In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

In [2]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [3]:
```
import time
import math
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt

from scipy import stats

%matplotlib inline
```

In [4]:
```
pd.read_excel('D:\MTU KHARIF AND RABI DATA.xlsx')
```

Out[4]:

	YEAR	SMW	YSB	GM	LF	BPH	TMAX	TMIN	RF	RHM	RHE	SSH
0	2003	1	1304	15	0	175	27.71	19.50	1.00	90.57	68.57	3.69
1	2003	2	300	15	1	200	27.29	18.21	0.00	91.57	61.43	1.43
2	2003	3	215	0	0	112	26.00	16.43	0.00	84.86	66.29	4.39
3	2003	4	550	0	0	40	28.64	19.07	0.00	97.57	70.14	5.71
4	2003	5	176	1	1	28	29.07	19.29	0.00	94.43	72.85	5.50
...	...	...	...	...	...	...	...	...	...	...	...	...
395	2022	16	1709	190	48	1134	35.00	26.00	7.20	85.85	44.70	8.37
396	2022	17	4593	180	112	891	36.20	27.00	0.00	84.30	45.70	8.11
397	2022	18	3012	118	0	5558	34.13	25.09	0.90	86.79	49.53	8.51
398	2022	19	2014	93	0	3011	34.44	25.43	1.01	86.40	48.54	6.93
399	2022	20	2077	99	0	2400	34.38	25.78	1.14	86.13	48.31	6.81

400 rows × 12 columns

In [5]:
```
rice_data = pd.read_excel('D:\MTU KHARIF AND RABI DATA.xlsx')
rice_data
```

Out[5]:

	YEAR	SMW	YSB	GM	LF	BPH	TMAX	TMIN	RF	RHM	RHE	SSH
0	2003	1	1304	15	0	175	27.71	19.50	1.00	90.57	68.57	3.69
1	2003	2	300	15	1	200	27.29	18.21	0.00	91.57	61.43	1.43
2	2003	3	215	0	0	112	26.00	16.43	0.00	84.86	66.29	4.39
3	2003	4	550	0	0	40	28.64	19.07	0.00	97.57	70.14	5.71
4	2003	5	176	1	1	28	29.07	19.29	0.00	94.43	72.85	5.50
...	...	...	...	...	...	...	...	...	...	...	...	...
395	2022	16	1709	190	48	1134	35.00	26.00	7.20	85.85	44.70	8.37
396	2022	17	4593	180	112	891	36.20	27.00	0.00	84.30	45.70	8.11
397	2022	18	3012	118	0	5558	34.13	25.09	0.90	86.79	49.53	8.51
398	2022	19	2014	93	0	3011	34.44	25.43	1.01	86.40	48.54	6.93
399	2022	20	2077	99	0	2400	34.38	25.78	1.14	86.13	48.31	6.81

400 rows × 12 columns

EXPLORATORY DATA ANALYSIS

In [6]:
```
rice_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   YEAR    400 non-null    int64  
 1   SMW     400 non-null    int64  
 2   YSB     400 non-null    int64  
 3   GM      400 non-null    int64  
 4   LF      400 non-null    int64  
 5   BPH     400 non-null    int64  
 6   TMAX    400 non-null    float64
 7   TMIN    400 non-null    float64
 8   RF      400 non-null    float64
 9   RHM     400 non-null    float64
 10  RHE     400 non-null    float64
 11  SSH     400 non-null    float64
dtypes: float64(6), int64(6)
memory usage: 37.6 KB
```

In [7]:
```
rice_data.describe()
```

Out[7]:

	YEAR	SMW	YSB	GM	LF	BPH	TMAX	
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400
mean	2012.500000	10.500000	1264.652500	210.902500	14.295000	4669.070000	31.201700	22
std	5.773503	5.773503	2425.172834	432.191139	28.201619	9638.694624	2.923558	2
min	2003.000000	1.000000	0.000000	0.000000	0.000000	0.000000	19.300000	12
25%	2007.750000	5.750000	92.000000	15.000000	0.000000	188.250000	29.115000	20
50%	2012.500000	10.500000	322.000000	58.500000	4.000000	751.500000	31.380000	22
75%	2017.250000	15.250000	1281.250000	162.000000	16.000000	4203.500000	33.017500	24
max	2022.000000	20.000000	20648.000000	3000.000000	279.000000	77725.000000	42.780000	29

