3",
             "region_4",
             "region_5",
             "region_6",
             "yield",
         ]
         lst_all_columns = dict_df_normalized["df_w_outliers"][
             1
         ].columns  # all columns except the target variable 'yield'
         lst_columns = [
             [lst_all_columns, "all features"],
             [lst_columns_feature_selection, "selected features"],
         ]
```

7.1 5.3 Non-Linear and Linear Regression with Cross Validation

Because the dataset is relatively small with 1,000 records, I'll try to get a more constant performance with Cross Validadation. I'm running the same kernels/models as before with the Cross Validation being the only difference.

```
In [29]: # setting up all different kernels for the SVR model
         svr_linear = SVR(kernel="linear", C=100, gamma="auto")
         svr_rbf = SVR(kernel="rbf", C=100, gamma=0.1, epsilon=0.1)
         svr_poly = SVR(kernel="poly", C=100, gamma="auto", degree=3, epsilon=0.1, coef0=1)
         svr_sigmoid = SVR(kernel="sigmoid", C=100, gamma="auto", epsilon=0.1)  # , coef0=1)

         # list of all kernels that a loop will run through
         svrs = [
             [svr_rbf, "RBF"],
             [svr_linear, "Linear"],
             [svr_poly, "Polynomial"],
             [svr_sigmoid, "Sigmoid"],
         ]
```

```
# setting up the cross validation
int_splits = 5
cv = KFold(n_splits=int_splits, shuffle=False)
print("running cross validation in", int_splits, "splits", "\n")

# create a df to store error values in
df_scores = pd.DataFrame(
    columns=[
        "type",
        "kernel",
        "root_mean_squared_error",
        "r_quared",
        "dataset",
        "with_cross_validation",
        "selected_columns",
    ]
)

for iter_df in list_dfs:
    for svr in svrs:
        for column_selection in lst_columns:
            array = dict_df_normalized[iter_df][1][column_selection[0]].values

            # convert pandas df to an array for the model
            X = array[:, 0 : array.shape[1] - 1]
            Y = array[:, array.shape[1] - 1]

            scores_rmse = []
            scores_r2 = []
            scores_mae = []

            for train_index, test_index in cv.split(X):

                # setting up the data
                x_train, x_test = X[train_index], X[test_index]
                y_train, y_test = Y[train_index], Y[test_index]

                # data fitting and prediction
                pred = svr[0].fit(x_train, y_train).predict(x_test)

                # calculating the errors
                float_rmse, float_r2 = get_errors(y_test, pred)
                scores_rmse.append(float_rmse)
                scores_r2.append(float_r2)
            float_rmse = np.mean(scores_rmse)
            float_r2 = np.mean(scores_r2)

            # storing the errors
```

```
                df_scores.loc[len(df_scores)] = [
                    "regression",
                    svr[1],
                    float_rmse,
                    float_r2,
                    dict_df_normalized[iter_df][0],
                    "TRUE",
                    column_selection[1],
                ]
        print('running of model completed. Please move to the next cell.')
        # display(
        #     df_scores[
        #         (df_scores["type"] == "regression")
        #         & (df_scores["with_cross_validation"] == "TRUE")
        #     ]
        # )
```

```
running cross validation in 5 splits

running of model completed. Please move to the next cell.
```

7.2 5.4 Forecast Performance

7.2.1 5.4.1 Table with Errors

Here's a table of the 5 best models and their respective errors.

In [30]:
```
# SORT ERRORS BY R_SQUARED
df_scores = df_scores.sort_values(
    ["r_quared", "root_mean_squared_error"], ascending=(False, True)
)
df_scores.reset_index(drop=True, inplace=True)
df_scores.head(5)
```

Out[30]:

	type	kernel	root_mean_squared_error	r_quared	\
0	regression	RBF	0.074041	0.865658	
1	regression	Polynomial	0.074677	0.863072	
2	regression	RBF	0.061816	0.862148	
3	regression	Polynomial	0.062385	0.859235	
4	regression	Polynomial	0.087075	0.813502	

	dataset	with_cross_validation	selected_columns
0	without outliers	TRUE	selected features
1	without outliers	TRUE	selected features
2	with outliers	TRUE	selected features
3	with outliers	TRUE	selected features
4	without outliers	TRUE	all features

7.2.2 5.4.2 Results

- Using the dataset without the outliers and a subset of all features performed better than the dataset with the outliers.
  - This indicates that an improvement was achieved by further processing and filtering the data.
- The R Squared value of ~0.87 means that this particular model is able to explain the crop outcome by ~87% using the selected features.

If I was to proceed further with this project, I would

- go more detailed on the region.
  - This would give me more data on the climate, temperatures, precipitation, uv, etc.
  - Having the zip code would enable me to feed official weather forecasts to the model, and improve model forecasts for the next season/year
- go more detailed on the pesticides.
  - In the current dataset there's no information on the ratio of each pesticide if there were 2 or more pesticides used.
    - Knowing the exaction ration could make a difference.
- do a cost benefit analysis.
  - with the more information such as costs/prices for:
    - land
    - yield
    - workforce/labor
    - pesticides
    - etc
  - E.g., region 4 has the biggest positive effect on the yield.
    - However, if the cost of land in region 4 outweights the added return of the increase in yield, then a farm in another region may have a better profit by having lower costs.
  - The costs for labor is higher in certain regions and could reduce the profits compared to other regions.
  - With the cost of the pesticides I could calculate the optimal amount of pesticides to use in regards to profits and yields.
  - Overall I could forecast the best combination in order to get the biggest profit possible.
- calculate the optimal occupation of the land.
  - The data shows that the bigger the area, the higher the yield.
  - However, this could also mean that farms with less land tend to over plant, whereas farms with more land have the luxury of being able to spread out their crops.
  - With the crop occupation data I could calculate the idea amount of crops per hecare/square meter, etc.
- Since a selection of features had better forecasts than using all features, I could further try to find an even better subset of features.

- Since the non-linear regression models performed well, I could try other non-linear models such as
  - KNeighborsRegressor
  - DecisionTreeRegression()
- In this analyais I only did supervised learning and calculated the correlation between non-dependant and the target variable 'yield.
  - However, I could also perform unsupervised learning and try to find non-dependant variables that are correlated.