ocean DIMITRA

Data Challenge

Phase 2

Prediction of soybean crop yields in Madhya Pradesh using machine learning

2022

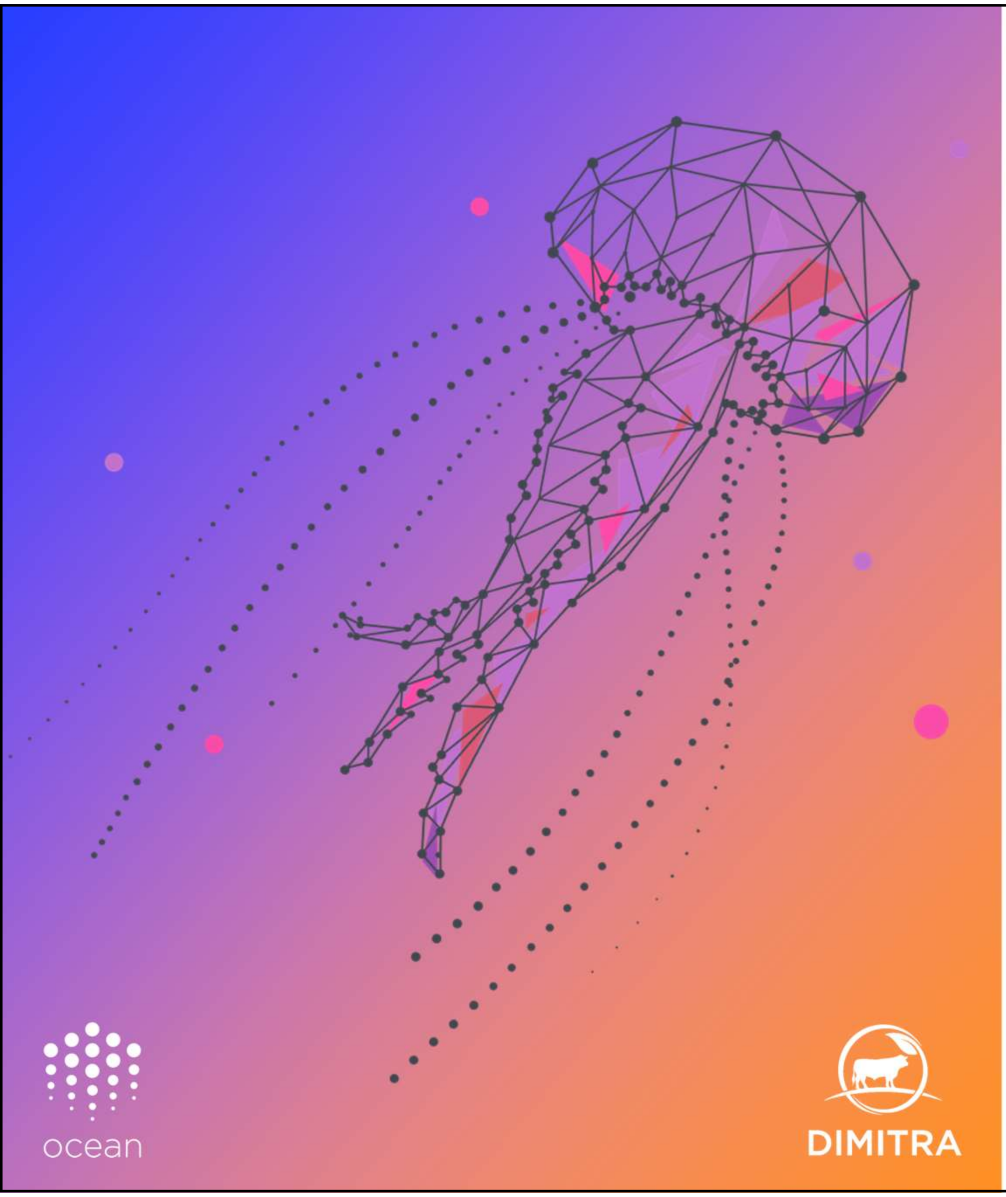


Table of contents

Introduction

The economy of Madhya Pradesh is an agrarian economy, which means that it is mainly based on the production and maintenance of crops and agricultural land. Agricultural yield is therefore a priority for its citizens who depend on it and especially the yield of soybean crops, the second most important crop in this state after wheat. In this sense, the prediction of soybean crops in Madhya Pradesh would be a real progress for agricultural management and farmers. So that is what we are going to do here using data and machine learning.

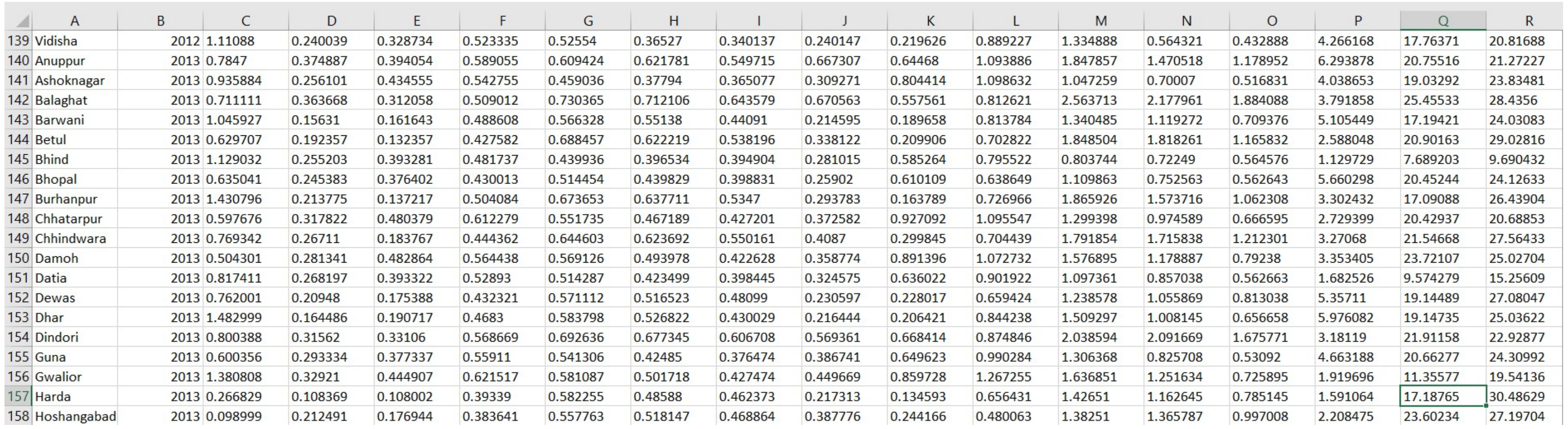
Understanding of the data

Here are the descriptions and remarks that we can attribute to each feature in order to understand their potential impact on yield :

- YIELD : Annual soybean yields by district in tons/hectares (Soybean is sown in June and harvested in November)
- NDVI : Normalized Difference Vegetation Index (It allows to determine the health of the vegetation by measuring the chlorophyll content of the plants via sensors embedded on satellites). Here it is not specific to soybean crops.
- LAI : Leaf Area Index (It quantifies the amount of leaf material in a canopy). Here it is not specific to soybean crops.
- ET : Evapotranspiration (It is the transfer of a quantity of water to the atmosphere, by evaporation from the soil and by transpiration from plants.). Here it is not specific to soybean crops. (in mm per unit time)
- LST : Land Surface Temperature (in Kelvin)
- RF : Rainfall (in mm/m^2)

Cleaning of the data

Since the data is already clean, we just need to fill in the only missing data in the dataset. This missing data is the "ET_JUL" feature of the Harda district in 2013. So I replaced it with the average of the ET_JUL of this district for the other years :



