Forecast_Farming_Output

September 22, 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.

```
In [1]: int_epochs = 500
```

## 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 [2]: # %load_ext lab_black
```

## 1.3 1.3 Importing Libraries

### 1.3.1 1.3.1 General Libraries

```
In [3]: import pandas as pd
        import numpy as np
        import sys
        import warnings
```

### 1.3.2 1.3.2 Scikit-learn Libraries

```
In [4]: 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

```
In [5]: from keras.models import Sequential
        from keras.layers import Dense
```

```
Using TensorFlow backend.
```

## 1.4 1.4 Jupyter/Lab Notebook Display Settings

```
In [6]: 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 [7]: 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 [8]: df = pd.read_csv("dataset.csv")
        df.head()
```

```
Out[8]:     id  water      uv   area  fertilizer_usage   yield  pesticides  region  \
        0  169  5.615  65.281  3.230                 0   7.977       8.969       0
        1  476  7.044  73.319  9.081                 0  23.009       7.197       0
        2  152  5.607  60.038  2.864                 2  23.019       7.424       0
```

```
3  293  9.346  64.719  2.797                 2  28.066       1.256       0
4   10  7.969     NaN  5.407                 1  29.140       0.274       0

  categories
0      b,a,c
1      c,a,d
2        d,a
3          d
4        c,d
```

## 2.2 2.2 Dataset Structure

```
In [9]: 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[9]: ['id',
         'water',
         'uv',
         'area',
         'fertilizer_usage',
         'yield',
         'pesticides',
         'region',
         'categories']
```

## 2.3 2.3 Null Values

The dataset contains several null values

```
In [10]: # the dataset contains whitespaces instead of empty cells <-- replacing whiespaces wi
         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 wil
         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)
```

```python
    # 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

```python
In [11]: 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 [12]:
```
df["water"] = df["water"].fillna(df.groupby("region")["water"].transform("median"))
df["uv"] = df["uv"].fillna(df.groupby("region")["uv"].transform("median"))
df.head()
```

Out[12]:
```
    id  water      uv   area  fertilizer_usage   yield  pesticides  region  \
0  169  5.615  65.281  3.230                 0   7.977       8.969       0
1  476  7.044  73.319  9.081                 0  23.009       7.197       0
2  152  5.607  60.038  2.864                 2  23.019       7.424       0
3  293  9.346  64.719  2.797                 2  28.066       1.256       0
4   10  7.969  73.146  5.407                 1  29.140       0.274       0

  categories
0      b,a,c
1      c,a,d
```

```
2         d,a
3           d
4         c,d
```

## 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 [13]: df.describe()
```

```
Out[13]:                 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 [14]: df.sort_values(["water"], ascending=(False)).head()
```

```
Out[14]:       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 [15]: 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 [16]: df.describe()
```

```
Out[16]:                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 [17]: 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 [18]: 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()
```

```
Out[18]:     id   water      uv    area  fertilizer_usage    yield  pesticides  region  \
         0   45   6.272  82.678   9.107                 2   68.502       0.239       0
         1  678   4.945  70.920   6.971                 1   34.591       6.386       2
         2  646   8.183  81.743  11.287                 3  116.495       2.636       4
         3  744  11.511  66.432  12.033                 3   93.824       1.898       6
         4  158   4.580  81.187  12.782                 0   71.319       1.365       4

           categories
         0        d,b
         1        c,b
         2        c,d
         3          a
         4          b
```

## 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 [19]: # 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_catego
                 dict_df[iter_df][1]["pesticide_contains_" + str_pesticide_category] = result

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

```
Out[19]:     id   water      uv    area  fertilizer_usage    yield  pesticides  region  \
         0   45   6.272  82.678   9.107                 2   68.502       0.239       0
         1  678   4.945  70.920   6.971                 1   34.591       6.386       2
         2  646   8.183  81.743  11.287                 3  116.495       2.636       4
         3  744  11.511  66.432  12.033                 3   93.824       1.898       6
         4  158   4.580  81.187  12.782                 0   71.319       1.365       4

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

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

**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 [20]: 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[20]:     id   water      uv    area  fertilizer_usage    yield  pesticides  region  \
         0   45   6.272  82.678   9.107                 2   68.502       0.239       0
         1  678   4.945  70.920   6.971                 1   34.591       6.386       2
         2  646   8.183  81.743  11.287                 3  116.495       2.636       4
         3  744  11.511  66.432  12.033                 3   93.824       1.898       6
         4  158   4.580  81.187  12.782                 0   71.319       1.365       4

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

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

**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 [21]: 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[21]:     id   water      uv    area  fertilizer_usage    yield  pesticides  region  \
         0   45   6.272  82.678   9.107                 2   68.502       0.239       0
         1  678   4.945  70.920   6.971                 1   34.591       6.386       2
         2  646   8.183  81.743  11.287                 3  116.495       2.636       4
         3  744  11.511  66.432  12.033                 3   93.824       1.898       6
         4  158   4.580  81.187  12.782                 0   71.319       1.365       4

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

            pesticide_contains_c  pesticide_contains_d categories_sorted
         0                     0                     1                bd
         1                     1                     0                bc
         2                     1                     1                cd
         3                     0                     0                 a
         4                     0                     0                 b
```

**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 [22]: 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[22]:     id   water      uv    area  fertilizer_usage    yield  pesticides  region  \
         0   45   6.272  82.678   9.107                 2   68.502       0.239       0
         1  678   4.945  70.920   6.971                 1   34.591       6.386       2
         2  646   8.183  81.743  11.287                 3  116.495       2.636       4
         3  744  11.511  66.432  12.033                 3   93.824       1.898       6
         4  158   4.580  81.187  12.782                 0   71.319       1.365       4

           categories  pesticide_contains_a  pesticide_contains_b  \
         0        d,b                     0                     1
```

```
1        c,b                     0                      1
2        c,d                     0                      0
3          a                     1                      0
4          b                     0                      1

   pesticide_contains_c  pesticide_contains_d categories_sorted  pesticide_bd  \
0                     0                     1                bd          True
1                     1                     0                bc         False
2                     1                     1                cd         False
3                     0                     0                 a         False
4                     0                     0                 b         False

   pesticide_bc  pesticide_cd  pesticide_a  pesticide_b  pesticide_abc  \
0         False         False        False        False          False
1          True         False        False        False          False
2         False          True        False        False          False
3         False         False         True        False          False
4         False         False        False         True          False

   pesticide_bcd  pesticide_ab  pesticide_abcd  pesticide_acd  pesticide_ac  \
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_c  pesticide_d  pesticide_abd  pesticide_ad
0        False        False          False         False
1        False        False          False         False
2        False        False          False         False
3        False        False          False         False
4        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 [23]:
```python
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_d
            1
        ]["pesticide_" + str_category_combination].astype(int)
dict_df[iter_df][1].head()
```

Out[23]:
```
    id   water      uv    area  fertilizer_usage    yield  pesticides  region  \
0   45   6.272  82.678   9.107                 2   68.502       0.239       0
1  678   4.945  70.920   6.971                 1   34.591       6.386       2
2  646   8.183  81.743  11.287                 3  116.495       2.636       4
3  744  11.511  66.432  12.033                 3   93.824       1.898       6
```