In [8]:

```
# Let us find out the datatypes of the data present in the dataset
rice_data.dtypes
```

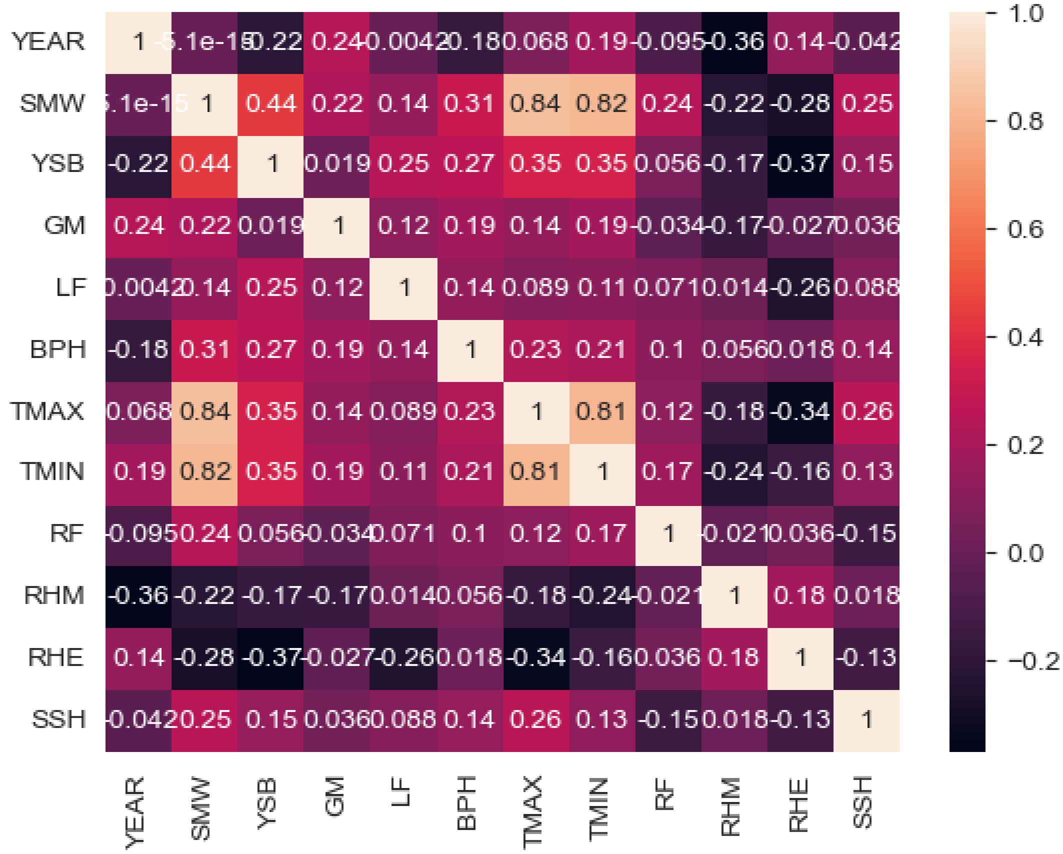
Out[8]:

```
YEAR      int64
SMW       int64
YSB       int64
GM        int64
LF        int64
BPH       int64
TMAX    float64
TMIN    float64
RF      float64
RHM     float64
RHE     float64
SSH     float64
dtype: object
```

In [9]:

```
import seaborn as sns
sns.set_style("whitegrid")
sns.heatmap(rice_data.corr(), annot= True)
```

Out[9]: <AxesSubplot:>



In [11]:
```
rice_data.isnull().sum()
```

Out[11]:
```
YEAR    0
SMW     0
YSB     0
GM      0
LF      0
BPH     0
TMAX    0
TMIN    0
RF      0
RHM     0
RHE     0
SSH     0
dtype: int64
```

In [13]:
```
object_frame = rice_data.select_dtypes(include = 'O')
numbers_frame = rice_data.select_dtypes(exclude = 'O')
object_frame
```