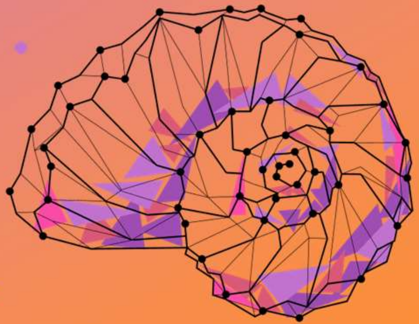
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
139	Vidisha	2012	1.11088	0.240039	0.328734	0.523335	0.52554	0.36527	0.340137	0.240147	0.219626	0.889227	1.334888	0.564321	0.432888	4.266168	17.76371	20.81688
140	Anuppur	2013	0.7847	0.374887	0.394054	0.589055	0.609424	0.621781	0.549715	0.667307	0.64468	1.093886	1.847857	1.470518	1.178952	6.293878	20.75516	21.27227
141	Ashoknagar	2013	0.935884	0.256101	0.434555	0.542755	0.459036	0.37794	0.365077	0.309271	0.804414	1.098632	1.047259	0.70007	0.516831	4.038653	19.03292	23.83481
142	Balaghat	2013	0.711111	0.363668	0.312058	0.509012	0.730365	0.712106	0.643579	0.670563	0.557561	0.812621	2.563713	2.177961	1.884088	3.791858	25.45533	28.4356
143	Barwani	2013	1.045927	0.15631	0.161643	0.488608	0.566328	0.55138	0.44091	0.214595	0.189658	0.813784	1.340485	1.119272	0.709376	5.105449	17.19421	24.03083
144	Betul	2013	0.629707	0.192357	0.132357	0.427582	0.688457	0.622219	0.538196	0.338122	0.209906	0.702822	1.848504	1.818261	1.165832	2.588048	20.90163	29.02816
145	Bhind	2013	1.129032	0.255203	0.393281	0.481737	0.439936	0.396534	0.394904	0.281015	0.585264	0.795522	0.803744	0.72249	0.564576	1.129729	7.689203	9.690432
146	Bhopal	2013	0.635041	0.245383	0.376402	0.430013	0.514454	0.439829	0.398831	0.25902	0.610109	0.638649	1.109863	0.752563	0.562643	5.660298	20.45244	24.12633
147	Burhanpur	2013	1.430796	0.213775	0.137217	0.504084	0.673653	0.637711	0.5347	0.293783	0.163789	0.726966	1.865926	1.573716	1.062308	3.302432	17.09088	26.43904
148	Chhatarpur	2013	0.597676	0.317822	0.480379	0.612279	0.551735	0.467189	0.427201	0.372582	0.927092	1.095547	1.299398	0.974589	0.666595	2.729399	20.42937	20.68853
149	Chhindwara	2013	0.769342	0.26711	0.183767	0.444362	0.644603	0.623692	0.550161	0.4087	0.299845	0.704439	1.791854	1.715838	1.212301	3.27068	21.54668	27.56433
150	Damoh	2013	0.504301	0.281341	0.482864	0.564438	0.569126	0.493978	0.422628	0.358774	0.891396	1.072732	1.576895	1.178887	0.79238	3.353405	23.72107	25.02704
151	Datia	2013	0.817411	0.268197	0.393322	0.52893	0.514287	0.423499	0.398445	0.324575	0.636022	0.901922	1.097361	0.857038	0.562663	1.682526	9.574279	15.25609
152	Dewas	2013	0.762001	0.20948	0.175388	0.432321	0.571112	0.516523	0.48099	0.230597	0.228017	0.659424	1.238578	1.055869	0.813038	5.35711	19.14489	27.08047
153	Dhar	2013	1.482999	0.164486	0.190717	0.4683	0.583798	0.526822	0.430029	0.216444	0.206421	0.844238	1.509297	1.008145	0.656658	5.976082	19.14735	25.03622
154	Dindori	2013	0.800388	0.31562	0.33106	0.568669	0.692636	0.677345	0.606708	0.569361	0.668414	0.874846	2.038594	2.091669	1.675771	3.18119	21.91158	22.92877
155	Guna	2013	0.600356	0.293334	0.377337	0.55911	0.541306	0.42485	0.376474	0.386741	0.649623	0.990284	1.306368	0.825708	0.53092	4.663188	20.66277	24.30992
156	Gwalior	2013	1.380808	0.32921	0.444907	0.621517	0.581087	0.501718	0.427474	0.449669	0.859728	1.267255	1.636851	1.251634	0.725895	1.919696	11.35577	19.54136
157	Harda	2013	0.266829	0.108369	0.108002	0.39339	0.582255	0.48588	0.462373	0.217313	0.134593	0.656431	1.42651	1.162645	0.785145	1.591064	17.18765	30.48629
158	Hoshangabad	2013	0.098999	0.212491	0.176944	0.383641	0.557763	0.518147	0.468864	0.387776	0.244166	0.480063	1.38251	1.365787	0.997008	2.208475	23.60234	27.19704

ocean

DIMITRA

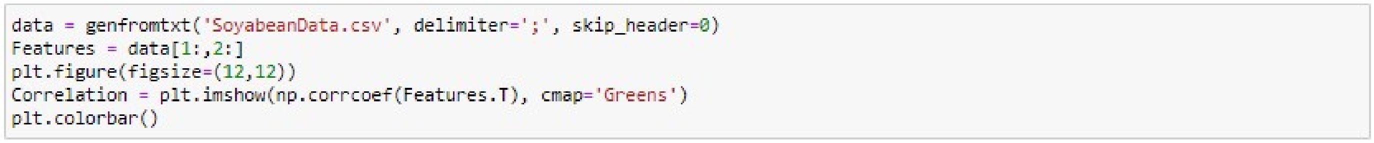
FEATURE SELECTION ANALYSIS

Graphical and Advanced Methods



Correlation between numerical variables

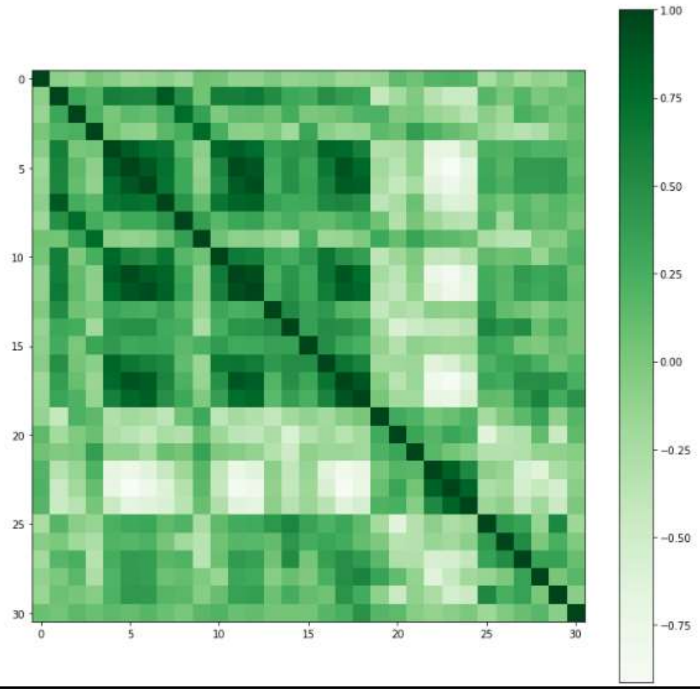
Code :



```
data = genfromtxt('SoyabeanData.csv', delimiter=';', skip_header=0)
Features = data[1:,2:]
plt.figure(figsize=(12,12))
Correlation = plt.imshow(np.corrcoef(Features.T), cmap='Greens')
plt.colorbar()
```

Output :

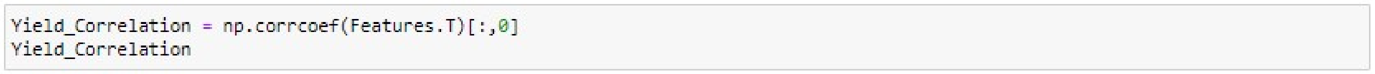
```
Out[13]: <matplotlib.colorbar.Colorbar at 0x2136d3e75b0>
```



We can observe a high correlation between some features of the dataset. But what we are interested in here is the correlation between row 0 (the yields) and columns 1 to 30 (the features).

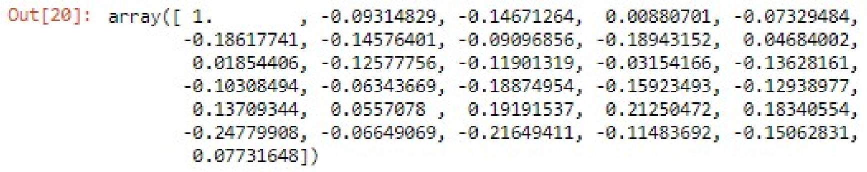
Ranking of variables based on their correlation

Code :



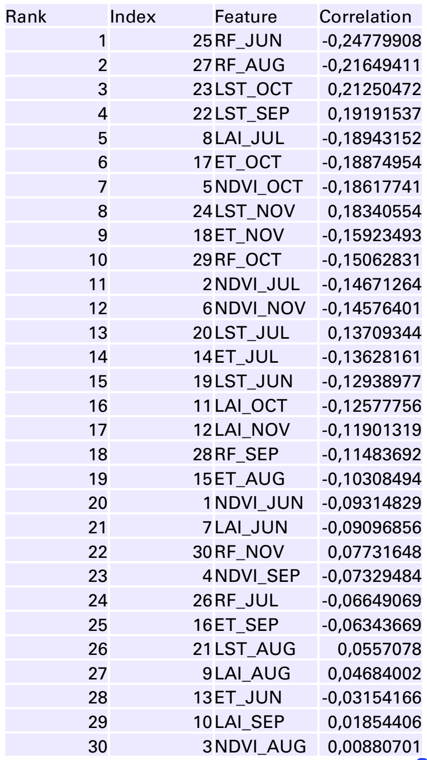
```
Yield_Correlation = np.corrcoef(Features.T)[:,0]
Yield_Correlation
```

Output :



```
Out[20]: array([ 1.        , -0.09314829, -0.14671264,  0.00880701, -0.07329484,
                -0.18617741, -0.14576401, -0.09096856, -0.18943152,  0.04684002,
                 0.01854406, -0.12577756, -0.11901319, -0.03154166, -0.13628161,
                -0.10308494, -0.06343669, -0.18874954, -0.15923493, -0.12938977,
                 0.13709344,  0.0557078 ,  0.19191537,  0.21250472,  0.18340554,
                -0.24779908, -0.06649069, -0.21649411, -0.11483692, -0.15062831,
                 0.07731648])
```

Rank :