```
4  158   4.580  81.187  12.782                  0   71.319       1.365       4

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

   pesticide_contains_c  pesticide_contains_d categories_sorted  pesticide_bd  \
0                     0                     1                bd             1
1                     1                     0                bc             0
2                     1                     1                cd             0
3                     0                     0                 a             0
4                     0                     0                 b             0

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

   pesticide_bcd  pesticide_ab  pesticide_abcd  pesticide_acd  pesticide_ac  \
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_c  pesticide_d  pesticide_abd  pesticide_ad
0            0            0              0             0
1            0            0              0             0
2            0            0              0             0
3            0            0              0             0
4            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 [24]: 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[24]:

```
    id   water      uv    area  fertilizer_usage    yield  pesticides  region  \
0   45   6.272  82.678   9.107                 2   68.502       0.239       0
1  678   4.945  70.920   6.971                 1   34.591       6.386       2
2  646   8.183  81.743  11.287                 3  116.495       2.636       4
3  744  11.511  66.432  12.033                 3   93.824       1.898       6
4  158   4.580  81.187  12.782                 0   71.319       1.365       4

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

   pesticide_contains_c  pesticide_contains_d categories_sorted  pesticide_bd  \
0                     0                     1                bd             1
1                     1                     0                bc             0
2                     1                     1                cd             0
3                     0                     0                 a             0
4                     0                     0                 b             0

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

   pesticide_bcd  pesticide_ab  pesticide_abcd  pesticide_acd  pesticide_ac  \
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_c  pesticide_d  pesticide_abd  pesticide_ad region_temp  \
0            0            0              0             0    region_0
1            0            0              0             0    region_2
2            0            0              0             0    region_4
3            0            0              0             0    region_6
4            0            0              0             0    region_4

   region_0  region_1  region_2  region_3  region_4  region_5  region_6
0         1         0         0         0         0         0         0
```

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

## 2.8 2.6 Removing unused Columns

In [25]:
```python
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[25]:
```
    water      uv    area  fertilizer_usage    yield  pesticides  \
0   6.272  82.678   9.107                 2   68.502       0.239
1   4.945  70.920   6.971                 1   34.591       6.386
2   8.183  81.743  11.287                 3  116.495       2.636
3  11.511  66.432  12.033                 3   93.824       1.898
4   4.580  81.187  12.782                 0   71.319       1.365

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

   pesticide_contains_d  pesticide_bd  pesticide_bc  pesticide_cd  \
0                     1             1             0             0
1                     0             0             1             0
2                     1             0             0             1
3                     0             0             0             0
4                     0             0             0             0

   pesticide_a  pesticide_b  pesticide_abc  pesticide_bcd  pesticide_ab  \
0            0            0              0              0             0
1            0            0              0              0             0
2            0            0              0              0             0
3            1            0              0              0             0
4            0            1              0              0             0

   pesticide_abcd  pesticide_acd  pesticide_ac  pesticide_c  pesticide_d  \
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_abd  pesticide_ad  region_0  region_1  region_2  region_3  \
0              0             0         1         0         0         0
1              0             0         0         0         1         0
2              0             0         0         0         0         0
3              0             0         0         0         0         0
4              0             0         0         0         0         0

   region_4  region_5  region_6
0         0         0         0
1         0         0         0
2         1         0         0
3         0         0         1
4         1         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 [26]:
```python
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 [27]: # 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[27]:       water        uv      area  fertilizer_usage     yield  pesticides  \
         0  0.438317  0.677439  0.574507               0.4  0.542402    0.023639
         1  0.344503  0.448260  0.422576               0.2  0.238504    0.669468
         2  0.573418  0.659215  0.729568               0.6  0.972497    0.275478
         3  0.808696  0.360784  0.782630               0.6  0.769328    0.197941
         4  0.318699  0.648377  0.835906               0.0  0.567647    0.141942

            pesticide_contains_a  pesticide_contains_b  pesticide_contains_c  \
```

```
0                0.0                 1.0                 0.0
1                0.0                 1.0                 1.0
2                0.0                 0.0                 1.0
3                1.0                 0.0                 0.0
4                0.0                 1.0                 0.0

   pesticide_contains_d  pesticide_bd  pesticide_bc  pesticide_cd  \
0                   1.0           1.0           0.0           0.0
1                   0.0           0.0           1.0           0.0
2                   1.0           0.0           0.0           1.0
3                   0.0           0.0           0.0           0.0
4                   0.0           0.0           0.0           0.0

   pesticide_a  pesticide_b  pesticide_abc  pesticide_bcd  pesticide_ab  \
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          1.0          0.0            0.0            0.0           0.0
4          0.0          1.0            0.0            0.0           0.0

   pesticide_abcd  pesticide_acd  pesticide_ac  pesticide_c  pesticide_d  \
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_abd  pesticide_ad  region_0  region_1  region_2  region_3  \
0            0.0           0.0       1.0       0.0       0.0       0.0
1            0.0           0.0       0.0       0.0       1.0       0.0
2            0.0           0.0       0.0       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       0.0       0.0

   region_4  region_5  region_6
0       0.0       0.0       0.0
1       0.0       0.0       0.0
2       1.0       0.0       0.0
3       0.0       0.0       1.0
4       1.0       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 [28]: str_target = "yield"
```

```python
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[28]:

```
      water        uv      area  fertilizer_usage  pesticides  \
0  0.438317  0.677439  0.574507               0.4    0.023639
1  0.344503  0.448260  0.422576               0.2    0.669468
2  0.573418  0.659215  0.729568               0.6    0.275478
3  0.808696  0.360784  0.782630               0.6    0.197941
4  0.318699  0.648377  0.835906               0.0    0.141942