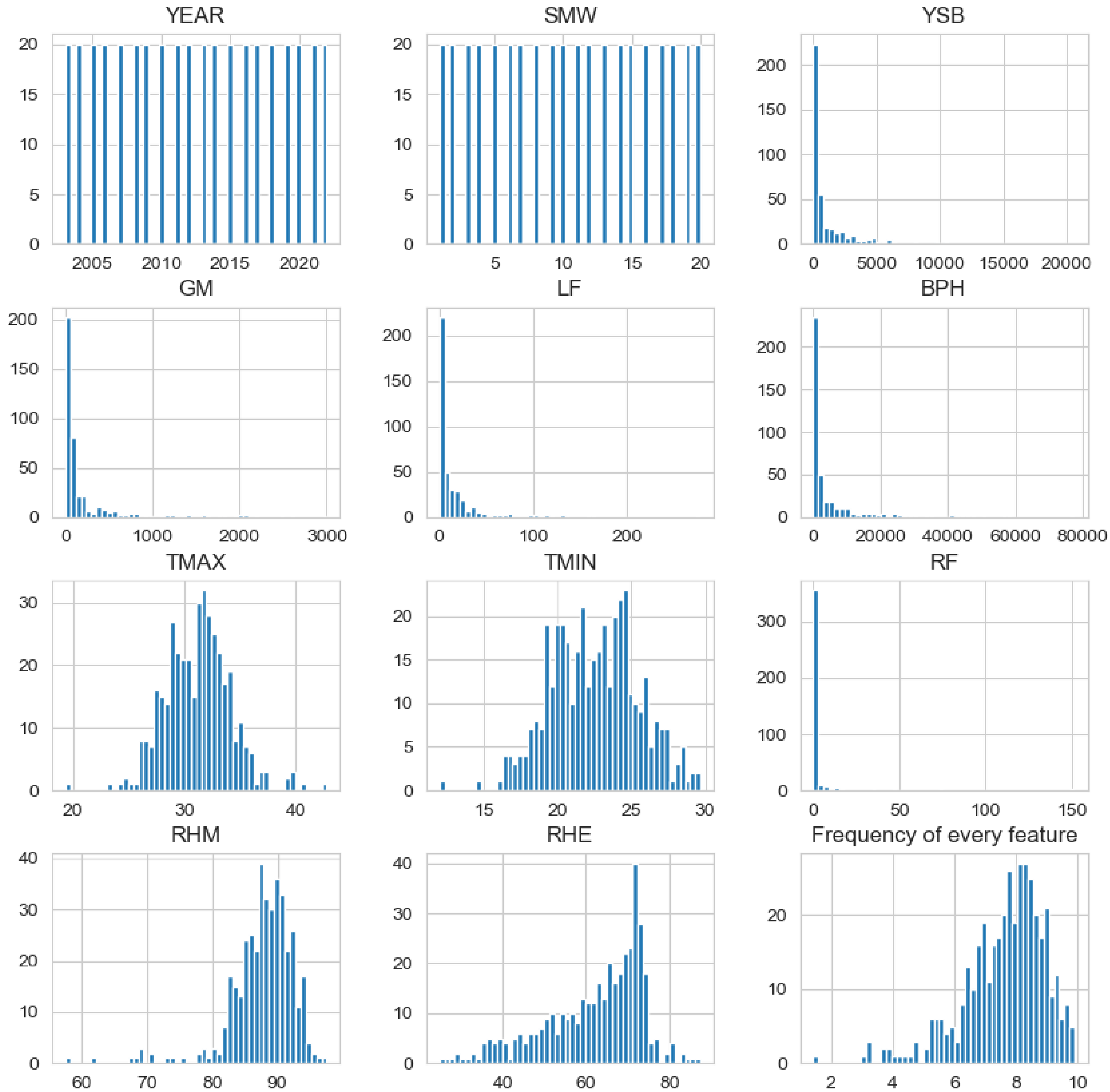
Out[13]:

0
1
2
3
4
...
395
396
397
398
399

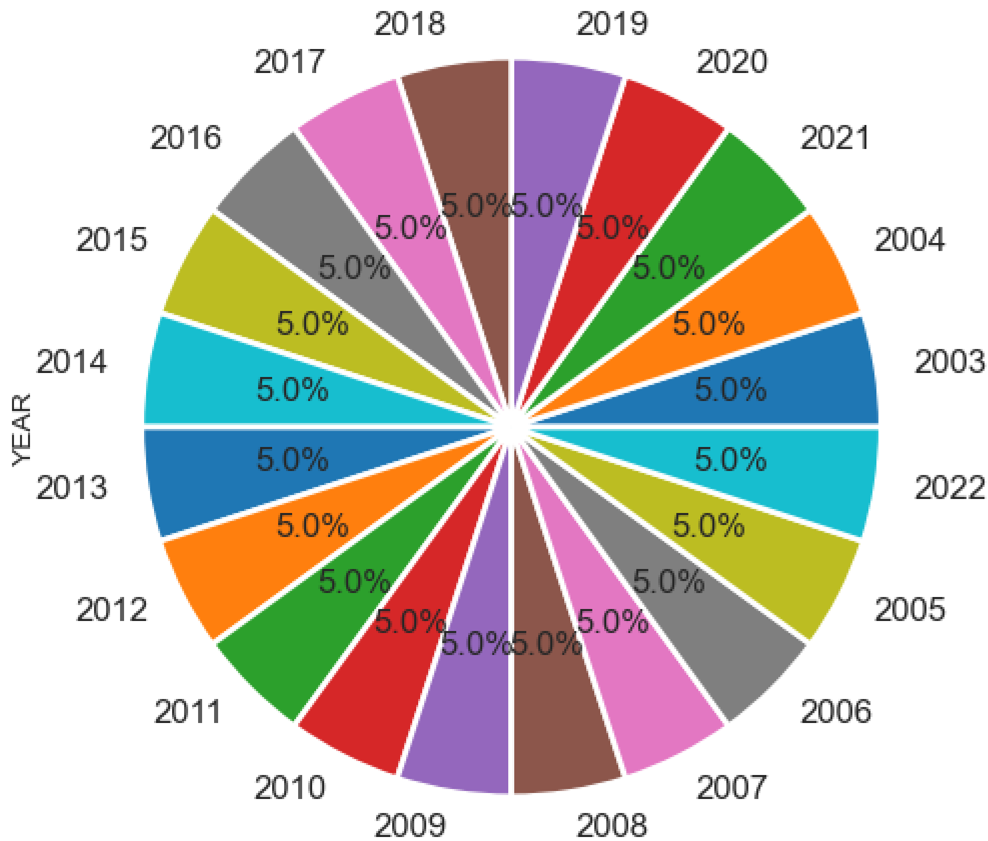
400 rows × 0 columns

In [14]:

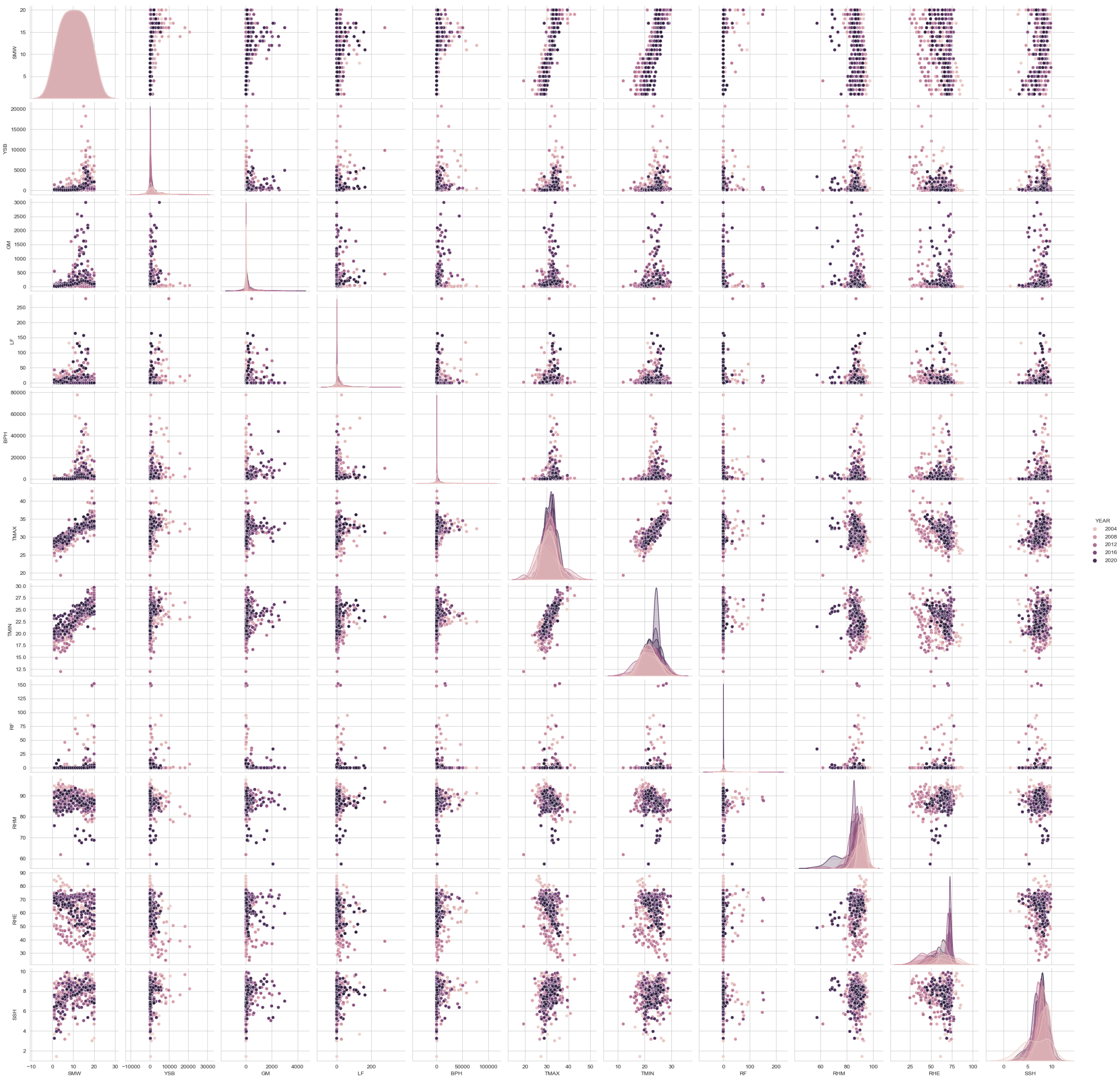
```
numbers_frame.hist(figsize=(10, 10), bins=50, xlabelsize=10, ylabelsize=10)
plt.title('Frequency of every feature');
```