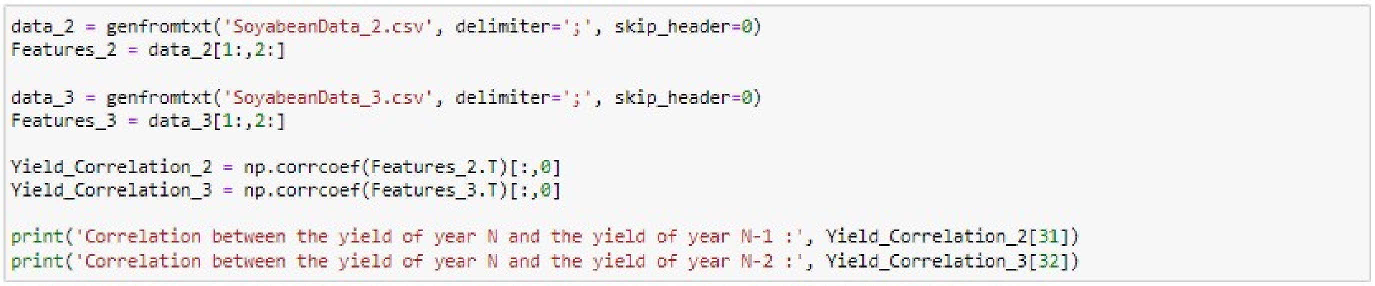


Rank	Index	Feature	Correlation
1	25	RF_JUN	-0,24779908
2	27	RF_AUG	-0,21649411
3	23	LST_OCT	0,21250472
4	22	LST_SEP	0,19191537
5	8	LAI_JUL	-0,18943152
6	17	ET_OCT	-0,18874954
7	5	NDVI_OCT	-0,18617741
8	24	LST_NOV	0,18340554
9	18	ET_NOV	-0,15923493
10	29	RF_OCT	-0,15062831
11	2	NDVI_JUL	-0,14671264
12	6	NDVI_NOV	-0,14576401
13	20	LST_JUL	0,13709344
14	14	ET_JUL	-0,13628161
15	19	LST_JUN	-0,12938977
16	11	LAI_OCT	-0,12577756
17	12	LAI_NOV	-0,11901319
18	28	RF_SEP	-0,11483692
19	15	ET_AUG	-0,10308494
20	1	NDVI_JUN	-0,09314829
21	7	LAI_JUN	-0,09096856
22	30	RF_NOV	0,07731648
23	4	NDVI_SEP	-0,07329484
24	26	RF_JUL	-0,06649069
25	16	ET_SEP	-0,06343669
26	21	LST_AUG	0,0557078
27	9	LAI_AUG	0,04684002
28	13	ET_JUN	-0,03154166
29	10	LAI_SEP	0,01854406
30	3	NDVI_AUG	0,00880701

Note : The ranking is based on the absolute value of the correlation between the target and the features because decorrelations are as important as correlations (an interesting feature is a feature that is farthest from 0, on the negative or positive side). Here, the feature that seems to be the most interesting for a model is the June rainfall feature which is however a decorrelation (the less it rains in June, the higher the yield).

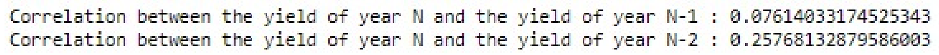
Correlation between the yield of year N and the yield of year N-1 / N-2

Code :



```
data_2 = genfromtxt('SoyabeanData_2.csv', delimiter=';', skip_header=0)
Features_2 = data_2[1:,2:]

data_3 = genfromtxt('SoyabeanData_3.csv', delimiter=';', skip_header=0)
Features_3 = data_3[1:,2:]

Yield_Correlation_2 = np.corrcoef(Features_2.T)[:,0]
Yield_Correlation_3 = np.corrcoef(Features_3.T)[:,0]

print('Correlation between the yield of year N and the yield of year N-1 :', Yield_Correlation_2[31])
print('Correlation between the yield of year N and the yield of year N-2 :', Yield_Correlation_3[32])
```

Output :

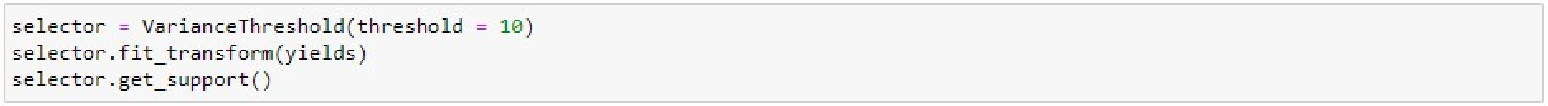


```
Correlation between the yield of year N and the yield of year N-1 : 0.07614033174525343
Correlation between the yield of year N and the yield of year N-2 : 0.25768132879586003
```

Note : We notice that the correlation between the yields of years N and N-2 is strong and better than any other features.

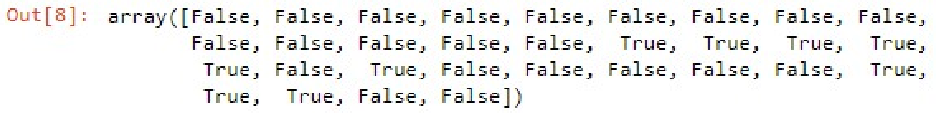
Feature selection : Advanced method (Variance threshold)

Code :



```
selector = VarianceThreshold(threshold = 10)
selector.fit_transform(yields)
selector.get_support()
```

Output :

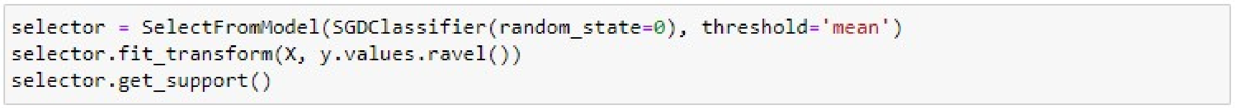


```
Out[8]: array([False, False, False, False, False, False, False, False, False,
               False, False, False, False, False,  True,  True,  True,  True,
                True, False,  True, False, False, False, False, False,  True,
                True,  True, False, False])
```

Features selected by the variance threshold : ET_JUL, ET_AUG, ET_SEP, ET_OCT, ET_NOV, LST_JUL, RF_JUL, RF_AUG, RF_SEP

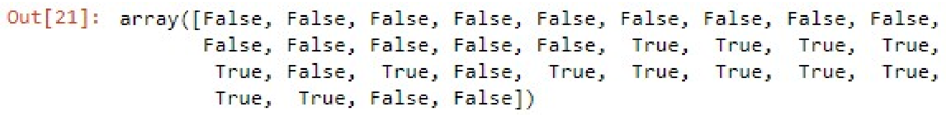
Feature selection for regression : Advanced method (SGDClassifier)

Code :



```
selector = SelectFromModel(SGDClassifier(random_state=0), threshold='mean')
selector.fit_transform(X, y.values.ravel())
selector.get_support()
```

Output :

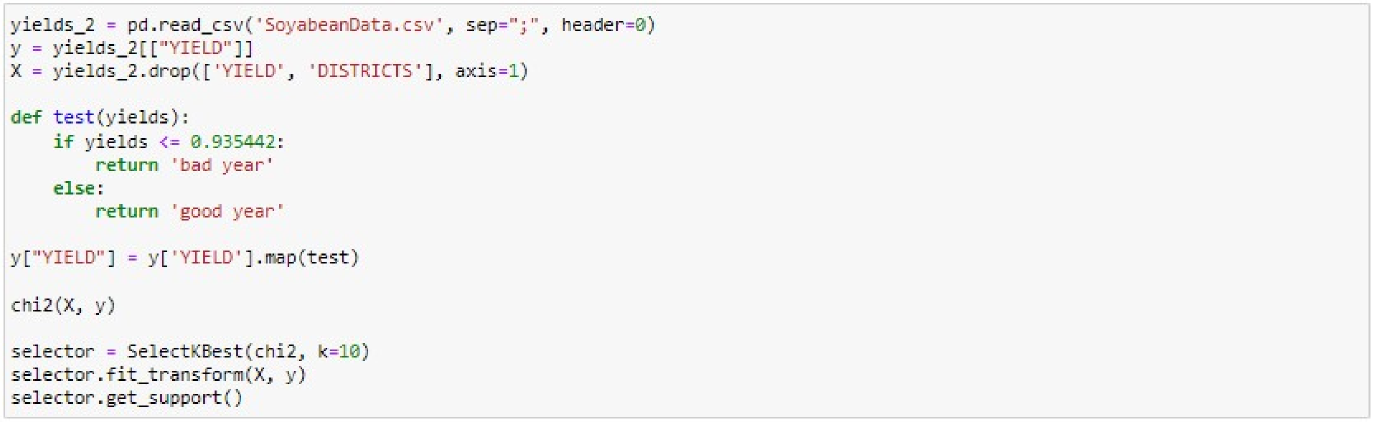


```
Out[21]: array([False, False, False, False, False, False, False, False, False,
                False, False, False, False, False,  True,  True,  True,  True,
                 True, False,  True, False,  True,  True,  True,  True,  True,
                 True,  True, False, False])
```

Features selected by SGDClassifier : ET_JUL, ET_AUG, ET_SEP, ET_OCT, ET_NOV, LST_JUL, LST_SEP, LST_OCT, LST_NOV, RF_JUN, RF_JUL, RF_AUG, RF_SEP

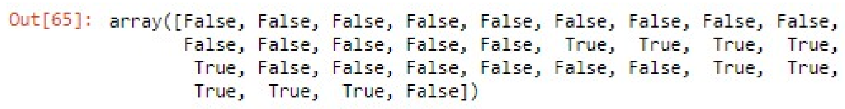
Feature selection for classification : Advanced method (Chi square)

Code :



```
yields_2 = pd.read_csv('SoyabeanData.csv', sep=";", header=0)
y = yields_2[["YIELD"]]
X = yields_2.drop(['YIELD', 'DISTRICTS'], axis=1)

def test(yields):
    if yields <= 0.935442:
        return 'bad year'
    else:
        return 'good year'

y["YIELD"] = y['YIELD'].map(test)

chi2(X, y)

selector = SelectKBest(chi2, k=10)
selector.fit_transform(X, y)
selector.get_support()
```