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

   pesticide_contains_d  pesticide_bd  pesticide_bc  pesticide_cd  \
0                   1.0           1.0           0.0           0.0
1                   0.0           0.0           1.0           0.0
2                   1.0           0.0           0.0           1.0
3                   0.0           0.0           0.0           0.0
4                   0.0           0.0           0.0           0.0

   pesticide_a  pesticide_b  pesticide_abc  pesticide_bcd  pesticide_ab  \
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          1.0          0.0            0.0            0.0           0.0
4          0.0          1.0            0.0            0.0           0.0

   pesticide_abcd  pesticide_acd  pesticide_ac  pesticide_c  pesticide_d  \
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_abd  pesticide_ad  region_0  region_1  region_2  region_3  \
0            0.0           0.0       1.0       0.0       0.0       0.0
1            0.0           0.0       0.0       0.0       1.0       0.0
2            0.0           0.0       0.0       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       0.0       0.0

   region_4  region_5  region_6     yield
0       0.0       0.0       0.0  0.542402
1       0.0       0.0       0.0  0.238504
2       1.0       0.0       0.0  0.972497
3       0.0       0.0       1.0  0.769328
4       1.0       0.0       0.0  0.567647
```

# 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 [29]: 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 [30]: 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 Simple Linear Regression

- Since the target variable 'yield' is numerical and not categorical, the forecast model will need to be a regression and not a classification.
- To check whether the data is behaving linear or non-linear, I'll run a simple linear regression forecast.

### 5.3.1 Setting up the Model

```
In [31]: # 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:

             # setting up the model
             lin_reg_mod = LinearRegression()

             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]

                 # set up the training and testing data
```

```
            x_train, x_test, y_train, y_test = train_test_split(
                X, Y, test_size=0.2, random_state=9
            )

            # data fitting and prediction
            lin_reg_mod.fit(x_train, y_train)
            pred = lin_reg_mod.predict(x_test)

            # calculate the errors
            float_rmse, float_r2 = get_errors(y_test, pred)

            # store the errors
            df_scores.loc[len(df_scores)] = [
                "simple linear regression",
                "linear",
                float_rmse,
                float_r2,
                dict_df_normalized[iter_df][0],
                "FALSE",
                column_selection[1],
            ]

    display(
        df_scores[
            (df_scores["type"] == "simple linear regression")
            & (df_scores["with_cross_validation"] == "FALSE")
        ]
    )
```

```
                       type  kernel  root_mean_squared_error  r_quared  \
0  simple linear regression  linear                 0.081116  0.758543
1  simple linear regression  linear                 0.078537  0.773653
2  simple linear regression  linear                 0.108790  0.746350
3  simple linear regression  linear                 0.107384  0.752865

           dataset with_cross_validation   selected_columns
0     with outliers                FALSE       all features
1     with outliers                FALSE  selected features
2  without outliers                FALSE       all features
3  without outliers                FALSE  selected features
```

### 7.1.1 5.3.2 Simple Linear Regression Results

The R-squared is ok but I'd like to see whether non-linear models will result in a higher R-squared.

## 7.2 5.4 Non-Linear Regression

I'm using a Support Vector Regression model which is capable of running linear and non-linear kernels for easy comparison.

### 5.4.1 Setting up the Model

In [32]:
```
# 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"],
]

for iter_df in list_dfs:
    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]

        # setting up the data
        x_train, x_test, y_train, y_test = train_test_split(
            X, Y, test_size=0.2, random_state=9
        )

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

            # calculate the errors
            float_rmse, float_r2 = get_errors(y_test, pred)

            # store the errors
            df_scores.loc[len(df_scores)] = [
                "regression",
                svr[1],
                float_rmse,
                float_r2,
                dict_df_normalized[iter_df][0],
```

```
                    "FALSE",
                    column_selection[1],
                ]

    display(
        df_scores[
            (df_scores["type"] == "regression")
            & (df_scores["with_cross_validation"] == "FALSE")
        ]
    )
```

```
          type      kernel  root_mean_squared_error    r_quared  \
4   regression         RBF                 0.076720    0.784003
5   regression      Linear                 0.081493    0.756289
6   regression  Polynomial                 0.073699    0.800680
7   regression     Sigmoid                 1.490326  -80.506677
8   regression         RBF                 0.056798    0.881616
9   regression      Linear                 0.078927    0.771398
10  regression  Polynomial                 0.058043    0.876367
11  regression     Sigmoid                 3.452038 -436.302452
12  regression         RBF                 0.090365    0.824991
13  regression      Linear                 0.111015    0.735868
14  regression  Polynomial                 0.089113    0.829809
15  regression     Sigmoid                 1.441761  -43.549812
16  regression         RBF                 0.077192    0.872297
17  regression      Linear                 0.106098    0.758745
18  regression  Polynomial                 0.077608    0.870917
19  regression     Sigmoid                 3.567945 -271.832018

             dataset with_cross_validation   selected_columns
4      with outliers                 FALSE       all features
5      with outliers                 FALSE       all features
6      with outliers                 FALSE       all features
7      with outliers                 FALSE       all features
8      with outliers                 FALSE  selected features
9      with outliers                 FALSE  selected features
10     with outliers                 FALSE  selected features
11     with outliers                 FALSE  selected features
12  without outliers                 FALSE       all features
13  without outliers                 FALSE       all features
14  without outliers                 FALSE       all features
15  without outliers                 FALSE       all features
16  without outliers                 FALSE  selected features
17  without outliers                 FALSE  selected features
18  without outliers                 FALSE  selected features
19  without outliers                 FALSE  selected features
```

#### 7.2.1 5.4.2 Non-Linear Regression Results

- The non-linear kernels Polynomial and RBF achieved a lower R-squared than the linear Kernel.
- Only using a selection of variables achieved a lower R-squared than using all variables.

### 7.3 5.5 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.

#### 7.3.1 5.5.1 Running the Model

In [33]:
```
# setting up the cross validation
cv = KFold(n_splits=5, shuffle=False)
print("running cross validation in", cv.get_n_splits(X), "splits", "\n")

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],
                    ]

        display(
            df_scores[
                (df_scores["type"] == "regression")
                & (df_scores["with_cross_validation"] == "TRUE")
            ]
        )
```

```
running cross validation in 5 splits
```

```
          type      kernel  root_mean_squared_error    r_quared  \
20  regression         RBF                 0.076234    0.791343
21  regression         RBF                 0.062288    0.861642
22  regression      Linear                 0.088847    0.717449
23  regression      Linear                 0.088865    0.716434
24  regression  Polynomial                 0.078153    0.782056
25  regression  Polynomial                 0.062669    0.859874
26  regression     Sigmoid                 1.466412  -76.140513
27  regression     Sigmoid                 3.472238 -431.053880
28  regression         RBF                 0.091730    0.792738
29  regression         RBF                 0.074031    0.866079
30  regression      Linear                 0.110568    0.700097
31  regression      Linear                 0.111264    0.696555
32  regression  Polynomial                 0.090986    0.796766
33  regression  Polynomial                 0.074820    0.863138
34  regression     Sigmoid                 1.478559  -52.865534
35  regression     Sigmoid                 3.740371 -342.050891

           dataset with_cross_validation   selected_columns
20     with outliers                  TRUE       all features
21     with outliers                  TRUE  selected features
22     with outliers                  TRUE       all features
23     with outliers                  TRUE  selected features
24     with outliers                  TRUE       all features
25     with outliers                  TRUE  selected features
26     with outliers                  TRUE       all features
27     with outliers                  TRUE  selected features
```