In [15]:
```
plt.figure(figsize=(6,6))
ax = rice_data['YEAR'].value_counts().plot(kind='pie', autopct='%.1f%%', wedgeprops={'
```

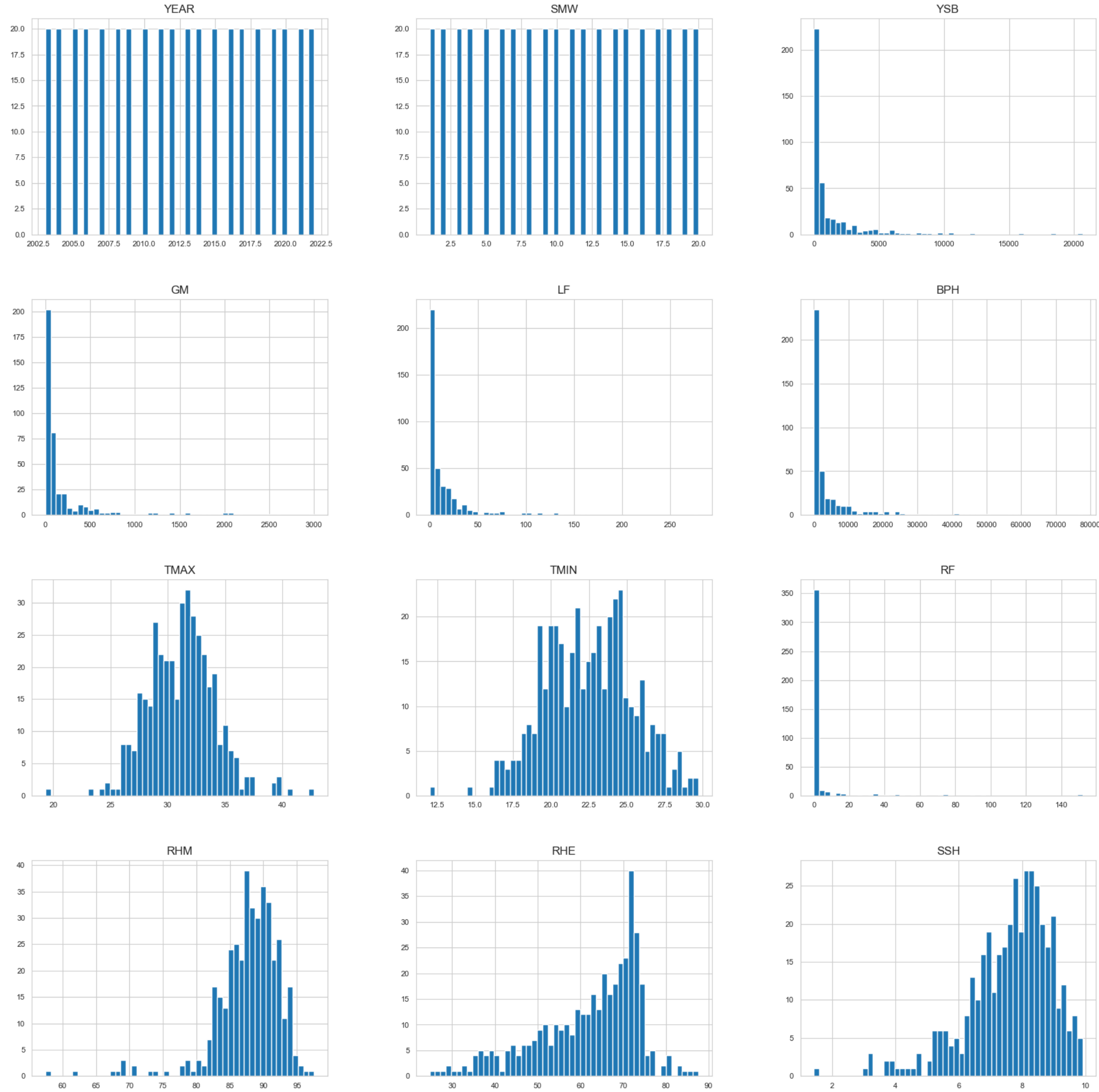


In [16]:
```
sns.pairplot(rice_data, hue='YEAR');
```



In [17]:
```
rice_data1 = rice_data[rice_data['SMW'] < 600]
rice_data1 = rice_data1[rice_data1['TMAX'] < 50]
rice_data1 = rice_data1[rice_data1['SSH'] < 20]
rice_data1 = rice_data1[rice_data1['RF'] < 200]
print(rice_data1.shape)
rice_data1.hist(figsize=(20, 20), bins=50, xlabelsize=8, ylabelsize=8);
```

```
(400, 12)
```



FEATURE ENGINEERING¶¶

In [24]:
```
model_data = rice_data1.drop(['SMW'], axis=1)
model_data
```

Out[24]:

	YEAR	YSB	GM	LF	BPH	TMAX	TMIN	RF	RHM	RHE	SSH
0	2003	1304	15	0	175	27.71	19.50	1.00	90.57	68.57	3.69
1	2003	300	15	1	200	27.29	18.21	0.00	91.57	61.43	1.43
2	2003	215	0	0	112	26.00	16.43	0.00	84.86	66.29	4.39
3	2003	550	0	0	40	28.64	19.07	0.00	97.57	70.14	5.71
4	2003	176	1	1	28	29.07	19.29	0.00	94.43	72.85	5.50
...	...	...	...	...	...	...	...	...	...	...	...
395	2022	1709	190	48	1134	35.00	26.00	7.20	85.85	44.70	8.37
396	2022	4593	180	112	891	36.20	27.00	0.00	84.30	45.70	8.11
397	2022	3012	118	0	5558	34.13	25.09	0.90	86.79	49.53	8.51
398	2022	2014	93	0	3011	34.44	25.43	1.01	86.40	48.54	6.93
399	2022	2077	99	0	2400	34.38	25.78	1.14	86.13	48.31	6.81

400 rows × 11 columns

In [25]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier

le1 = LabelEncoder()
model_data['YEAR'] = le1.fit_transform(model_data['YEAR'])

le2 = LabelEncoder()
model_data['GM'] = le2.fit_transform(model_data['GM'])