Output :



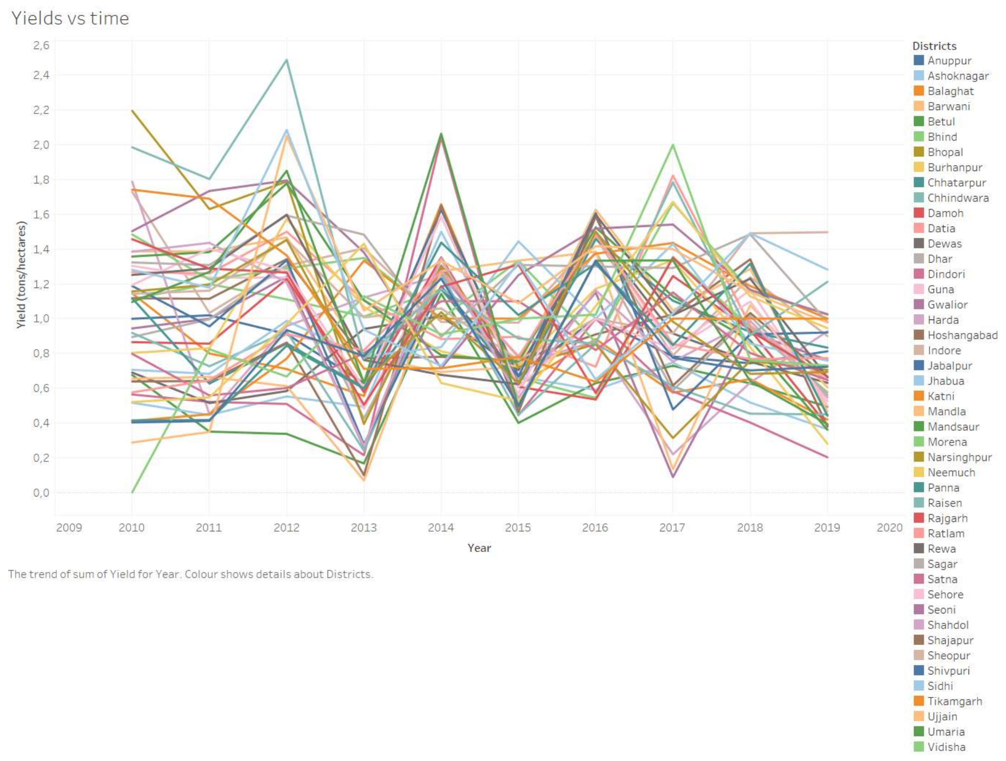
```
Out[65]: array([False, False, False, False, False, False, False, False, False,
                False, False, False, False, False,  True,  True,  True,  True,
                 True, False, False, False, False, False, False,  True,  True,
                 True,  True,  True, False])
```

Features selected by chi square : ET_JUL, ET_AUG, ET_SEP, ET_OCT, ET_NOV, RF_JUN, RF_JUL, RF_AUG, RF_SEP, RF_OCT

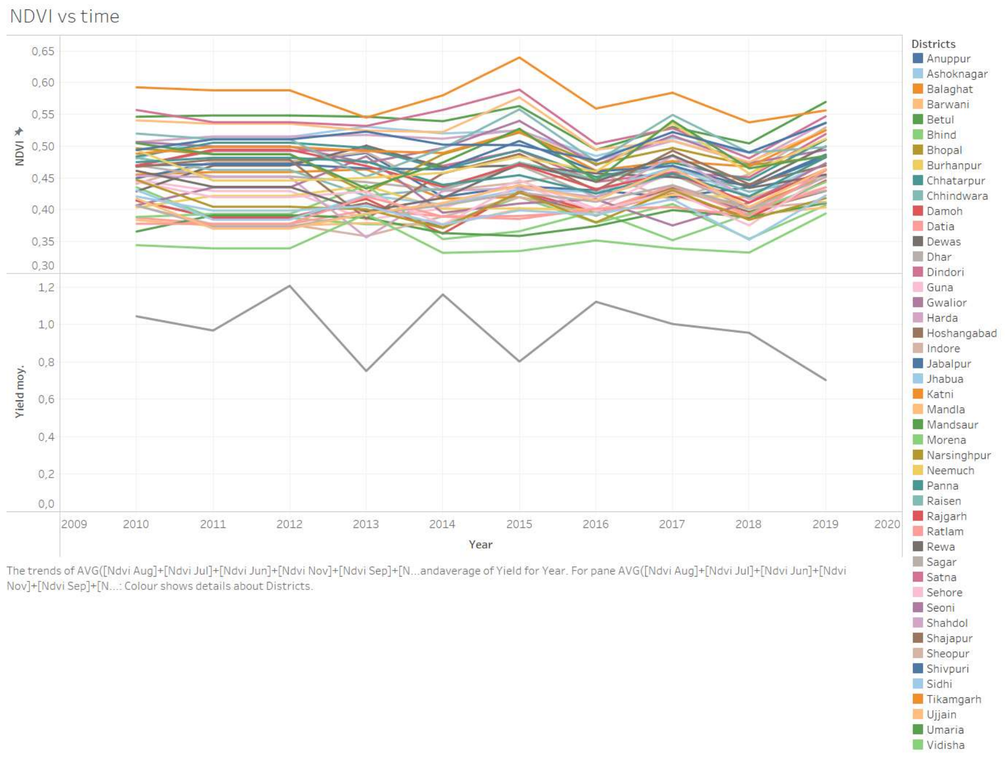
TIME SERIES ANALYSIS

Graphical Method

Yields vs time

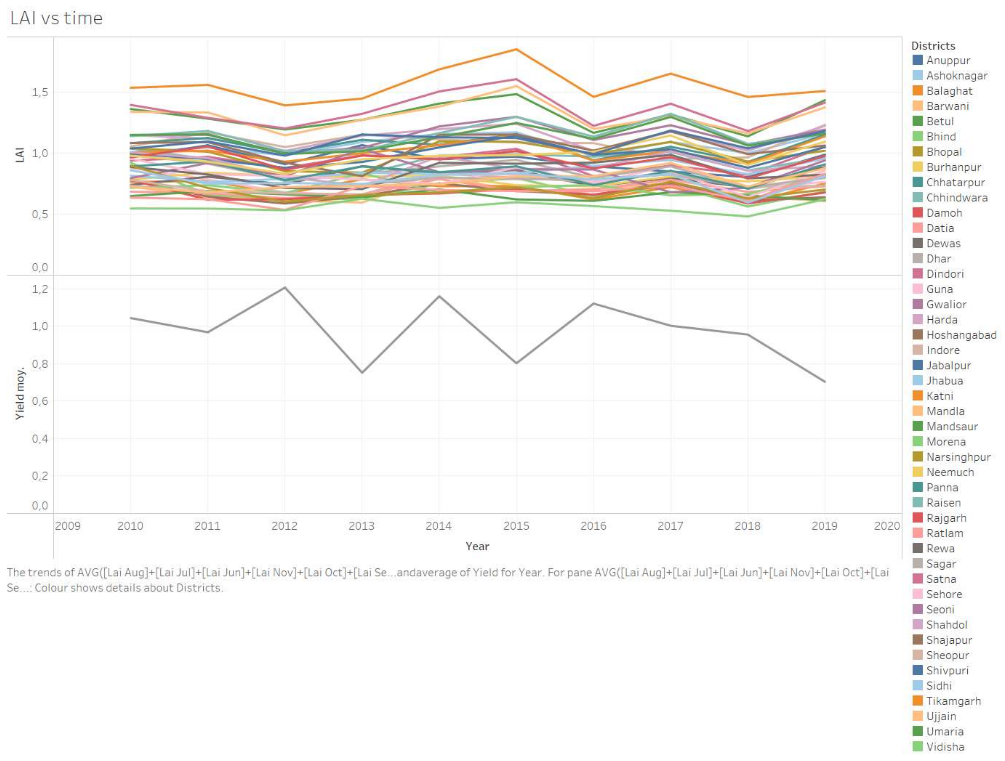


NDVI vs time



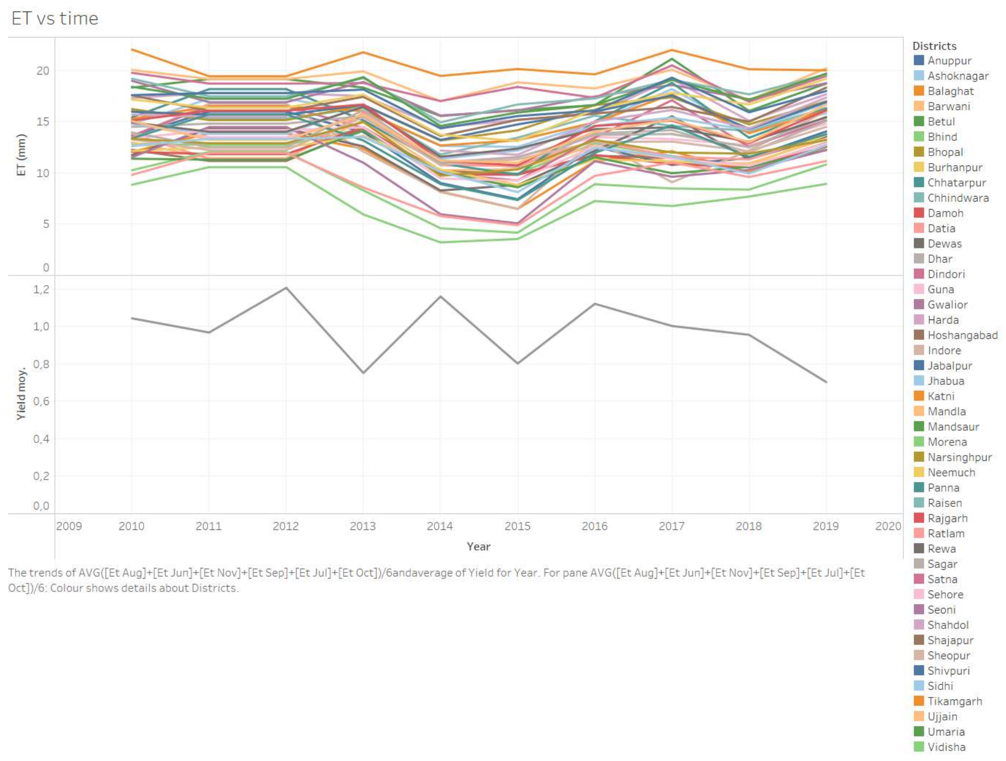
The trends of AVG([Ndvi Aug]+[Ndvi Jul]+[Ndvi Jun]+[Ndvi Nov]+[Ndvi Sep]+[N...andaverage of Yield for Year. For pane AVG([Ndvi Aug]+[Ndvi Jul]+[Ndvi Jun]+[Ndvi Nov]+[Ndvi Sep]+[N...: Colour shows details about Districts.

LAI vs time



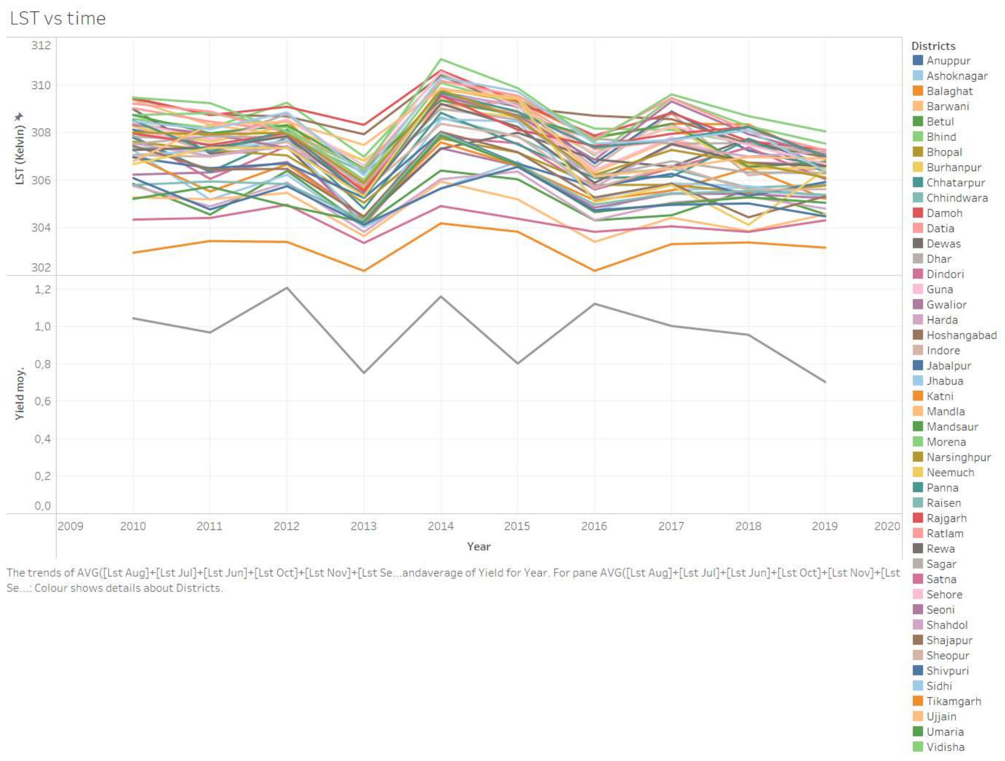
The trends of AVG([Lai Aug]+[Lai Jul]+[Lai Jun]+[Lai Nov]+[Lai Oct]+[Lai Se...andaverage of Yield for Year. For pane AVG([Lai Aug]+[Lai Jul]+[Lai Jun]+[Lai Nov]+[Lai Oct]+[Lai Se...: Colour shows details about Districts.

ET vs time



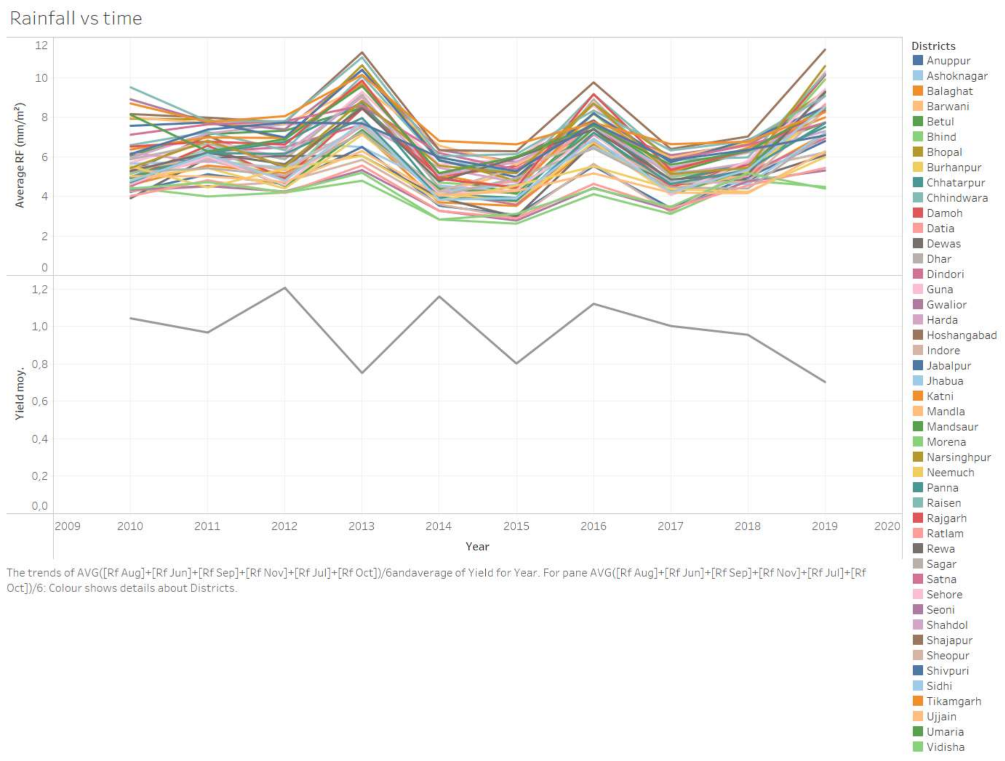
The trends of AVG([Et Aug]+[Et Jun]+[Et Nov]+[Et Sep]+[Et Jul]+[Et Oct])/6andaverage of Yield for Year. For pane AVG([Et Aug]+[Et Jun]+[Et Nov]+[Et Sep]+[Et Jul]+[Et Oct])/6: Colour shows details about Districts.

LST vs time



The trends of AVG([Lst Aug]+[Lst Jul]+[Lst Jun]+[Lst Oct]+[Lst Nov]+[Lst Se...andaverage of Yield for Year. For pane AVG([Lst Aug]+[Lst Jul]+[Lst Jun]+[Lst Oct]+[Lst Nov]+[Lst Se...: Colour shows details about Districts.

RF vs time



The trends of AVG([Rf Aug]+[Rf Jun]+[Rf Sep]+[Rf Nov]+[Rf Jul]+[Rf Oct])/6andaverage of Yield for Year. For pane AVG([Rf Aug]+[Rf Jun]+[Rf Sep]+[Rf Nov]+[Rf Jul]+[Rf Oct])/6: Colour shows details about Districts.

Observations

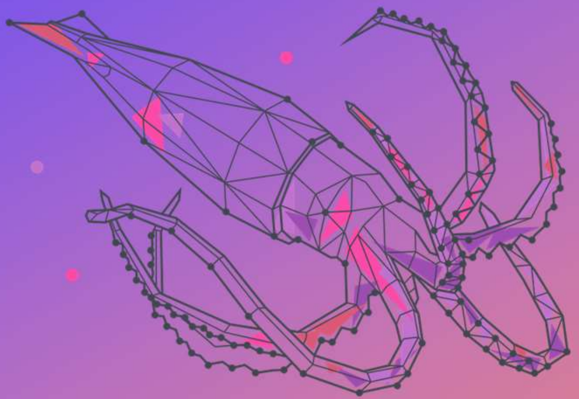
- The yield is between 0 and 2.5 tons/hectare.
- The yield seems similar to the one 2 years earlier (oscillation).
- Rainfall appears to be decorrelated with yield.
- The LST appears to correlate with yield.
- There is not much difference between the yields of the different districts in the same year.

ocean

DIMITRA

MACHINE LEARNING

Advanced Method

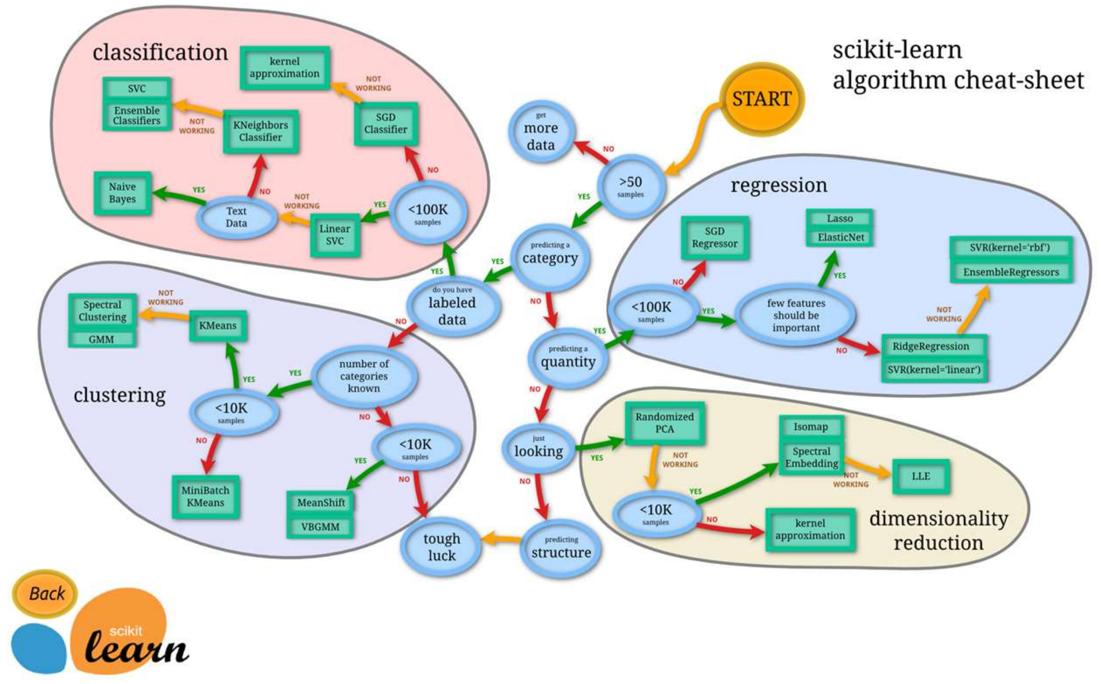


Usable models

With the data we have, we can use these models:

<u>Regression models:</u>
- RidgeRegression
- SVR(kernel='linear')
- SVR(kernel='rbf')
- EnsembleRegressors

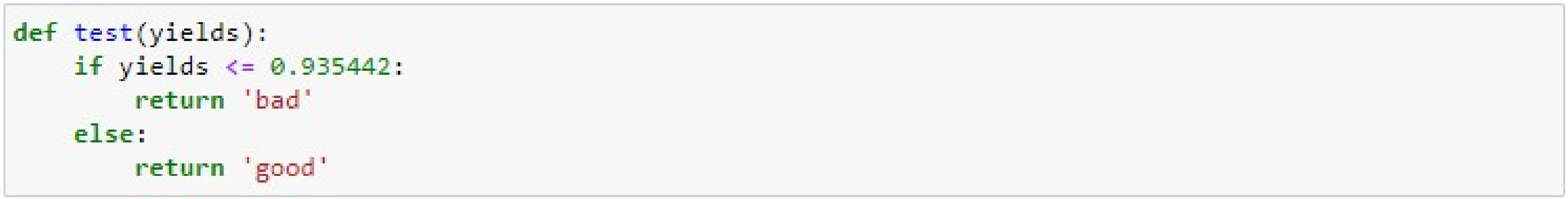
<u>Classification models :</u>
- K-nearest neighbors



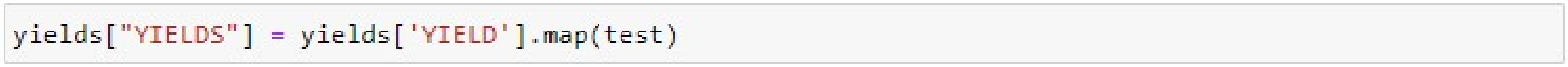
Selected model

Because of the weak correlations between the variables of the dataset (see Correlation slide), the regression models do not obtain satisfying scores.

For this reason, we will transform this regression problem (numerical value of yield) into a classification problem. To do this, we will predict whether the yield will be good (> the median yield) or bad (<= the median yield). With the following code, we can transform the problem :



```python
def test(yields):
    if yields <= 0.935442:
        return 'bad'
    else:
        return 'good'
```



```python
yields["YIELDS"] = yields['YIELD'].map(test)
```

We will therefore use the only usable classification model, which is the K-nearest neighbors model.

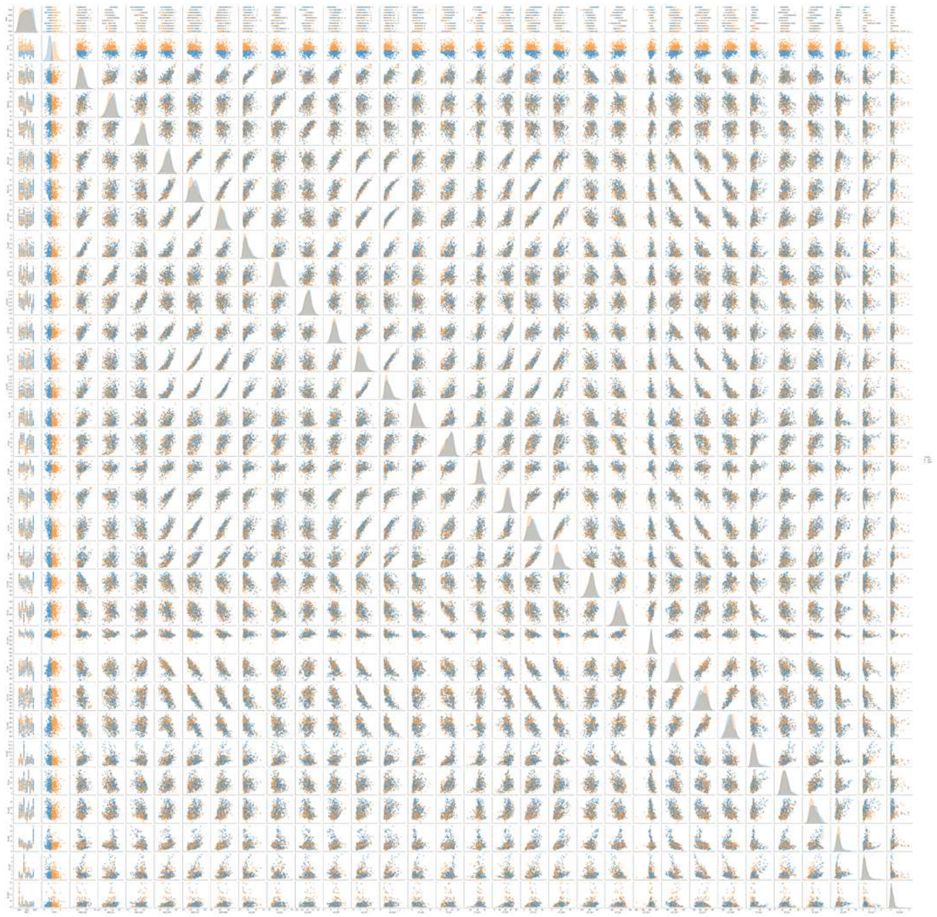
Why the K-nearest neighbors model could work ?

Code :

```
sns.pairplot(yield_3, hue="YIELDS")
```

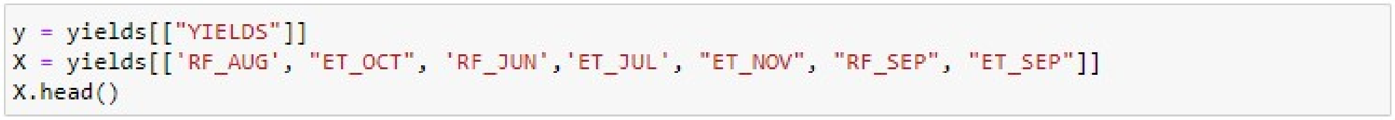
This code represents the features 2 to 2 and puts in orange the good yields (> than the yield median) and in blue the bad yields (<= to the yield median).

We can see that the years with a good yield and the one with a bad yield are sometimes grouped together. We can therefore use a model named K-nearest neighbors, this model tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closet to the test data.



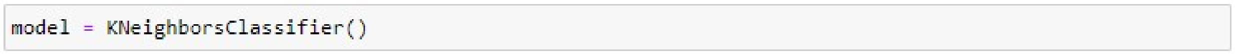
K-nearest neighbors model

Let's define two matrices, "y" is the one of the target (good or bad yield) and "X" is the one of the features we have selected previously by the Chi square method :



```
y = yields[["YIELDS"]]
X = yields[['RF_AUG', "ET_OCT", 'RF_JUN','ET_JUL', "ET_NOV", "RF_SEP", "ET_SEP"]]
X.head()
```

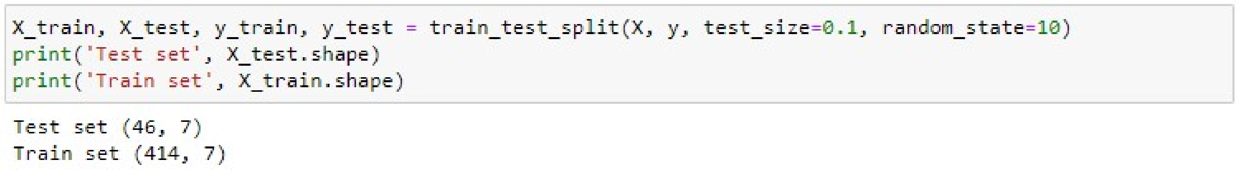
Let's then select the K-nearest neighbors model :



```
model = KNeighborsClassifier()
```

Train set, test set, training and first score

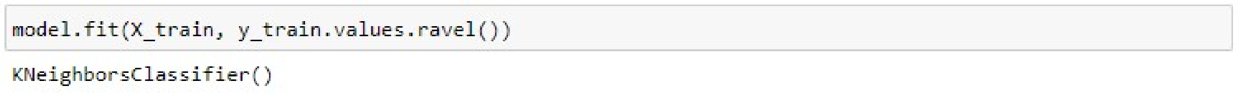
As their name indicates, the train set will be used to train our model while the test set will be used to evaluate it in order to know its accuracy. 90% of the data will be in the train set and 10% in the test set :



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=10)
print('Test set', X_test.shape)
print('Train set', X_train.shape)
```

```
Test set (46, 7)
Train set (414, 7)
```

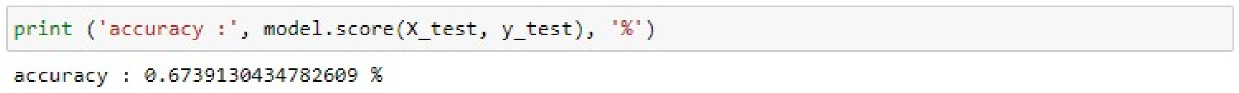
Let's train the model on the train set data :



```
model.fit(X_train, y_train.values.ravel())
```

```
KNeighborsClassifier()
```

Let's test the model on the test set data :



```
print ('accuracy :', model.score(X_test, y_test), '%')
```

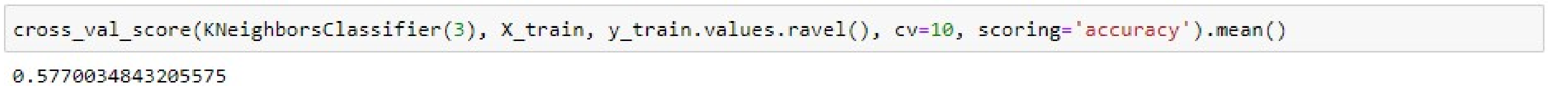
```
accuracy : 0.6739130434782609 %
```

We obtain an accuracy of 67%, which means that 67% of the predictions (good or bad yield) are good. This is already a good score but we can improve it.

Hyperparameters selection using Cross validation

In order to improve the accuracy of the model, we will modify the hyperparameters of the K-nearest neighbors model, namely the number of neighbors to be considered as well as the type of distance (Euclidian or Manhattan). To do this, we will validate our model on different cuts (validation set) of the train set (cv=10) and then average the results obtained for each hyper-parameter value to determine the best model.

For example, here is the result of this cross validation for a number of neighbors equal to 3 and a Euclidean distance :

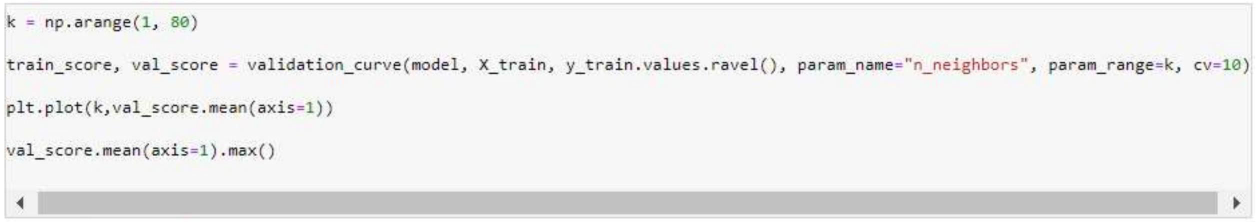


```
cross_val_score(KNeighborsClassifier(3), X_train, y_train.values.ravel(), cv=10, scoring='accuracy').mean()
```

```
0.5770034843205575
```

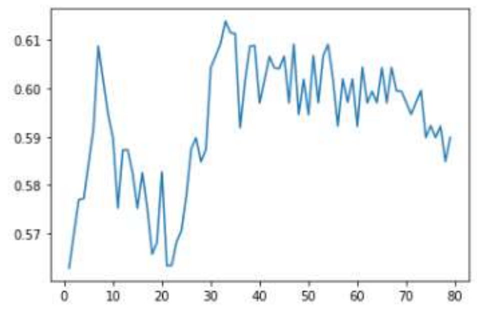
Validation Curve

Now, let's take the code from the previous slide and plot the results of the models for each value of the "neighbors" hyperparameter :



```
k = np.arange(1, 80)

train_score, val_score = validation_curve(model, X_train, y_train.values.ravel(), param_name="n_neighbors", param_range=k, cv=10)

plt.plot(k,val_score.mean(axis=1))

val_score.mean(axis=1).max()
```

```
0.6138792102206735
```



We therefore obtain this validation curve with the number of neighbors on the x-axis and the accuracy of the model on the validation sets on the y-axis. We can conclude that the most accurate model with a Euclidean distance is the one with a number of neighbors equal to 33 and an accuracy of 61,39% (on validation sets).

Hyperparameters selection using GridSearchCV + Cross validation

Now let's use GridSearchCV to determine the best model according to all parameters (number of neighbors and distance) :

```
param_grid = {'n_neighbors': np.arange(1,20), 'metric':['euclidean', 'manhattan']}

grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=10)

grid.fit(X_train, y_train.values.ravel())

grid.best_score_
```

```
0.6235191637630663
```

The best model obtains an accuracy of 62.35% on the validation sets, which is better than our previous result.

Let's look at the hyperparameters of this model :

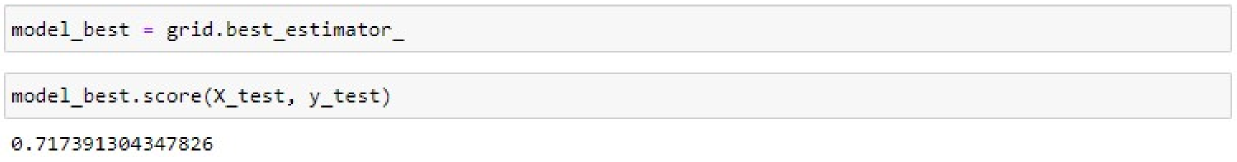
```
grid.best_params_
```

```
{'metric': 'manhattan', 'n_neighbors': 7}
```

We can conclude that the best model has the following hyperparameters: a Manhattan distance and a number of neighbors of 7.

Model’s accuracy

Let’s save this model in a variable and determine its accuracy on the test set (these data have never been seen by the model) :



```
model_best = grid.best_estimator_
```

```
model_best.score(X_test, y_test)
```

```
0.717391304347826
```

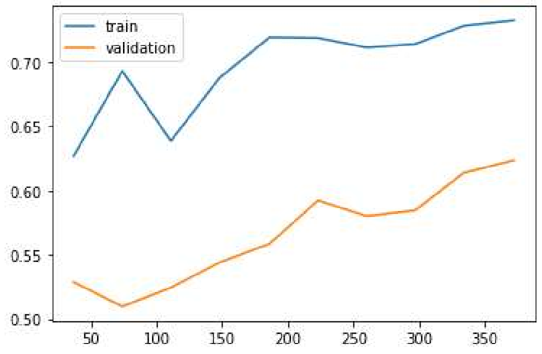
The model therefore predicts with an accuracy of 71.74% whether the yield will be good (> to the median of existing yield) or bad (<= to the median of existing yield).

Learning Curve

The learning curve shows the evolution of the accuracy of the model as a function of the number of examples it is trained with. A curve converging to a limit shows that the model does not need any additional data, it has reached its learning maximum. On the contrary, if the curve does not seem to converge, it means that the model could improve its accuracy with more data.

```
N, train_score, val_score =
learning_curve(model_best, X_train, y_train.values.ravel(), train_sizes = np.linspace(0.1, 1.0, 10), cv=10)

plt.plot(N, train_score.mean(axis=1), label='train')
plt.plot(N, val_score.mean(axis=1), label='validation')
plt.legend()
```

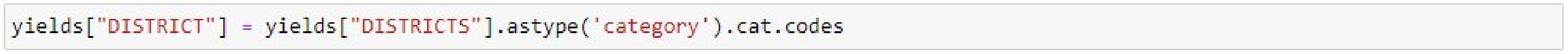
```
<matplotlib.legend.Legend at 0x25003ff6190>
```



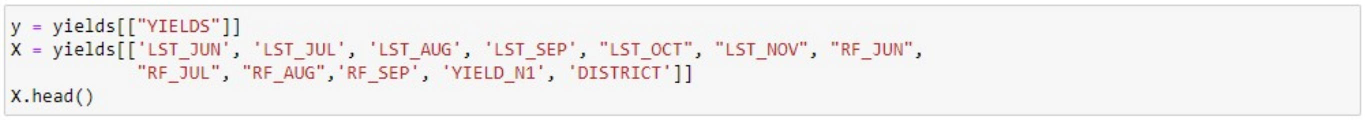
Here we notice that the learning curves of the model do not converge. We can therefore conclude that the model could improve its accuracy if we gave it more data to train.

Best Model

Finally, by trying hundreds of combinations of features, I found a better K-nearest neighbors model using the previous year's yield as a feature and districts transformed into numerical values :

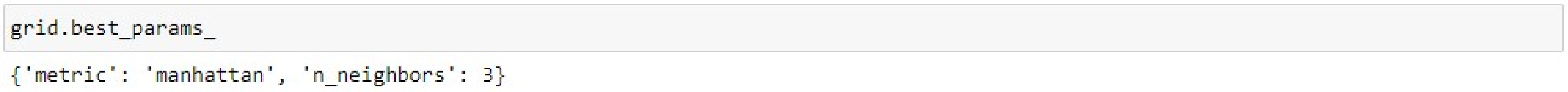


```
yields["DISTRICT"] = yields["DISTRICTS"].astype('category').cat.codes
```



```
y = yields[["YIELDS"]]
X = yields[['LST_JUN', 'LST_JUL', 'LST_AUG', 'LST_SEP', "LST_OCT", "LST_NOV", "RF_JUN",
           "RF_JUL", "RF_AUG",'RF_SEP', 'YIELD_N1', 'DISTRICT']]
X.head()
```

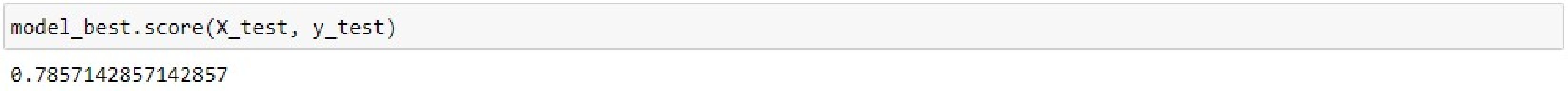
The hyperparameters of this model are the following :



```
grid.best_params_
```

```
{'metric': 'manhattan', 'n_neighbors': 3}
```

This model makes a good prediction in 78.57% of cases



```
model_best.score(X_test, y_test)
```

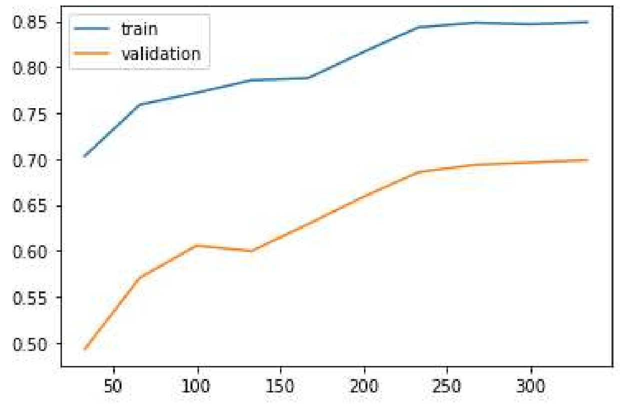
```
0.7857142857142857
```

Best Model : Learning Curve

By observing the learning curves of this last model, we can see that they converge. This means that this model does not require any additional data :

```
N, train_score, val_score =
learning_curve(model_best, X_train, y_train.values.ravel(), train_sizes = np.linspace(0.1, 1.0, 10), cv=10)

plt.plot(N, train_score.mean(axis=1), label='train')
plt.plot(N, val_score.mean(axis=1), label='validation')
plt.legend()
```

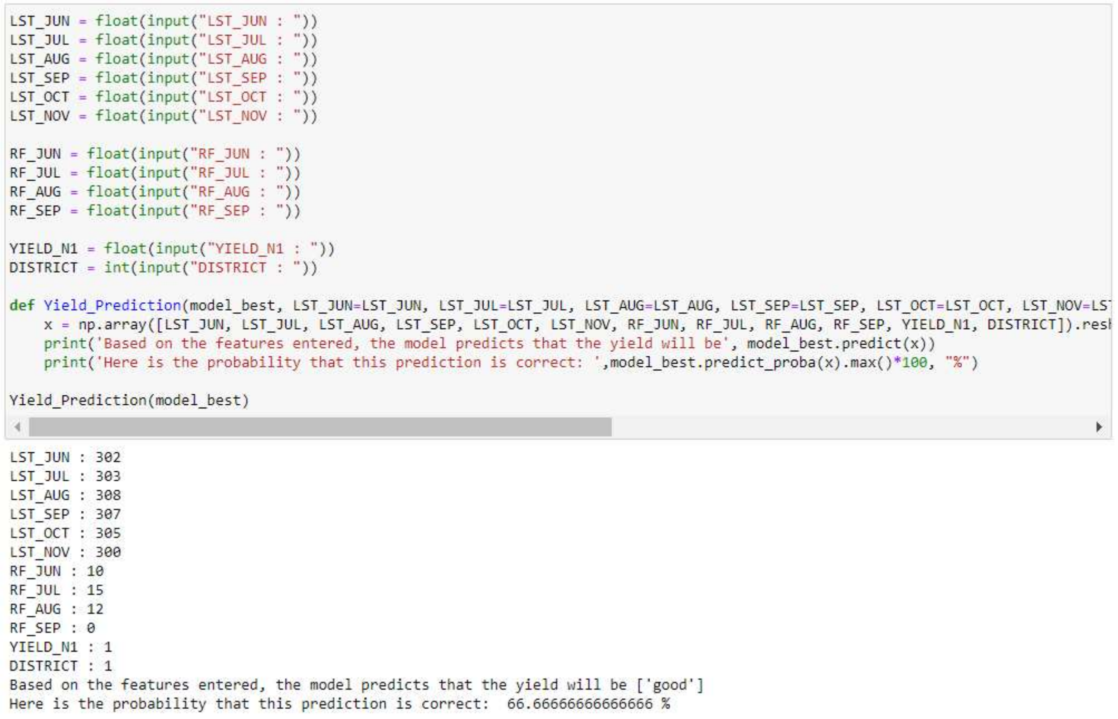
```
<matplotlib.legend.Legend at 0x1c4ce4aee50>
```



This model is therefore more accurate than the previous one but has no room for improvement.

Function to predict yield using the best model

Finally, I made a function to predict whether the yield will be good or bad depending on the features entered by the user. (Warning : you have to indicate the district number (Anuppur = 0, ..., Vidisha =43). Thus, farmers will be able to use my machine learning model easily, know the prediction and the probability that it will come true.



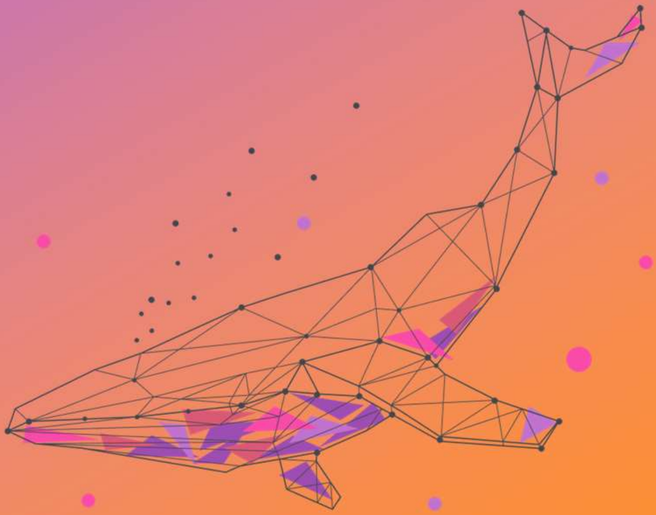
```
LST_JUN = float(input("LST_JUN : "))
LST_JUL = float(input("LST_JUL : "))
LST_AUG = float(input("LST_AUG : "))
LST_SEP = float(input("LST_SEP : "))
LST_OCT = float(input("LST_OCT : "))
LST_NOV = float(input("LST_NOV : "))

RF_JUN = float(input("RF_JUN : "))
RF_JUL = float(input("RF_JUL : "))
RF_AUG = float(input("RF_AUG : "))
RF_SEP = float(input("RF_SEP : "))

YIELD_N1 = float(input("YIELD_N1 : "))
DISTRICT = int(input("DISTRICT : "))

def Yield_Prediction(model_best, LST_JUN=LST_JUN, LST_JUL=LST_JUL, LST_AUG=LST_AUG, LST_SEP=LST_SEP, LST_OCT=LST_OCT, LST_NOV=LS
    x = np.array([LST_JUN, LST_JUL, LST_AUG, LST_SEP, LST_OCT, LST_NOV, RF_JUN, RF_JUL, RF_AUG, RF_SEP, YIELD_N1, DISTRICT]).res
    print('Based on the features entered, the model predicts that the yield will be', model_best.predict(x))
    print('Here is the probability that this prediction is correct: ',model_best.predict_proba(x).max()*100, "%")

Yield_Prediction(model_best)
```

```
LST_JUN : 302
LST_JUL : 303
LST_AUG : 308
LST_SEP : 307
LST_OCT : 305
LST_NOV : 300
RF_JUN : 10
RF_JUL : 15
RF_AUG : 12
RF_SEP : 0
YIELD_N1 : 1
DISTRICT : 1
Based on the features entered, the model predicts that the yield will be ['good']
Here is the probability that this prediction is correct:  66.66666666666666 %
```

ocean

DIMITRA

RESOURCES



Publish on Ocean Market

The improved and cleaned dataset used for the best model is available on the Ocean Market and can be consumed for free at this address :

https://market.oceanprotocol.com/asset/did:op:bca6f2c5ccb7d2af6527b194d6fd6792c76e81cedacebb00d16b8cd66f73b7dc

GitHub repository

This report, the 2 machine learning models, the 3 datasets and the notebook of the feature analysis can be found in the following GitHub repository :

https://github.com/Demoniarc/data-challenge-2.git