```
28  without outliers                  TRUE       all features
29  without outliers                  TRUE  selected features
30  without outliers                  TRUE       all features
31  without outliers                  TRUE  selected features
32  without outliers                  TRUE       all features
33  without outliers                  TRUE  selected features
34  without outliers                  TRUE       all features
35  without outliers                  TRUE  selected features
```

### 7.3.2 5.5.2 Cross Validation Results

Cross validation had similar results to the performance without Cross Validation.

## 7.4 5.6 Simple Neural Network

I want to compare the performance of the non-linear and linear regression to a simple neural network.

### 7.4.1 5.6.1 Function to set up the Model

```
In [34]: def run_neural_network(str_activation, x_train, x_test, y_train):
             NN_model = Sequential()

             # input layer
             NN_model.add(
                 Dense(
                     128,
                     kernel_initializer="normal",
                     input_dim=x_train.shape[1],
                     activation=str_activation[0],  # "relu",
                 )
             )

             # hidden layers
             NN_model.add(Dense(256, kernel_initializer="normal", activation=str_activation[0]
             NN_model.add(Dense(256, kernel_initializer="normal", activation=str_activation[0]
             NN_model.add(
                 Dense(256, kernel_initializer="normal", activation=str_activation[0])
             )  # 256

             # output layer
             NN_model.add(Dense(1, kernel_initializer="normal", activation=str_activation[0]))
             # Compile the network :
             NN_model.compile(
                 loss="mean_squared_error",
                 optimizer="adam",
                 metrics=["mean_squared_error"],  # mean_absolute_error, mean_squared_error
```

```
    )
    # NN_model.summary()

    # fitting and prediction
    NN_model.fit(
        x_train,
        y_train,
        epochs=int_epochs,  # 100
        batch_size=512,
        validation_split=0.2,
        verbose=0,
    )

    pred = NN_model.predict(x_test)

    return pred
```

### 7.4.2 5.6.2 Running the Model

```
In [35]: print("depending on the number of epochs, this could take a few minutes :)", "\n")

         # I'm running different activations
         lst_activations = [
             ["relu", "ReLU - Rectified Linear Unit"],
             ["sigmoid", "Sigmoid"],
             ["softmax", "Softmax"],
             ["softplus", "Softplus"],
             ["softsign", "Softsign"],
             ["tanh", "TanH"],
             ["selu", "SELU - Scaled Exponential Linear Unit"],
             ["elu", "ELU - Exponential Linear Unit"],
             ["linear", "Linear"],
         ]

         for iter_df in list_dfs:
             for activation in lst_activations:
                 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]

                     # setting up the train and test data
                     x_train, x_test, y_train, y_test = train_test_split(
                         X, Y, test_size=0.2
                     )  # , random_state=9
                     #         )
```

```
                # fitting and prediction
                pred = run_neural_network(activation, x_train, x_test, y_train)

                # calculating the errors
                float_rmse, float_r2 = get_errors(y_test, pred)

                # storing the errors
                df_scores.loc[len(df_scores)] = [
                    "neural_network",
                    activation[1],
                    float_rmse,
                    float_r2,
                    dict_df_normalized[iter_df][0],
                    "FALSE",
                    column_selection[1],
                ]

    display(
        df_scores[
            (df_scores["type"] == "neural_network")
            & (df_scores["with_cross_validation"] == "FALSE")
        ]
    )
```

```
depending on the number of epochs, this could take a few minutes :)
```

```
              type                                  kernel  \
36  neural_network           ReLU - Rectified Linear Unit
37  neural_network           ReLU - Rectified Linear Unit
38  neural_network                                 Sigmoid
39  neural_network                                 Sigmoid
40  neural_network                                 Softmax
41  neural_network                                 Softmax
42  neural_network                                Softplus
43  neural_network                                Softplus
44  neural_network                                Softsign
45  neural_network                                Softsign
46  neural_network                                    TanH
47  neural_network                                    TanH
48  neural_network  SELU - Scaled Exponential Linear Unit
49  neural_network  SELU - Scaled Exponential Linear Unit
50  neural_network          ELU - Exponential Linear Unit
51  neural_network          ELU - Exponential Linear Unit
52  neural_network                                  Linear
53  neural_network                                  Linear
```

```
54  neural_network           ReLU - Rectified Linear Unit
55  neural_network           ReLU - Rectified Linear Unit
56  neural_network                                Sigmoid
57  neural_network                                Sigmoid
58  neural_network                                Softmax
59  neural_network                                Softmax
60  neural_network                               Softplus
61  neural_network                               Softplus
62  neural_network                               Softsign
63  neural_network                               Softsign
64  neural_network                                   TanH
65  neural_network                                   TanH
66  neural_network  SELU - Scaled Exponential Linear Unit
67  neural_network  SELU - Scaled Exponential Linear Unit
68  neural_network          ELU - Exponential Linear Unit
69  neural_network          ELU - Exponential Linear Unit
70  neural_network                                 Linear
71  neural_network                                 Linear