model_data
```

Out[25]:

	YEAR	YSB	GM	LF	BPH	TMAX	TMIN	RF	RHM	RHE	SSH
0	0	1304	15	0	175	27.71	19.50	1.00	90.57	68.57	3.69
1	0	300	15	1	200	27.29	18.21	0.00	91.57	61.43	1.43
2	0	215	0	0	112	26.00	16.43	0.00	84.86	66.29	4.39
3	0	550	0	0	40	28.64	19.07	0.00	97.57	70.14	5.71
4	0	176	1	1	28	29.07	19.29	0.00	94.43	72.85	5.50
...	...	...	...	...	...	...	...	...	...	...	...
395	19	1709	111	48	1134	35.00	26.00	7.20	85.85	44.70	8.37
396	19	4593	109	112	891	36.20	27.00	0.00	84.30	45.70	8.11
397	19	3012	90	0	5558	34.13	25.09	0.90	86.79	49.53	8.51
398	19	2014	76	0	3011	34.44	25.43	1.01	86.40	48.54	6.93
399	19	2077	80	0	2400	34.38	25.78	1.14	86.13	48.31	6.81

400 rows × 11 columns

In [26]:
```
final_model_data = pd.get_dummies(data=model_data, drop_first=True)
final_model_data.head()
```

Out[26]:

	YEAR	YSB	GM	LF	BPH	TMAX	TMIN	RF	RHM	RHE	SSH
0	0	1304	15	0	175	27.71	19.50	1.0	90.57	68.57	3.69
1	0	300	15	1	200	27.29	18.21	0.0	91.57	61.43	1.43
2	0	215	0	0	112	26.00	16.43	0.0	84.86	66.29	4.39
3	0	550	0	0	40	28.64	19.07	0.0	97.57	70.14	5.71
4	0	176	1	1	28	29.07	19.29	0.0	94.43	72.85	5.50

In [27]:
```
final_model_data.dtypes
```

Out[27]:
```
YEAR      int64
YSB       int64
GM        int64
LF        int64
BPH       int64
TMAX    float64
TMIN    float64
RF      float64
RHM     float64
RHE     float64
SSH     float64
dtype: object
```

In [28]:
```
# Now that we have all the columns in numerical datatype let us try to train the model
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

```
Y = final_model_data['GM']
X = final_model_data.drop('GM', axis=1)
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.3)
```

In [29]:
```
import warnings
warnings.filterwarnings("ignore")

def best_model(x,y):
    algos = {
        'logistic_regression' : {
            'model': LogisticRegression(),
            'params':{'fit_intercept': [True, False]}
        },

#        'SVC' : {
#            'model': SVC(),
#            'params':{
#                'kernel' : ['rbf', 'poly', 'sigmoid'],
#                'degree': [2,3,4],
#                'gamma': ['scale', 'auto', 0.22]
#             }
#        },
        'decision_tree' : {
            'model': DecisionTreeClassifier(),
            'params':{
                'criterion': ['gini', 'entropy', 'log_loss'],
                'max_features': ['auto', 'sqrt', 'log2']
            }
        },
        'Adaboost': {
            'model': AdaBoostClassifier(),
            'params':{
                'n_estimators': [50,100],
                'learning_rate': [1,1.5,2]
            }
        },
        'GaussionNB':{
            'model': GaussianNB(),
            'params':{
            }
        },
        'SGDClassifier': {
            'model': SGDClassifier(),
            'params':{
                'loss' : ['hinge', 'perceptron', 'squared_error'],
                'learning_rate': ['optimal', 'invscaling', 'adaptive']
            }
        },
        'RandomForest': {
            'model': RandomForestClassifier(),
            'params': {
                'criterion': ['gini', 'entropy', 'log_loss'],
                'max_features': ['auto', 'sqrt', 'log2']
            }
        }
    }
    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2,random_state=0)
```

```
    for algo_name, config in algos.items():
        print(f'Working on {algo_name}......')
        gs = GridSearchCV(config['model'], config['params'], cv=cv, return_train_score
        gs.fit(x,y)
        scores.append({
            'model':algo_name,
            'best_score':gs.best_score_,
            'best_params': gs.best_params_
        })
        print(f'Best accuracy for the Algorithm is {gs.best_score_}')
    return pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'])
result = best_model(X,Y)
result
```

```
Working on logistic_regression......
Best accuracy for the Algorithm is 0.10499999999999998
Working on decision_tree......
Best accuracy for the Algorithm is 0.09
Working on Adaboost......
Best accuracy for the Algorithm is 0.1125
Working on GaussionNB......
Best accuracy for the Algorithm is 0.039999999999999994
Working on SGDClassifier......
Best accuracy for the Algorithm is 0.06250000000000001
Working on RandomForest......
Best accuracy for the Algorithm is 0.11500000000000002
```