    root_mean_squared_error   r_quared          dataset  \
36                 0.076457   0.794860     with outliers
37                 0.054254   0.888782     with outliers
38                 0.080273   0.778777     with outliers
39                 0.090985   0.680555     with outliers
40                 0.629643 -13.372948     with outliers
41                 0.623934 -11.443360     with outliers
42                 0.079483   0.749949     with outliers
43                 0.096300   0.720523     with outliers
44                 0.071206   0.805948     with outliers
45                 0.056552   0.879200     with outliers
46                 0.073151   0.819492     with outliers
47                 0.094323   0.664072     with outliers
48                 0.058561   0.876479     with outliers
49                 0.053105   0.896711     with outliers
50                 0.066079   0.822431     with outliers
51                 0.060090   0.883066     with outliers
52                 0.087308   0.734234     with outliers
53                 0.090963   0.688694     with outliers
54                 0.096107   0.774213  without outliers
55                 0.073037   0.866787  without outliers
56                 0.121140   0.609411  without outliers
57                 0.106637   0.757489  without outliers
58                 0.584532  -6.447121  without outliers
59                 0.588103  -7.154905  without outliers
60                 0.116088   0.654376  without outliers
61                 0.111241   0.695368  without outliers
62                 0.113359   0.704017  without outliers
63                 0.081737   0.850747  without outliers
```

| | | | |
|---|---|---|---|
| 64 | 0.110933 | 0.676413 | without outliers |
| 65 | 0.117355 | 0.703304 | without outliers |
| 66 | 0.097799 | 0.770217 | without outliers |
| 67 | 0.078152 | 0.856779 | without outliers |
| 68 | 0.067728 | 0.892694 | without outliers |
| 69 | 0.072032 | 0.865422 | without outliers |
| 70 | 0.116624 | 0.671154 | without outliers |
| 71 | 0.112141 | 0.737414 | without outliers |

| | with_cross_validation | selected_columns |
|---|---|---|
| 36 | FALSE | all features |
| 37 | FALSE | selected features |
| 38 | FALSE | all features |
| 39 | FALSE | selected features |
| 40 | FALSE | all features |
| 41 | FALSE | selected features |
| 42 | FALSE | all features |
| 43 | FALSE | selected features |
| 44 | FALSE | all features |
| 45 | FALSE | selected features |
| 46 | FALSE | all features |
| 47 | FALSE | selected features |
| 48 | FALSE | all features |
| 49 | FALSE | selected features |
| 50 | FALSE | all features |
| 51 | FALSE | selected features |
| 52 | FALSE | all features |
| 53 | FALSE | selected features |
| 54 | FALSE | all features |
| 55 | FALSE | selected features |
| 56 | FALSE | all features |
| 57 | FALSE | selected features |
| 58 | FALSE | all features |
| 59 | FALSE | selected features |
| 60 | FALSE | all features |
| 61 | FALSE | selected features |
| 62 | FALSE | all features |
| 63 | FALSE | selected features |
| 64 | FALSE | all features |
| 65 | FALSE | selected features |
| 66 | FALSE | all features |
| 67 | FALSE | selected features |
| 68 | FALSE | all features |
| 69 | FALSE | selected features |
| 70 | FALSE | all features |
| 71 | FALSE | selected features |

### 7.4.3 5.6.3 Results

The R-squared is around the same than in the prior regression models. However, the number of epochs, layers, and other hyperparameters play a big role in the optimization and could be played with.

## 7.5 5.7 Simple Neural Network 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 activations/settings as before with the Cross Validation being the only difference.

### 7.5.1 5.6.1 Running the Model

In [36]:
```
# setting up the cross validation
cv = KFold(n_splits=5, shuffle=False)
print("running cross validation in", cv.get_n_splits(X), "splits", "\n")
print("depending on the number of epochs, this could take a few minutes :)", "\n")

int_iteration_counter = 0

for iter_df in list_dfs:
    for activation in lst_activations:
        for column_selection in lst_columns:
            int_iteration_counter += 1
            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 train and test data
                x_train, x_test = X[train_index], X[test_index]
                y_train, y_test = Y[train_index], Y[test_index]

                # fitting and prediction
                pred = run_neural_network(activation, x_train, x_test, y_train)

                # 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)] = [
                    "neural_network",
                    activation[1],
                    float_rmse,
                    float_r2,
                    dict_df_normalized[iter_df][0],
                    "TRUE",
                    column_selection[1],
                ]
                print(
                    "model",
                    int_iteration_counter,
                    "of",
                    len(lst_activations) * len(lst_columns) * len(list_dfs),
                    "completed",
                )

    display(
        df_scores[
            (df_scores["type"] == "neural_network")
            & (df_scores["with_cross_validation"] == "TRUE")
        ]
    )
```

```
running cross validation in 5 splits

depending on the number of epochs, this could take a few minutes :)

model 1 of 36 completed
model 2 of 36 completed
model 3 of 36 completed
model 4 of 36 completed
model 5 of 36 completed
model 6 of 36 completed
model 7 of 36 completed
model 8 of 36 completed
model 9 of 36 completed
model 10 of 36 completed
model 11 of 36 completed
model 12 of 36 completed
model 13 of 36 completed
model 14 of 36 completed
model 15 of 36 completed
model 16 of 36 completed
model 17 of 36 completed
```

```
model 18 of 36 completed
model 19 of 36 completed
model 20 of 36 completed
model 21 of 36 completed
model 22 of 36 completed
model 23 of 36 completed
model 24 of 36 completed
model 25 of 36 completed
model 26 of 36 completed
model 27 of 36 completed
model 28 of 36 completed
model 29 of 36 completed
model 30 of 36 completed
model 31 of 36 completed
model 32 of 36 completed
model 33 of 36 completed
model 34 of 36 completed
model 35 of 36 completed
model 36 of 36 completed


              type                                  kernel  \
72  neural_network           ReLU - Rectified Linear Unit
73  neural_network           ReLU - Rectified Linear Unit
74  neural_network                                 Sigmoid
75  neural_network                                 Sigmoid
76  neural_network                                 Softmax
77  neural_network                                 Softmax
78  neural_network                                Softplus
79  neural_network                                Softplus
80  neural_network                                Softsign
81  neural_network                                Softsign
82  neural_network                                    TanH
83  neural_network                                    TanH
84  neural_network  SELU - Scaled Exponential Linear Unit
85  neural_network  SELU - Scaled Exponential Linear Unit
86  neural_network          ELU - Exponential Linear Unit
87  neural_network          ELU - Exponential Linear Unit
88  neural_network                                  Linear
89  neural_network                                  Linear
90  neural_network           ReLU - Rectified Linear Unit
91  neural_network           ReLU - Rectified Linear Unit
92  neural_network                                 Sigmoid
93  neural_network                                 Sigmoid
94  neural_network                                 Softmax
95  neural_network                                 Softmax
96  neural_network                                Softplus
97  neural_network                                Softplus
```

| | | |
|---|---|---|
| 98 | neural_network | Softsign |
| 99 | neural_network | Softsign |
| 100 | neural_network | TanH |
| 101 | neural_network | TanH |
| 102 | neural_network | SELU - Scaled Exponential Linear Unit |
| 103 | neural_network | SELU - Scaled Exponential Linear Unit |
| 104 | neural_network | ELU - Exponential Linear Unit |
| 105 | neural_network | ELU - Exponential Linear Unit |
| 106 | neural_network | Linear |
| 107 | neural_network | Linear |

| | root_mean_squared_error | r_quared | dataset \ |
|---|---|---|---|
| 72 | 0.078870 | 0.777554 | with outliers |
| 73 | 0.131485 | -0.463065 | with outliers |
| 74 | 0.089975 | 0.709556 | with outliers |
| 75 | 0.089582 | 0.712511 | with outliers |
| 76 | 0.639468 | -13.635205 | with outliers |
| 77 | 0.639468 | -13.635205 | with outliers |
| 78 | 0.087169 | 0.728165 | with outliers |
| 79 | 0.086506 | 0.730882 | with outliers |
| 80 | 0.066929 | 0.840191 | with outliers |
| 81 | 0.059604 | 0.873532 | with outliers |
| 82 | 0.089029 | 0.718182 | with outliers |
| 83 | 0.093021 | 0.690578 | with outliers |
| 84 | 0.063883 | 0.853940 | with outliers |
| 85 | 0.057204 | 0.882896 | with outliers |
| 86 | 0.059316 | 0.873813 | with outliers |
| 87 | 0.058263 | 0.879099 | with outliers |
| 88 | 0.089087 | 0.715958 | with outliers |
| 89 | 0.090090 | 0.709690 | with outliers |
| 90 | 0.095075 | 0.778298 | without outliers |
| 91 | 0.161229 | -0.342178 | without outliers |
| 92 | 0.109410 | 0.706567 | without outliers |
| 93 | 0.111093 | 0.696690 | without outliers |
| 94 | 0.591214 | -7.570628 | without outliers |
| 95 | 0.591214 | -7.570628 | without outliers |
| 96 | 0.108794 | 0.710062 | without outliers |
| 97 | 0.112842 | 0.687195 | without outliers |
| 98 | 0.092015 | 0.791171 | without outliers |
| 99 | 0.077555 | 0.853009 | without outliers |
| 100 | 0.107126 | 0.715479 | without outliers |
| 101 | 0.107307 | 0.712117 | without outliers |
| 102 | 0.080048 | 0.843221 | without outliers |
| 103 | 0.068793 | 0.883478 | without outliers |
| 104 | 0.074233 | 0.864750 | without outliers |
| 105 | 0.071721 | 0.874464 | without outliers |
| 106 | 0.111664 | 0.694141 | without outliers |
| 107 | 0.112413 | 0.690223 | without outliers |

| | with_cross_validation | selected_columns |
|---|---|---|
| 72 | TRUE | all features |
| 73 | TRUE | selected features |
| 74 | TRUE | all features |
| 75 | TRUE | selected features |
| 76 | TRUE | all features |
| 77 | TRUE | selected features |
| 78 | TRUE | all features |
| 79 | TRUE | selected features |
| 80 | TRUE | all features |
| 81 | TRUE | selected features |
| 82 | TRUE | all features |
| 83 | TRUE | selected features |
| 84 | TRUE | all features |
| 85 | TRUE | selected features |
| 86 | TRUE | all features |
| 87 | TRUE | selected features |
| 88 | TRUE | all features |
| 89 | TRUE | selected features |
| 90 | TRUE | all features |
| 91 | TRUE | selected features |
| 92 | TRUE | all features |
| 93 | TRUE | selected features |
| 94 | TRUE | all features |
| 95 | TRUE | selected features |
| 96 | TRUE | all features |
| 97 | TRUE | selected features |
| 98 | TRUE | all features |
| 99 | TRUE | selected features |
| 100 | TRUE | all features |
| 101 | TRUE | selected features |
| 102 | TRUE | all features |
| 103 | TRUE | selected features |
| 104 | TRUE | all features |
| 105 | TRUE | selected features |
| 106 | TRUE | all features |
| 107 | TRUE | selected features |

### 7.5.2 5.7.2 Cross Validation Results

Cross validation was not able to improve R-squared. However, the number of times the cross validation runs can affect the performance.

## 7.6 5.8 Forecast Performance

### 7.6.1 5.8.1 Table with Errors

Here's a table of all models run and their respective errors.

In [37]:
```
# 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
```

Out[37]:

| | type | kernel \ |
|---|---|---|
| 0 | neural_network | SELU - Scaled Exponential Linear Unit |
| 1 | neural_network | ELU - Exponential Linear Unit |
| 2 | neural_network | ReLU - Rectified Linear Unit |
| 3 | neural_network | SELU - Scaled Exponential Linear Unit |
| 4 | neural_network | ELU - Exponential Linear Unit |
| 5 | neural_network | SELU - Scaled Exponential Linear Unit |
| 6 | regression | RBF |
| 7 | neural_network | Softsign |
| 8 | neural_network | ELU - Exponential Linear Unit |
| 9 | neural_network | SELU - Scaled Exponential Linear Unit |
| 10 | regression | Polynomial |
| 11 | neural_network | ELU - Exponential Linear Unit |
| 12 | neural_network | ELU - Exponential Linear Unit |
| 13 | neural_network | Softsign |
| 14 | regression | RBF |
| 15 | regression | Polynomial |
| 16 | neural_network | ReLU - Rectified Linear Unit |
| 17 | regression | RBF |
| 18 | neural_network | ELU - Exponential Linear Unit |
| 19 | neural_network | ELU - Exponential Linear Unit |
| 20 | regression | Polynomial |
| 21 | regression | RBF |
| 22 | regression | Polynomial |
| 23 | neural_network | SELU - Scaled Exponential Linear Unit |
| 24 | neural_network | SELU - Scaled Exponential Linear Unit |
| 25 | neural_network | Softsign |
| 26 | neural_network | Softsign |
| 27 | neural_network | SELU - Scaled Exponential Linear Unit |
| 28 | neural_network | Softsign |
| 29 | regression | Polynomial |
| 30 | regression | RBF |
| 31 | neural_network | ELU - Exponential Linear Unit |
| 32 | neural_network | TanH |
| 33 | neural_network | Softsign |
| 34 | regression | Polynomial |

| | | |
|---|---|---|
| 35 | regression | Polynomial |
| 36 | neural_network | ReLU - Rectified Linear Unit |
| 37 | regression | RBF |
| 38 | regression | RBF |
| 39 | neural_network | Softsign |
| 40 | regression | RBF |
| 41 | regression | Polynomial |
| 42 | neural_network | Sigmoid |
| 43 | neural_network | ReLU - Rectified Linear Unit |
| 44 | neural_network | ReLU - Rectified Linear Unit |
| 45 | neural_network | ReLU - Rectified Linear Unit |
| 46 | simple linear regression | linear |
| 47 | regression | Linear |
| 48 | neural_network | SELU - Scaled Exponential Linear Unit |
| 49 | regression | Linear |
| 50 | simple linear regression | linear |
| 51 | neural_network | Sigmoid |
| 52 | regression | Linear |
| 53 | simple linear regression | linear |
| 54 | neural_network | Softplus |
| 55 | simple linear regression | linear |
| 56 | neural_network | Linear |
| 57 | regression | Linear |
| 58 | neural_network | Linear |
| 59 | neural_network | Softplus |
| 60 | neural_network | Softplus |
| 61 | neural_network | Softplus |
| 62 | neural_network | TanH |
| 63 | regression | Linear |
| 64 | regression | Linear |
| 65 | neural_network | Linear |
| 66 | neural_network | TanH |
| 67 | neural_network | Sigmoid |
| 68 | neural_network | TanH |
| 69 | neural_network | Softplus |
| 70 | neural_network | Linear |
| 71 | neural_network | Sigmoid |
| 72 | neural_network | Sigmoid |
| 73 | neural_network | Softsign |
| 74 | neural_network | TanH |
| 75 | regression | Linear |
| 76 | neural_network | Sigmoid |
| 77 | regression | Linear |
| 78 | neural_network | Softplus |
| 79 | neural_network | Linear |
| 80 | neural_network | TanH |
| 81 | neural_network | Linear |
| 82 | neural_network | Linear |

| | | |
|---|---|---|
| 83 | neural_network | Softplus |
| 84 | neural_network | Sigmoid |
| 85 | neural_network | TanH |
| 86 | neural_network | Linear |
| 87 | neural_network | TanH |
| 88 | neural_network | Softplus |
| 89 | neural_network | Sigmoid |
| 90 | neural_network | ReLU - Rectified Linear Unit |
| 91 | neural_network | ReLU - Rectified Linear Unit |
| 92 | neural_network | Softmax |
| 93 | neural_network | Softmax |
| 94 | neural_network | Softmax |
| 95 | neural_network | Softmax |
| 96 | neural_network | Softmax |
| 97 | neural_network | Softmax |
| 98 | neural_network | Softmax |
| 99 | neural_network | Softmax |
| 100 | regression | Sigmoid |
| 101 | regression | Sigmoid |
| 102 | regression | Sigmoid |
| 103 | regression | Sigmoid |
| 104 | regression | Sigmoid |
| 105 | regression | Sigmoid |
| 106 | regression | Sigmoid |
| 107 | regression | Sigmoid |

| | root_mean_squared_error | r_quared | dataset \ |
|---|---|---|---|
| 0 | 0.053105 | 0.896711 | with outliers |
| 1 | 0.067728 | 0.892694 | without outliers |
| 2 | 0.054254 | 0.888782 | with outliers |
| 3 | 0.068793 | 0.883478 | without outliers |
| 4 | 0.060090 | 0.883066 | with outliers |
| 5 | 0.057204 | 0.882896 | with outliers |
| 6 | 0.056798 | 0.881616 | with outliers |
| 7 | 0.056552 | 0.879200 | with outliers |
| 8 | 0.058263 | 0.879099 | with outliers |
| 9 | 0.058561 | 0.876479 | with outliers |
| 10 | 0.058043 | 0.876367 | with outliers |
| 11 | 0.071721 | 0.874464 | without outliers |
| 12 | 0.059316 | 0.873813 | with outliers |
| 13 | 0.059604 | 0.873532 | with outliers |
| 14 | 0.077192 | 0.872297 | without outliers |
| 15 | 0.077608 | 0.870917 | without outliers |
| 16 | 0.073037 | 0.866787 | without outliers |
| 17 | 0.074031 | 0.866079 | without outliers |
| 18 | 0.072032 | 0.865422 | without outliers |
| 19 | 0.074233 | 0.864750 | without outliers |
| 20 | 0.074820 | 0.863138 | without outliers |

| | | | |
|---|---|---|---|
| 21 | 0.062288 | 0.861642 | with outliers |
| 22 | 0.062669 | 0.859874 | with outliers |
| 23 | 0.078152 | 0.856779 | without outliers |
| 24 | 0.063883 | 0.853940 | with outliers |
| 25 | 0.077555 | 0.853009 | without outliers |
| 26 | 0.081737 | 0.850747 | without outliers |
| 27 | 0.080048 | 0.843221 | without outliers |
| 28 | 0.066929 | 0.840191 | with outliers |
| 29 | 0.089113 | 0.829809 | without outliers |
| 30 | 0.090365 | 0.824991 | without outliers |
| 31 | 0.066079 | 0.822431 | with outliers |
| 32 | 0.073151 | 0.819492 | with outliers |
| 33 | 0.071206 | 0.805948 | with outliers |
| 34 | 0.073699 | 0.800680 | with outliers |
| 35 | 0.090986 | 0.796766 | without outliers |
| 36 | 0.076457 | 0.794860 | with outliers |
| 37 | 0.091730 | 0.792738 | without outliers |
| 38 | 0.076234 | 0.791343 | with outliers |
| 39 | 0.092015 | 0.791171 | without outliers |
| 40 | 0.076720 | 0.784003 | with outliers |
| 41 | 0.078153 | 0.782056 | with outliers |
| 42 | 0.080273 | 0.778777 | with outliers |
| 43 | 0.095075 | 0.778298 | without outliers |
| 44 | 0.078870 | 0.777554 | with outliers |
| 45 | 0.096107 | 0.774213 | without outliers |
| 46 | 0.078537 | 0.773653 | with outliers |
| 47 | 0.078927 | 0.771398 | with outliers |
| 48 | 0.097799 | 0.770217 | without outliers |
| 49 | 0.106098 | 0.758745 | without outliers |
| 50 | 0.081116 | 0.758543 | with outliers |
| 51 | 0.106637 | 0.757489 | without outliers |
| 52 | 0.081493 | 0.756289 | with outliers |
| 53 | 0.107384 | 0.752865 | without outliers |
| 54 | 0.079483 | 0.749949 | with outliers |
| 55 | 0.108790 | 0.746350 | without outliers |
| 56 | 0.112141 | 0.737414 | without outliers |
| 57 | 0.111015 | 0.735868 | without outliers |
| 58 | 0.087308 | 0.734234 | with outliers |
| 59 | 0.086506 | 0.730882 | with outliers |
| 60 | 0.087169 | 0.728165 | with outliers |
| 61 | 0.096300 | 0.720523 | with outliers |
| 62 | 0.089029 | 0.718182 | with outliers |
| 63 | 0.088847 | 0.717449 | with outliers |
| 64 | 0.088865 | 0.716434 | with outliers |
| 65 | 0.089087 | 0.715958 | with outliers |
| 66 | 0.107126 | 0.715479 | without outliers |
| 67 | 0.089582 | 0.712511 | with outliers |
| 68 | 0.107307 | 0.712117 | without outliers |

| | | | |
|---|---|---|---|
| 69 | 0.108794 | 0.710062 | without outliers |
| 70 | 0.090090 | 0.709690 | with outliers |
| 71 | 0.089975 | 0.709556 | with outliers |
| 72 | 0.109410 | 0.706567 | without outliers |
| 73 | 0.113359 | 0.704017 | without outliers |
| 74 | 0.117355 | 0.703304 | without outliers |
| 75 | 0.110568 | 0.700097 | without outliers |
| 76 | 0.111093 | 0.696690 | without outliers |
| 77 | 0.111264 | 0.696555 | without outliers |
| 78 | 0.111241 | 0.695368 | without outliers |
| 79 | 0.111664 | 0.694141 | without outliers |
| 80 | 0.093021 | 0.690578 | with outliers |
| 81 | 0.112413 | 0.690223 | without outliers |
| 82 | 0.090963 | 0.688694 | with outliers |
| 83 | 0.112842 | 0.687195 | without outliers |
| 84 | 0.090985 | 0.680555 | with outliers |
| 85 | 0.110933 | 0.676413 | without outliers |
| 86 | 0.116624 | 0.671154 | without outliers |
| 87 | 0.094323 | 0.664072 | with outliers |
| 88 | 0.116088 | 0.654376 | without outliers |
| 89 | 0.121140 | 0.609411 | without outliers |
| 90 | 0.161229 | -0.342178 | without outliers |
| 91 | 0.131485 | -0.463065 | with outliers |
| 92 | 0.584532 | -6.447121 | without outliers |
| 93 | 0.588103 | -7.154905 | without outliers |
| 94 | 0.591214 | -7.570628 | without outliers |
| 95 | 0.591214 | -7.570628 | without outliers |
| 96 | 0.623934 | -11.443360 | with outliers |
| 97 | 0.629643 | -13.372948 | with outliers |
| 98 | 0.639468 | -13.635205 | with outliers |
| 99 | 0.639468 | -13.635205 | with outliers |
| 100 | 1.441761 | -43.549812 | without outliers |
| 101 | 1.478559 | -52.865534 | without outliers |
| 102 | 1.466412 | -76.140513 | with outliers |
| 103 | 1.490326 | -80.506677 | with outliers |
| 104 | 3.567945 | -271.832018 | without outliers |
| 105 | 3.740371 | -342.050891 | without outliers |
| 106 | 3.472238 | -431.053880 | with outliers |
| 107 | 3.452038 | -436.302452 | with outliers |

| | with_cross_validation | selected_columns |
|---|---|---|
| 0 | FALSE | selected features |
| 1 | FALSE | all features |
| 2 | FALSE | selected features |
| 3 | TRUE | selected features |
| 4 | FALSE | selected features |
| 5 | TRUE | selected features |
| 6 | FALSE | selected features |

| | | |
|---|---|---|
| 7 | FALSE | selected features |
| 8 | TRUE | selected features |
| 9 | FALSE | all features |
| 10 | FALSE | selected features |
| 11 | TRUE | selected features |
| 12 | TRUE | all features |
| 13 | TRUE | selected features |
| 14 | FALSE | selected features |
| 15 | FALSE | selected features |
| 16 | FALSE | selected features |
| 17 | TRUE | selected features |
| 18 | FALSE | selected features |
| 19 | TRUE | all features |
| 20 | TRUE | selected features |
| 21 | TRUE | selected features |
| 22 | TRUE | selected features |
| 23 | FALSE | selected features |
| 24 | TRUE | all features |
| 25 | TRUE | selected features |
| 26 | FALSE | selected features |
| 27 | TRUE | all features |
| 28 | TRUE | all features |
| 29 | FALSE | all features |
| 30 | FALSE | all features |
| 31 | FALSE | all features |
| 32 | FALSE | all features |
| 33 | FALSE | all features |
| 34 | FALSE | all features |
| 35 | TRUE | all features |
| 36 | FALSE | all features |
| 37 | TRUE | all features |
| 38 | TRUE | all features |
| 39 | TRUE | all features |
| 40 | FALSE | all features |
| 41 | TRUE | all features |
| 42 | FALSE | all features |
| 43 | TRUE | all features |
| 44 | TRUE | all features |
| 45 | FALSE | all features |
| 46 | FALSE | selected features |
| 47 | FALSE | selected features |
| 48 | FALSE | all features |
| 49 | FALSE | selected features |
| 50 | FALSE | all features |
| 51 | FALSE | selected features |
| 52 | FALSE | all features |
| 53 | FALSE | selected features |
| 54 | FALSE | all features |

| | | |
|---|---|---|
| 55 | FALSE | all features |
| 56 | FALSE | selected features |
| 57 | FALSE | all features |
| 58 | FALSE | all features |
| 59 | TRUE | selected features |
| 60 | TRUE | all features |
| 61 | FALSE | selected features |
| 62 | TRUE | all features |
| 63 | TRUE | all features |
| 64 | TRUE | selected features |
| 65 | TRUE | all features |
| 66 | TRUE | all features |
| 67 | TRUE | selected features |
| 68 | TRUE | selected features |
| 69 | TRUE | all features |
| 70 | TRUE | selected features |
| 71 | TRUE | all features |
| 72 | TRUE | all features |
| 73 | FALSE | all features |
| 74 | FALSE | selected features |
| 75 | TRUE | all features |
| 76 | TRUE | selected features |
| 77 | TRUE | selected features |
| 78 | FALSE | selected features |
| 79 | TRUE | all features |
| 80 | TRUE | selected features |
| 81 | TRUE | selected features |
| 82 | FALSE | selected features |
| 83 | TRUE | selected features |
| 84 | FALSE | selected features |
| 85 | FALSE | all features |
| 86 | FALSE | all features |
| 87 | FALSE | selected features |
| 88 | FALSE | all features |
| 89 | FALSE | all features |
| 90 | TRUE | selected features |
| 91 | TRUE | selected features |
| 92 | FALSE | all features |
| 93 | FALSE | selected features |
| 94 | TRUE | all features |
| 95 | TRUE | selected features |
| 96 | FALSE | selected features |
| 97 | FALSE | all features |
| 98 | TRUE | all features |
| 99 | TRUE | selected features |
| 100 | FALSE | all features |
| 101 | TRUE | all features |
| 102 | TRUE | all features |

```
103                    FALSE       all features
104                    FALSE  selected features
105                     TRUE  selected features
106                     TRUE  selected features
107                    FALSE  selected features
```

#### 7.6.2 5.8.2 Results

**Depending on the number of epochs chosen and other factors, my results (with 500 epochs) may differ from the results when running with a different number of epochs.** - Neural networks outperform the regression models. - The SELU neural network performed best. (highest r-squared and lowest RMSE) - SELU performed well with and without cross validation. This means that this model performs well overall and is expected to work well with any new data that's within a regular, historic range. - Therefore, I'd pick this model for it's overall performance as it did well with and without cross validation. - Expectedly, cross validation did not perform as well since it's a mean measure and not a best measure like non cross validation. - Using the dataset with the outliers performed better than the dataset without the outliers. - However, if have seen this change quite a bit when running the models multiple times. - So, whether I removed outliers or not didn't seem to change the outcome much. - The limited selection of features performed better than using all features. - The linear regression models did not perform as well as the non-linear regression models.

## 8 6 Next Steps

**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.

- tune the hyperparameters for all models.
- 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()

- Since the simple neural network performed well, I could try other neural networks.
- 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.