Out[29]:

	model	best_score	best_params
0	logistic_regression	0.1050	{'fit_intercept': True}
1	decision_tree	0.0900	{'criterion': 'gini', 'max_features': 'log2'}
2	Adaboost	0.1125	{'learning_rate': 1.5, 'n_estimators': 100}
3	GaussionNB	0.0400	{}
4	SGDClassifier	0.0625	{'learning_rate': 'optimal', 'loss': 'hinge'}
5	RandomForest	0.1150	{'criterion': 'gini', 'max_features': 'log2'}

CONCLUSION¶

During our exploratory data analysis we could find that there was no significant amount of correlation of multiple data points with the pests. We tried to explore different ways to find the relationship between datapoints but the only relation we could establish was that until 1990s only few Pests were found and as we entered the 20th century the types of Pests increased.

The results thus indicate that the data available is insuffient to predict the type of Pests present in the rice plants. We need additional information such as quality of seeds used, type of fertilisers used, percentage of plant that is infected by the pests, the spread of the pests over different weeks to identify the acceleration of the growth of the pests. These additional information might help in identifying the type of pest that might be infected on the rice plants

In [31]:

```
# Now that we have all the columns in numerical datatype let us try to train the model
from sklearn.model_selection import train_test_split
```

```
from sklearn import metrics
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

Y = final_model_data['LF']
X = final_model_data.drop('LF', axis=1)
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.3)
```

In [32]:
```
import warnings
warnings.filterwarnings("ignore")

def best_model(x,y):
    algos = {
        'logistic_regression' : {
            'model': LogisticRegression(),
            'params':{'fit_intercept': [True, False]}
        },

#         'SVC' : {
#             'model': SVC(),
#             'params':{
#                 'kernel' : ['rbf', 'poly', 'sigmoid'],
#                 'degree': [2,3,4],
#                 'gamma': ['scale', 'auto', 0.22]
#             }
#         },
        'decision_tree' : {
            'model': DecisionTreeClassifier(),
            'params':{
                'criterion': ['gini', 'entropy', 'log_loss'],
                'max_features': ['auto', 'sqrt', 'log2']
            }
        },
        'Adaboost': {
            'model': AdaBoostClassifier(),
            'params':{
                'n_estimators': [50,100],
                'learning_rate': [1,1.5,2]
            }
        },
        'GaussionNB':{
            'model': GaussianNB(),
            'params':{
            }
        },
        'SGDClassifier': {
            'model': SGDClassifier(),
            'params':{
                'loss' : ['hinge', 'perceptron', 'squared_error'],
                'learning_rate': ['optimal', 'invscaling', 'adaptive']
            }
        },
        'RandomForest': {
            'model': RandomForestClassifier(),
            'params': {
                'criterion': ['gini', 'entropy', 'log_loss'],
                'max_features': ['auto', 'sqrt', 'log2']
            }
```

```
        }
    }
    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2,random_state=0)
    for algo_name, config in algos.items():
        print(f'Working on {algo_name}......')
        gs = GridSearchCV(config['model'], config['params'], cv=cv, return_train_score
        gs.fit(x,y)
        scores.append({
            'model':algo_name,
            'best_score':gs.best_score_,
            'best_params': gs.best_params_
        })
        print(f'Best accuracy for the Algorithm is {gs.best_score_}')
    return pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'])
result = best_model(X,Y)
result
```

```
Working on logistic_regression......
Best accuracy for the Algorithm is 0.2925
Working on decision_tree......
Best accuracy for the Algorithm is 0.24
Working on Adaboost......
Best accuracy for the Algorithm is 0.3075
Working on GaussionNB......
Best accuracy for the Algorithm is 0.085
Working on SGDClassifier......
Best accuracy for the Algorithm is 0.1475
Working on RandomForest......
Best accuracy for the Algorithm is 0.33999999999999997
```

Out[32]:

	model	best_score	best_params
0	logistic_regression	0.2925	{'fit_intercept': False}
1	decision_tree	0.2400	{'criterion': 'gini', 'max_features': 'log2'}
2	Adaboost	0.3075	{'learning_rate': 1, 'n_estimators': 50}
3	GaussionNB	0.0850	{}
4	SGDClassifier	0.1475	{'learning_rate': 'optimal', 'loss': 'perceptr...
5	RandomForest	0.3400	{'criterion': 'entropy', 'max_features': 'log2'}

In [ ]: