Análise das Características do Solo ideiais para cada Cultura

Projeto da disciplina SME0860 - Aprendizado de Máquina Aplicado a Problemas

Membros

- Alexandre E. de Souza Jesus - alexandre_souza@usp.br - **12559506**
- Eduardo Zaffari Monteiro - eduardozaffarimonteiro@usp.br - **12559490**
- Lucas Ivars Cadima Ciziks - luciziks@usp.br - **12559472**

1. Introdução

O uso do solo de maneira adequada é de fundamental importância para maximizar o retorno do plantio e mantê-lo em boas condições para que possa ser utilizado por vários anos sem degradação. Caso o solo seja mal manejado, pode-se acabar com um terreno infértil, o que aumenta a demanda de recursos para o cultivo e recuperação da área de plantio.

Conforme o solo é utilizado para o plantio de um tipo de alimento, ocorre a diminuição dos tipos de nutrientes consumidos por essa planta, e em contrapartida os outros se tornam abundantes pelo acúmulo durante o tempo em que não foi consumido. Dessa forma, pode-se realizar a rotação de culturas, que consiste em plantar alimentos que consomem nutrientes diferentes de maneira cíclica, fazendo com que o solo se mantenha mais bem preservado.

Com objetivo de facilitar a escolha da cultura a ser semeada em um terreno específico será feito o treinamento de um modelo de aprendizado supervisionado. Para isso, serão utilizados dados que consistem em características do terreno, principalmente relacionadas à quantidade de nutrientes e substâncias nele presentes. A classificação dar-se-á de acordo com um tipo de cultura que é considerado ideal para o solo observado.

A tabela de dados apresenta 2200 diferentes condições de solo e suas respectivas culturas ideais, as quais são divididas em 22 classes que indicam diferentes sementes e frutas. Para cada um desses terrenos existem valores de quantidade de nitrogênio, fósforo, potássio, além de temperatura, umidade e pH, além da precipitação plantação recebe durante o crescimento.

2. Metodologia

O projeto está sendo realizado e versionado remotamente através da plataforma Github. Seu acesso é possível por meio do link https://github.com/ale-souza/crop-recommendation

2.1. Origem dos Dados

Os dados foram obtidos diretamente da plataforma kaggle, um site para estudo de ciência de dados e machine learning, e podem ser obtidos através do link https://www.kaggle.com/datasets/aksahaha/crop-recommendation. Segundo o usuário Abhishek Kumar, que disponibilizou os dados, eles são provenientes do ICAR (Indian Council of Agriculture Research), e complementados por pesquisas na internet feitas por ele.

2.2 Dicionário de Dados

- **Nitrogênio (nitrogen)**: Representa a quantidade de nitrogênio (em kg/ha) presente no solo para a cultura. O nitrogênio é um nutriente essencial para o crescimento de plantas, e sua deficiência ou excesso pode afetar o crescimento e a produção da cultura;
- **Fósforo (phosphorus)**: Representa a quantidade de fósforo (em kg/ha) presente no solo para a cultura. Também é um elemento essencial no plantio, sendo importante para processos como transferência de energia e fotossíntese;
- **Potássio (potassium)**: Representa a quantidade de potássio (em kg/ha) presente no solo para a cultura. Também é um elemento essencial, e é importante para processos fisiológicos como regulação de água e transporte de nutrientes;
- **Temperatura (temperature)**: Representa a temperatura média (em Celsius) durante o período de crescimento da cultura. A temperatura é um fator ambiental importante que pode afetar o crescimento e o desenvolvimento das plantas, e cada cultura possui uma temperatura ideal;
- **Umidade (humidity)**: Representa a humidade relativa (em porcentagem) durante o período de crescimento da cultura. A humidade é outro fator ambiental importante, tendo em vista que uma alta humidade pode promover a proliferação de fungos e desenvolvimento de doenças;
- **pH**: Representa o pH da cultura durante seu período de crescimento. O pH é uma medida de acidez ou alcalinidade do solo e pode afetar a disponibilidade de nutrientes para a cultura;
- **Precipitação (rainfall)**: Representa a precipitação (em mm) durante o período de crescimento da cultura. Cada cultura necessita de uma quantidade diferente de água, o que torna a precipitação outro fator ambiental importante;
- **Crop (label)**: Representa o tipo da cultura.

3. Coleta dos Dados

In [44]:
```
# Importando bibliotecas
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

# Modelos de aprendizado supervisionado
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.svm import SVC

# Normalização dos dados
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Validação e Particionamento dos dados
from sklearn.model_selection import StratifiedKFold, LeaveOneOut, GridSearchCV, Stratifi

# Métricas de Avaliação
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score
```

In [6]:
```
# Leitura dos dados do problema
df = pd.read_csv("https://raw.githubusercontent.com/ale-souza/crop-recommendation/main/C

df.head()
```

Out[6]:

	Nitrogen	phosphorus	potassium	temperature	humidity	ph	rainfall	label	Unnamed: 8	Unnamed: 9
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice	NaN	NaN
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice	NaN	NaN
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice	NaN	NaN
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice	NaN	NaN
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice	NaN	NaN

In [7]:
```
# Limpando conjunto de dados
df = df.drop(['Unnamed: 8', 'Unnamed: 9'], axis=1)
df = df.rename(columns={"label": "crop"})
df.head()
```

Out[7]:

	Nitrogen	phosphorus	potassium	temperature	humidity	ph	rainfall	crop
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice

In [8]:
```
# Verificando as categorias de plantação
labels = df['crop'].astype('category').values
labels = list(labels.categories)
labels
```

Out[8]:
```
['apple',
 'banana',
 'blackgram',
 'chickpea',
 'coconut',
 'coffee',
 'cotton',
 'grapes',
 'jute',
 'kidneybeans',
 'lentil',
 'maize',
 'mango',
```

```
 'mothbeans',
 'mungbean',
 'muskmelon',
 'orange',
 'papaya',
 'pigeonpeas',
 'pomegranate',
 'rice',
 'watermelon']
```

In [57]:
```
# Discretizando categorias de plantação
df["crop_int"] = pd.Categorical(df["crop"]).codes
df["crop_int"]
```

Out[57]:
```
0       20
1       20
2       20
3       20
4       20
        ..
2195     5
2196     5
2197     5
2198     5
2199     5
Name: crop_int, Length: 2200, dtype: int8
```

4. Análise Exploratória dos Dados

4.1. Medidas Descritivas

In [ ]:
```
# Função para calcular o coeficiente de variância (CV)
def coeficiente_variancia(table):
    return 100 * table.std() / table.mean()

# Função para calcular a amplitude
def amplitude(table):
    return table.max() - table.min()

# Aplicando medidas de posição e dispersão aos atributos preditivos
medidas_descritivas = df.drop(["crop"], axis=1).agg(["min", "max", "mean", "median",
                                                     "var", "std",
                                                     coeficiente_variancia, amplitude])
```

In [ ]:
```
# Renomeando das medidas descritivas
novos_nomes = {
    "min":"Minimo",
    "max": "Maximo",
    "mean":"Media",
    "median": "Mediana",
    "var": "Variancia",
    "std": "Desvio-padrao",
    "coeficiente_variancia":"Coeficiente de Variancia",
    "amplitude": "Amplitude"
}

medidas_descritivas = medidas_descritivas.rename(novos_nomes)

# Arredondando casas decimais das medidas descritivas e de dispersão
medidas_descritivas = medidas_descritivas.round(3)
```

In [ ]:
```
medidas_descritivas
```

Out[ ]:

	Nitrogen	phosphorus	potassium	temperature	humidity	ph	rainfall
Minimo	0.000	5.000	5.000	8.826	14.258	3.505	20.211
Maximo	140.000	145.000	205.000	43.675	99.982	9.935	298.560
Media	50.552	53.363	48.149	25.616	71.482	6.469	103.464
Mediana	37.000	51.000	32.000	25.599	80.473	6.425	94.868
Variancia	1362.890	1088.068	2565.213	25.642	495.677	0.599	3020.424
Desvio-padrao	36.917	32.986	50.648	5.064	22.264	0.774	54.958
Coeficiente de Variancia	73.029	61.814	105.190	19.768	31.146	11.963	53.119
Amplitude	140.000	140.000	200.000	34.850	85.724	6.430	278.349

A partir das medidas descritivas, pode-se ter uma ideia inicial das distribuições das características. É possível inferir que as variáveis *temperature* e *pH* possuem uma curva simétrica, já que suas médias e medianas são bem próximas, enquanto *humidity* provavelmente possui uma curva assimétrica à esquerda. Para todos os outros atributos as curvas são possivelmente assimétricas à direita.

No que tange as medidas de dispersão, a análise anterior é reforçada. As medidas de variância e desvio-padrão apresentam valores altos para as variáveis que não são simétricas, o que indica que há uma alta variabilidade nos dados. Ou seja, há valores que possuem uma grande distância da média.

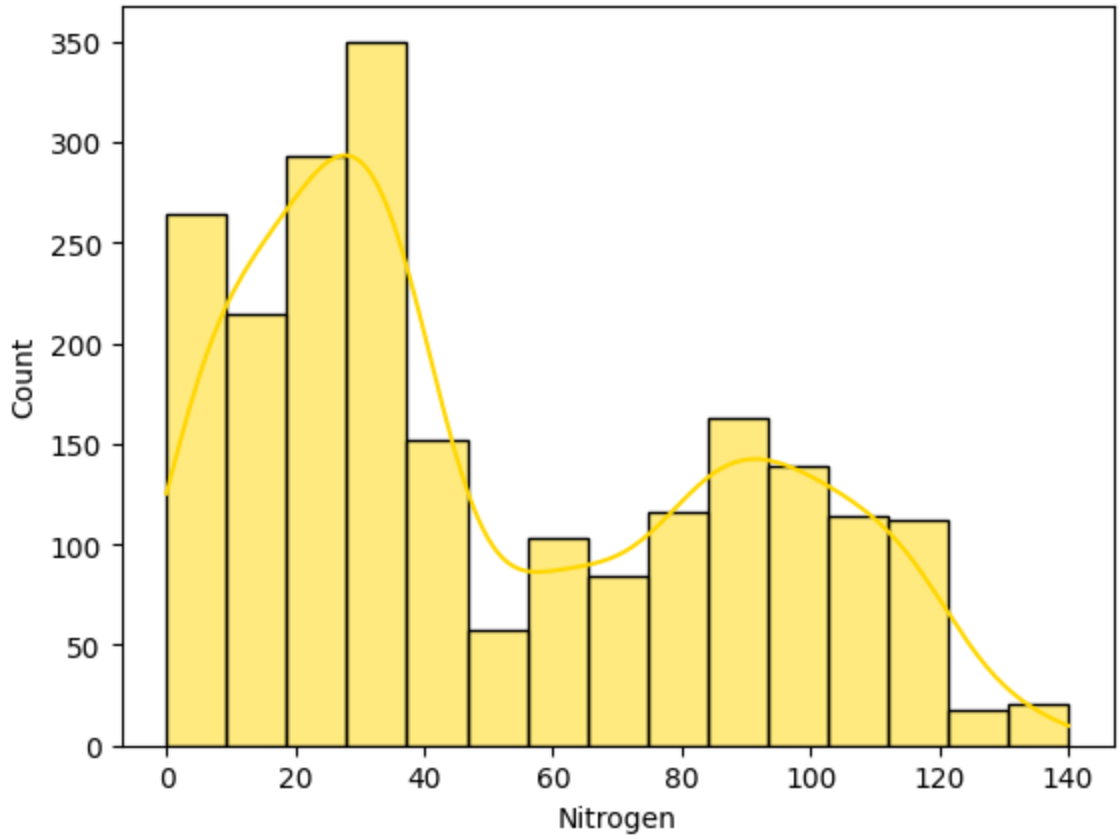
4.2. Visualização dos Dados

4.2.1. Histogramas

Abaixo, é possível observar o formato da curva dos atributos. Assim, há ainda mais evidências de que a análise anteriormente feita está, provavelmente, correta. Fazem-se necessários, então, testes de hipóteses.

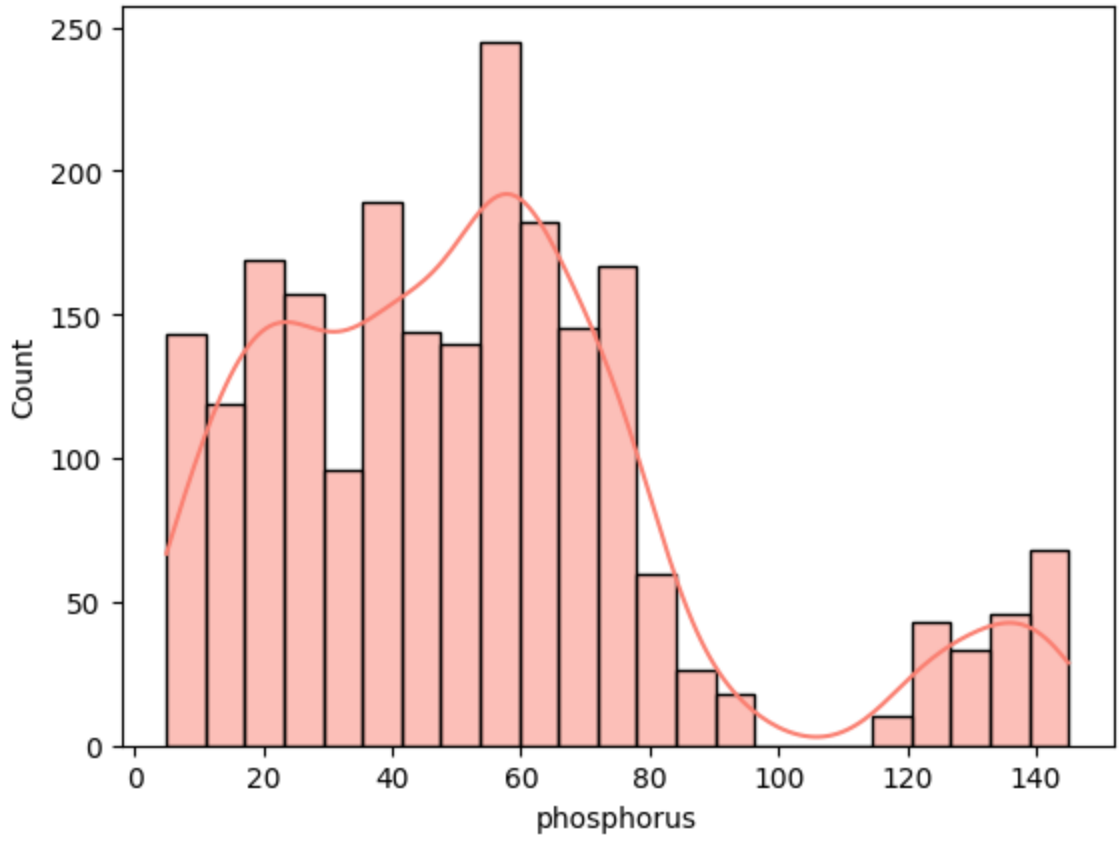
In [ ]:
```
# Nitrogen
sns.histplot(data=df, x="Nitrogen", kde=True, color="gold")
```

Out[ ]:
```
<Axes: xlabel='Nitrogen', ylabel='Count'>
```



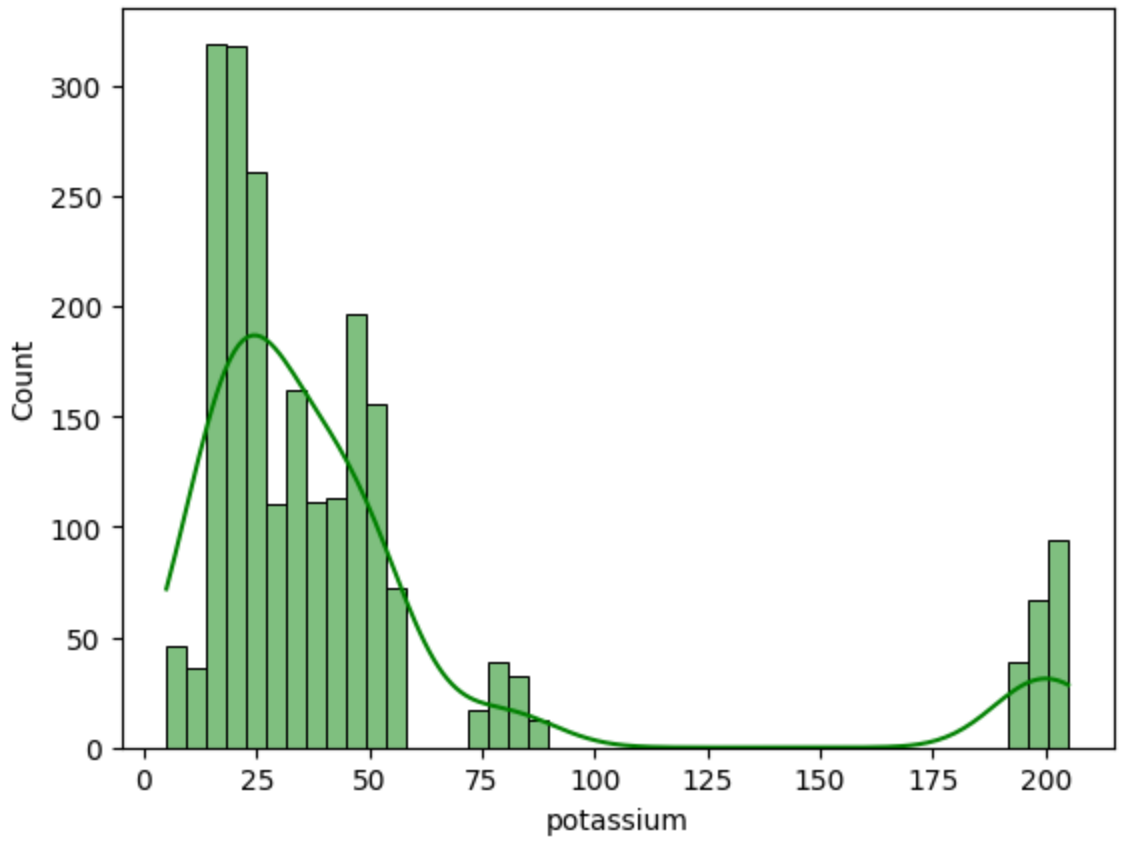
In [ ]:
```
# Phosphorus
sns.histplot(data=df, x="phosphorus", kde=True, color="salmon")
```

Out[ ]:
```
<Axes: xlabel='phosphorus', ylabel='Count'>
```



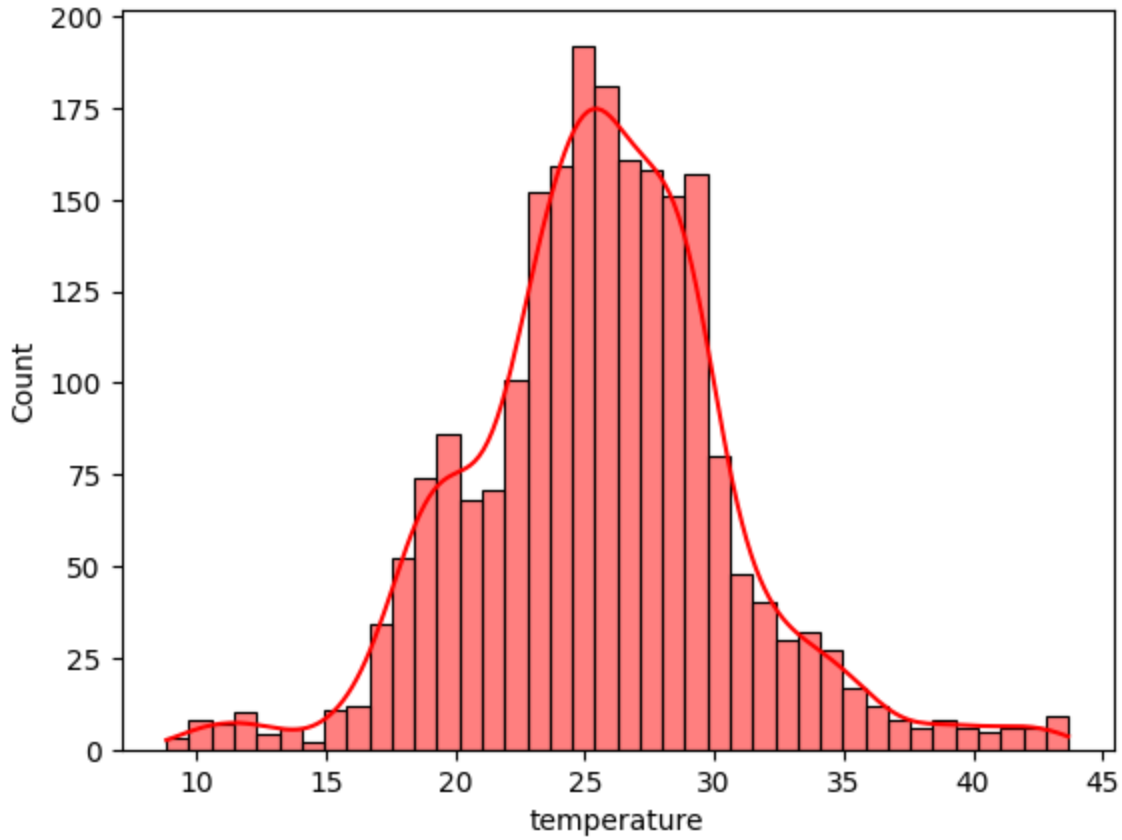
In [ ]:
```
# Potassium
sns.histplot(data=df, x="potassium", kde=True, color="green")
```

Out[ ]:
```
<Axes: xlabel='potassium', ylabel='Count'>
```



In [ ]:
```
# temperature
sns.histplot(data=df, x="temperature", kde=True, color="red")
```

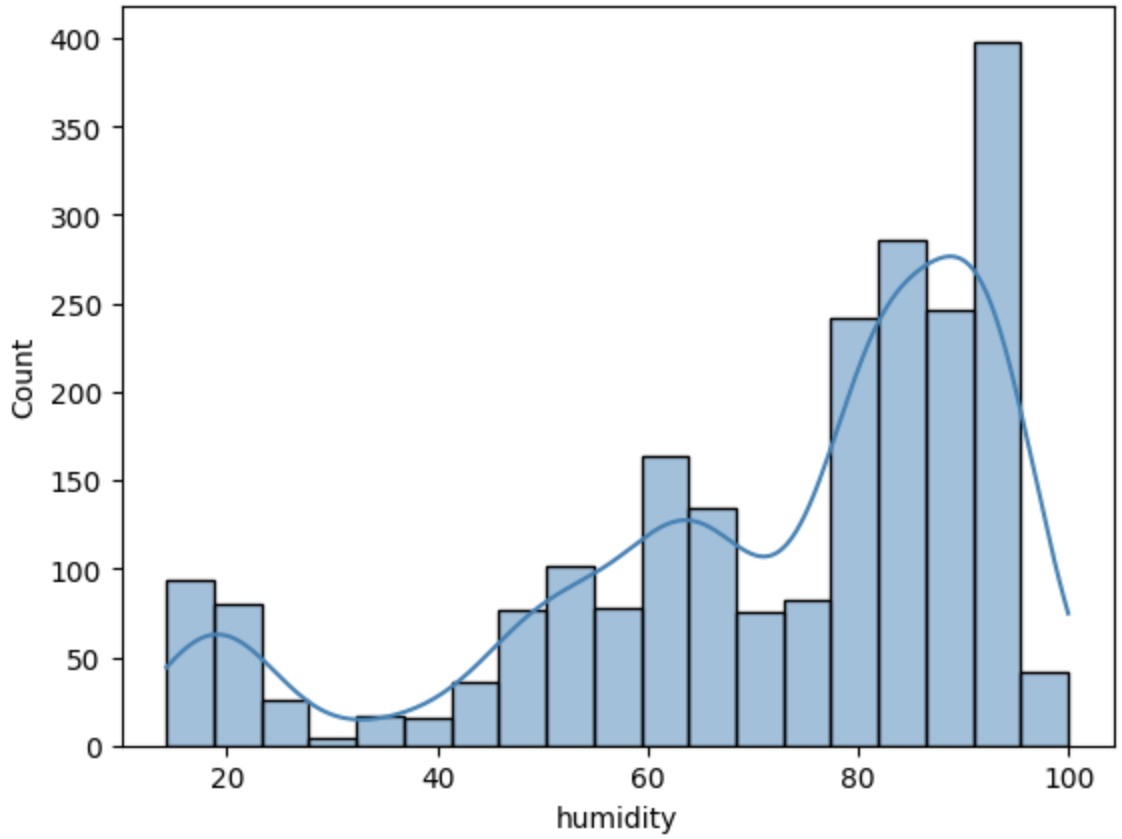
Out[ ]:
```
<Axes: xlabel='temperature', ylabel='Count'>
```



Percebe-se, como anteriormente dito, que a distribuição dos dados referentes à variável temperatura provavelmente segue uma distribuição.

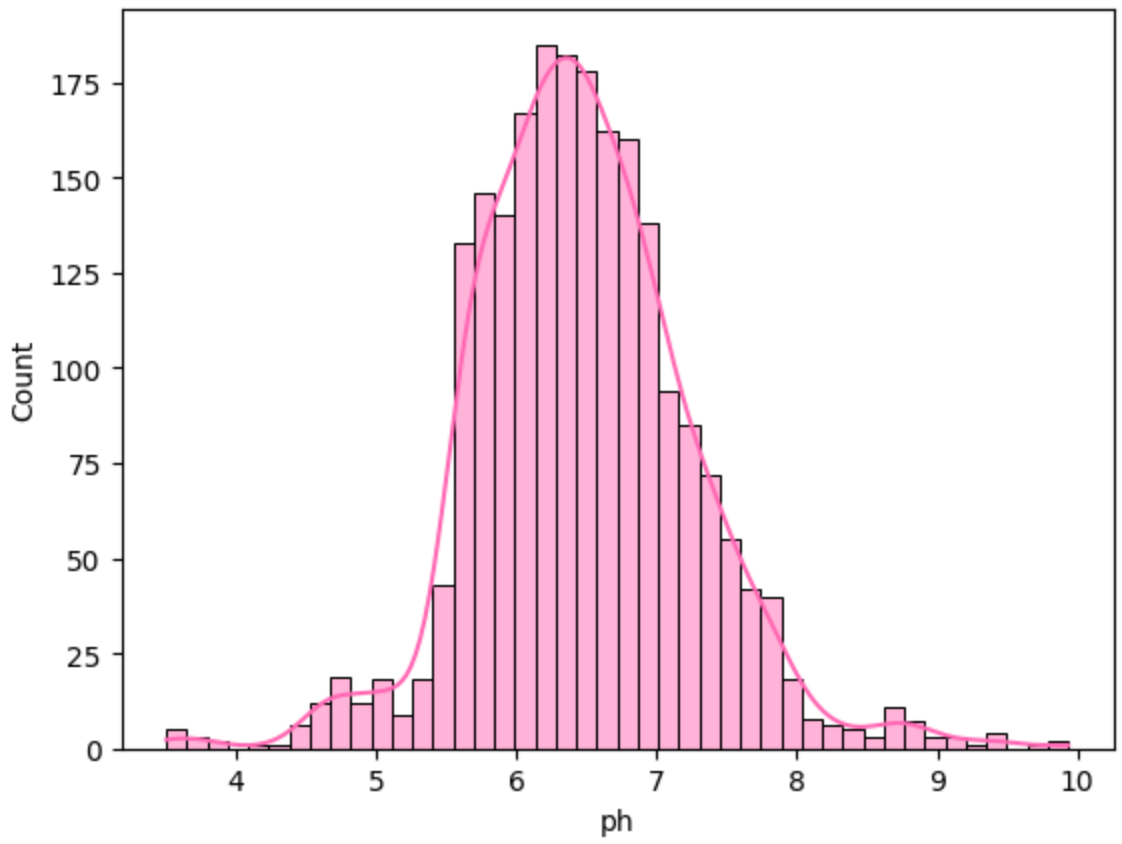
In [ ]:
```
# Humidity
sns.histplot(data=df, x="humidity", kde=True, color="steelblue")
```

Out[ ]:
```
<Axes: xlabel='humidity', ylabel='Count'>
```



In [ ]:
```
# ph
sns.histplot(data=df, x="ph", kde=True, color="hotpink")
```

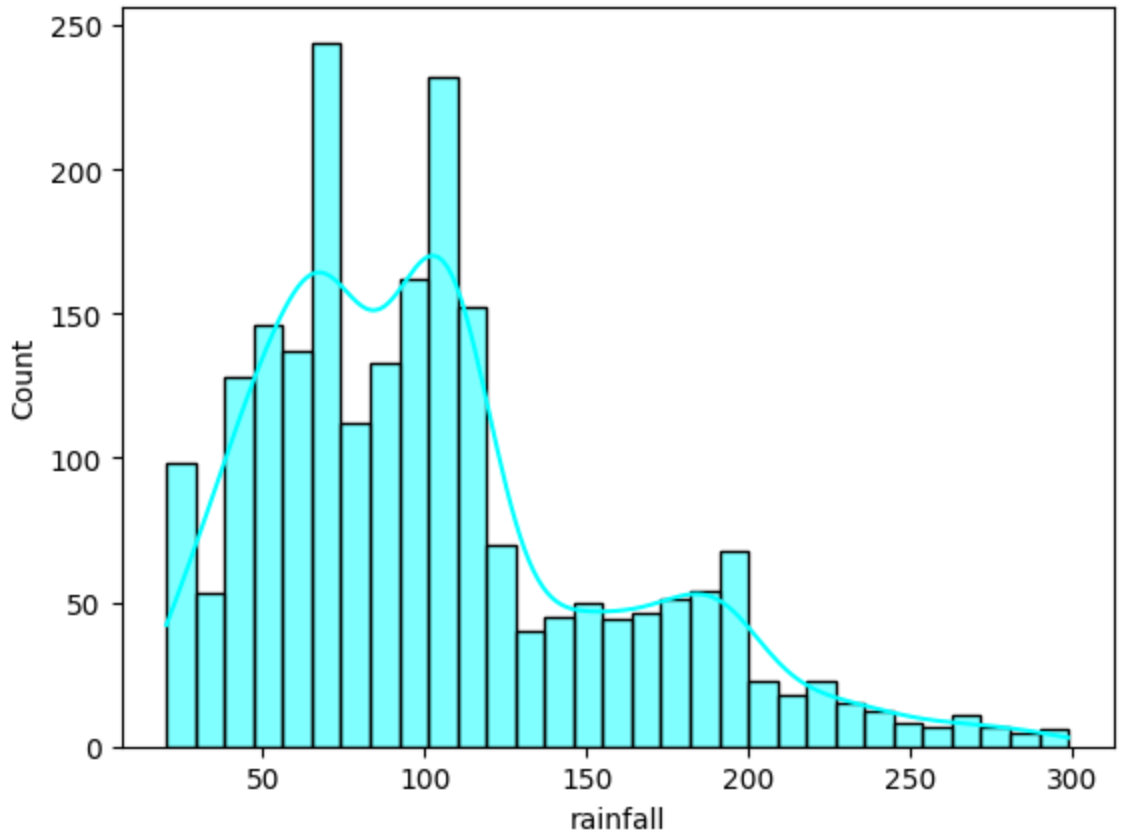
Out[ ]:
```
<Axes: xlabel='ph', ylabel='Count'>
```



Percebe-se, como anteriormente dito, que a distribuição dos dados referentes à variável pH provavelmente segue uma distribuição.

In [ ]:
```
## Rainfall
sns.histplot(data=df, x="rainfall", kde=True, color="aqua")
```

Out[ ]:
```
<Axes: xlabel='rainfall', ylabel='Count'>
```



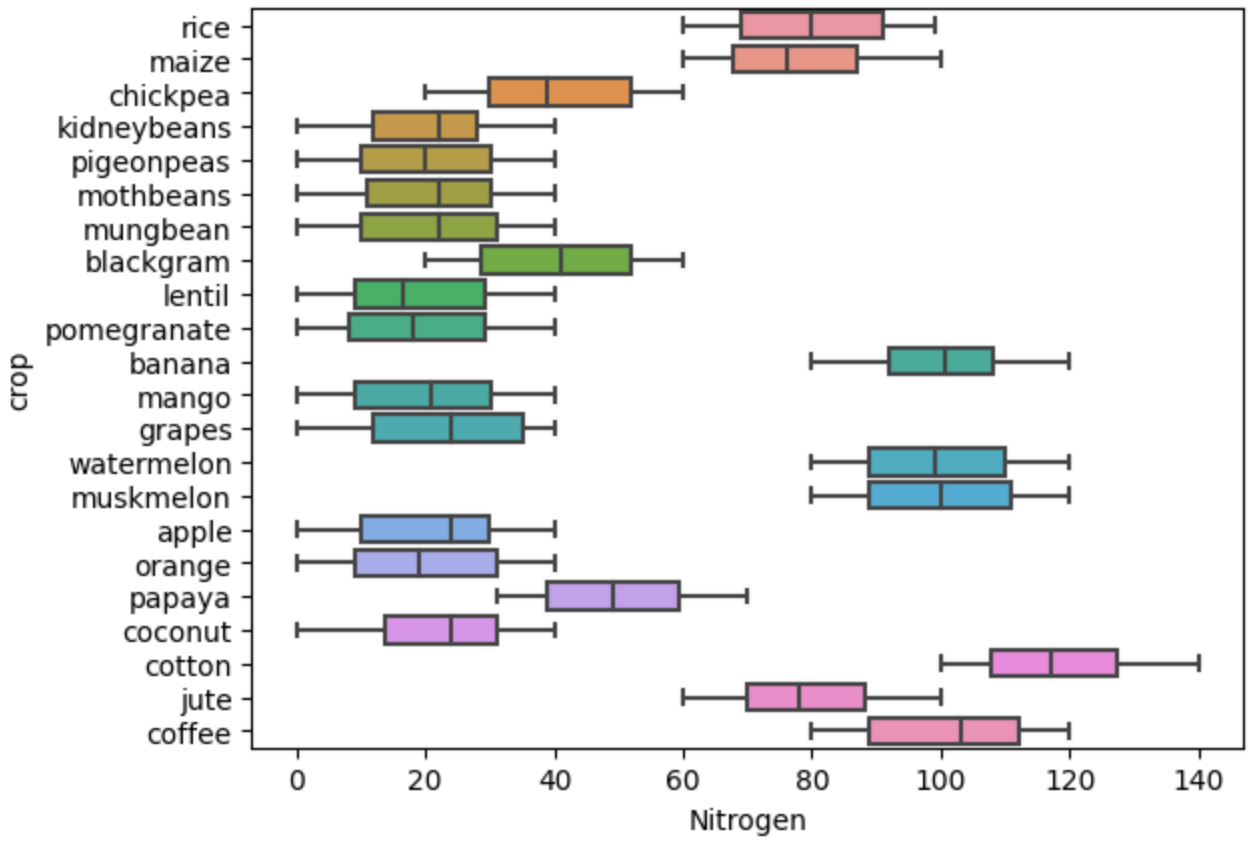
4.2.2. Boxplots

Com os *boxplots*, é possível comparar a distribuição dos dados em relação ao atributo-alvo. Mais uma vez, a teoria de que as variáveis *pH* e *temperature* são mais balanceadas é corroborada.

In [ ]:
```
# --------------------
#   Nitrogen x Crop
# --------------------

sns.boxplot(x=df["Nitrogen"], y=df["crop"])
```

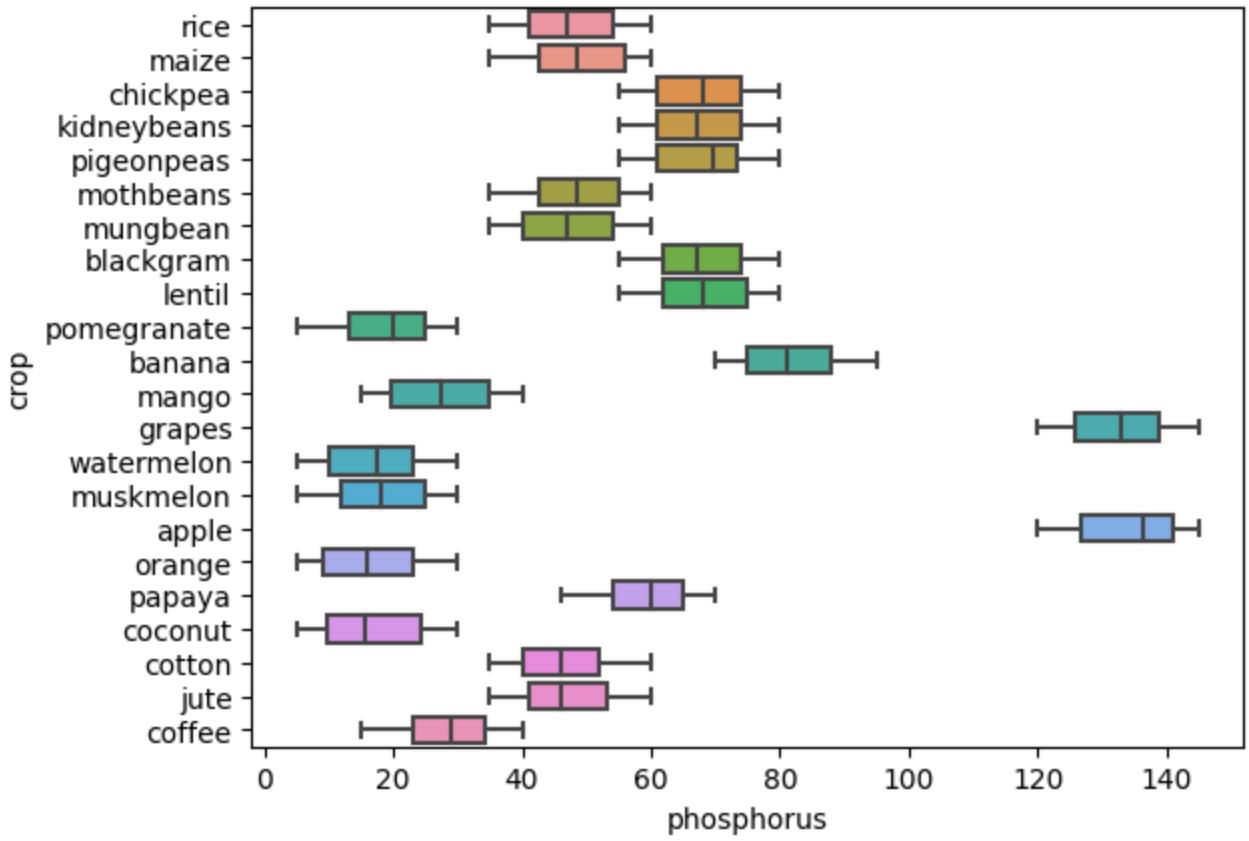
Out[ ]:
```
<Axes: xlabel='Nitrogen', ylabel='crop'>
```



In [ ]:
```
# ----------------------
#   Phosphorus x Crop
# ----------------------

sns.boxplot(x=df["phosphorus"], y=df["crop"])
```

Out[ ]:
```
<Axes: xlabel='phosphorus', ylabel='crop'>
```

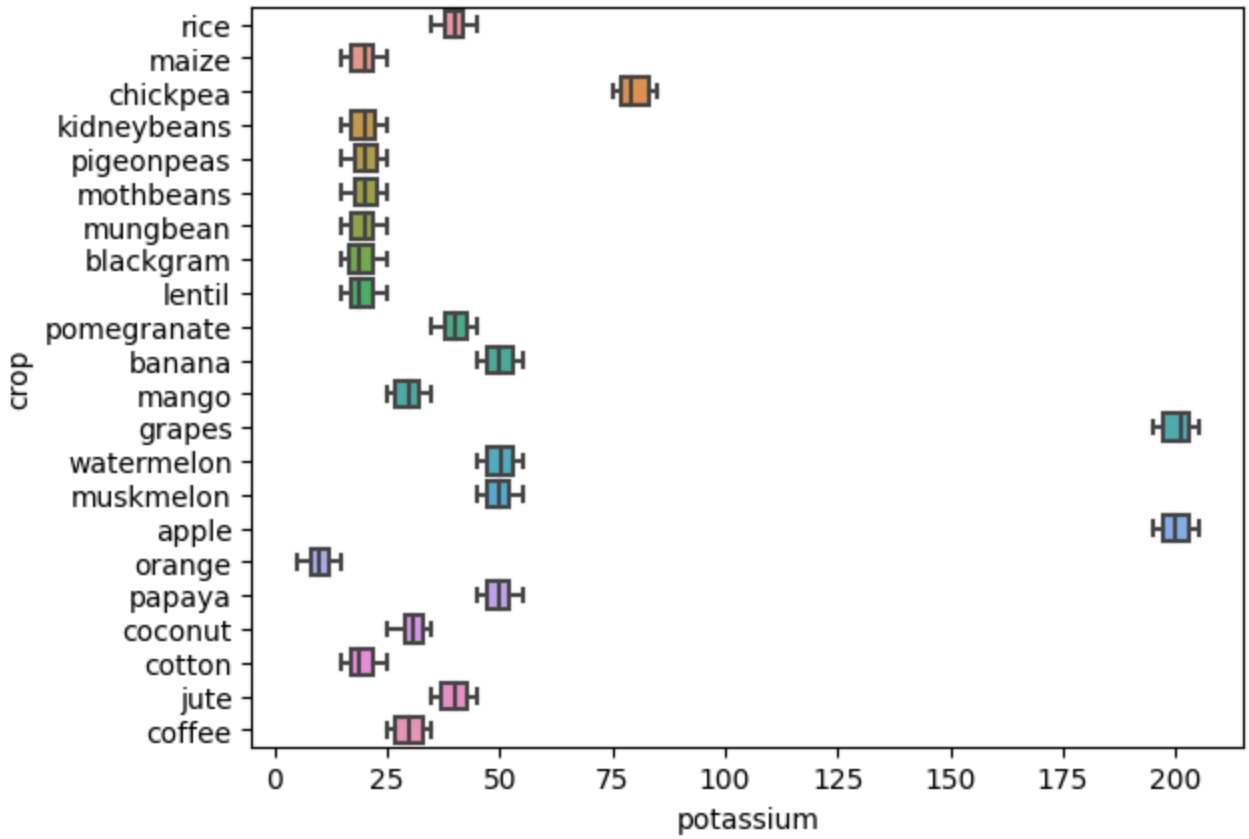


In [ ]:
```
# --------------------
```

```
#   Potassium x Crop
# ---------------------

sns.boxplot(x=df["potassium"], y=df["crop"])
```

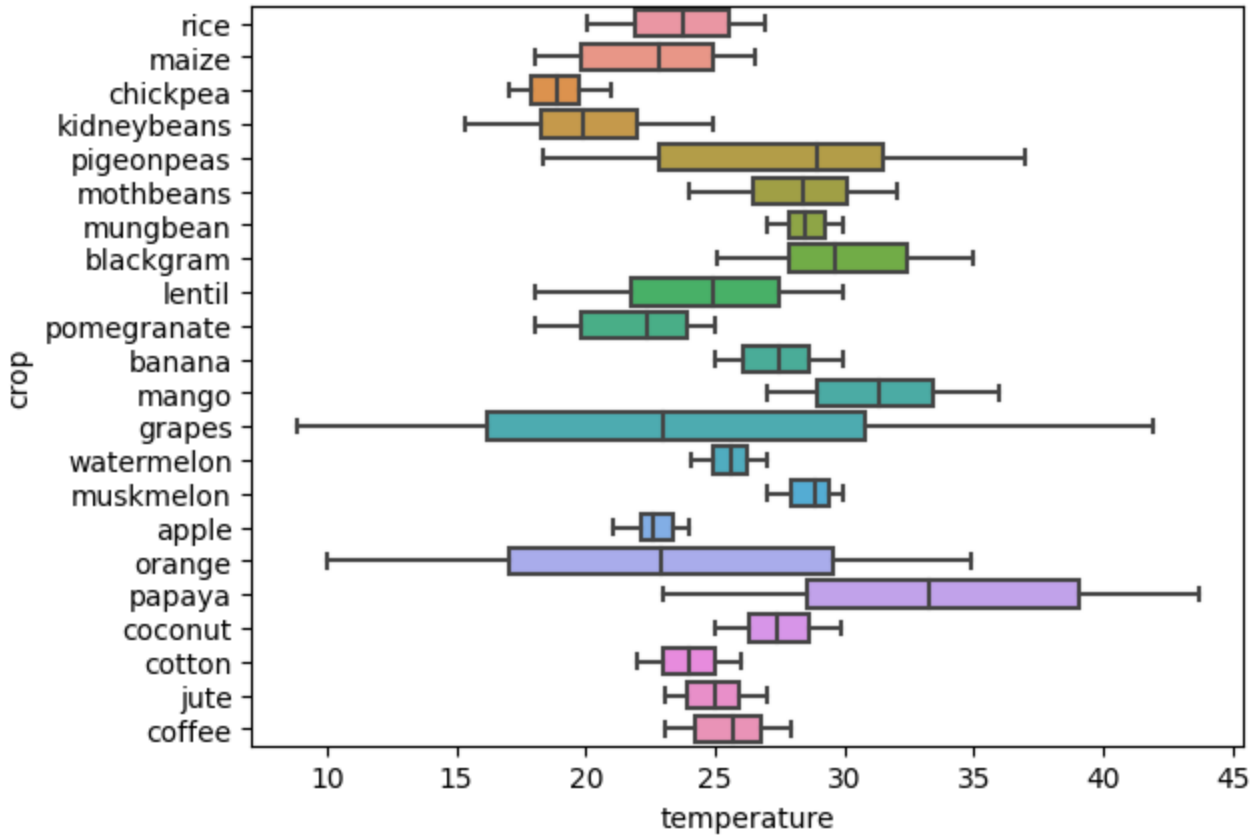
Out[ ]:
```
<Axes: xlabel='potassium', ylabel='crop'>
```



In [ ]:
```
# ----------------------
#   Temperature x Crop
# ----------------------

sns.boxplot(x=df["temperature"], y=df["crop"])
```

Out[ ]:
```
<Axes: xlabel='temperature', ylabel='crop'>
```

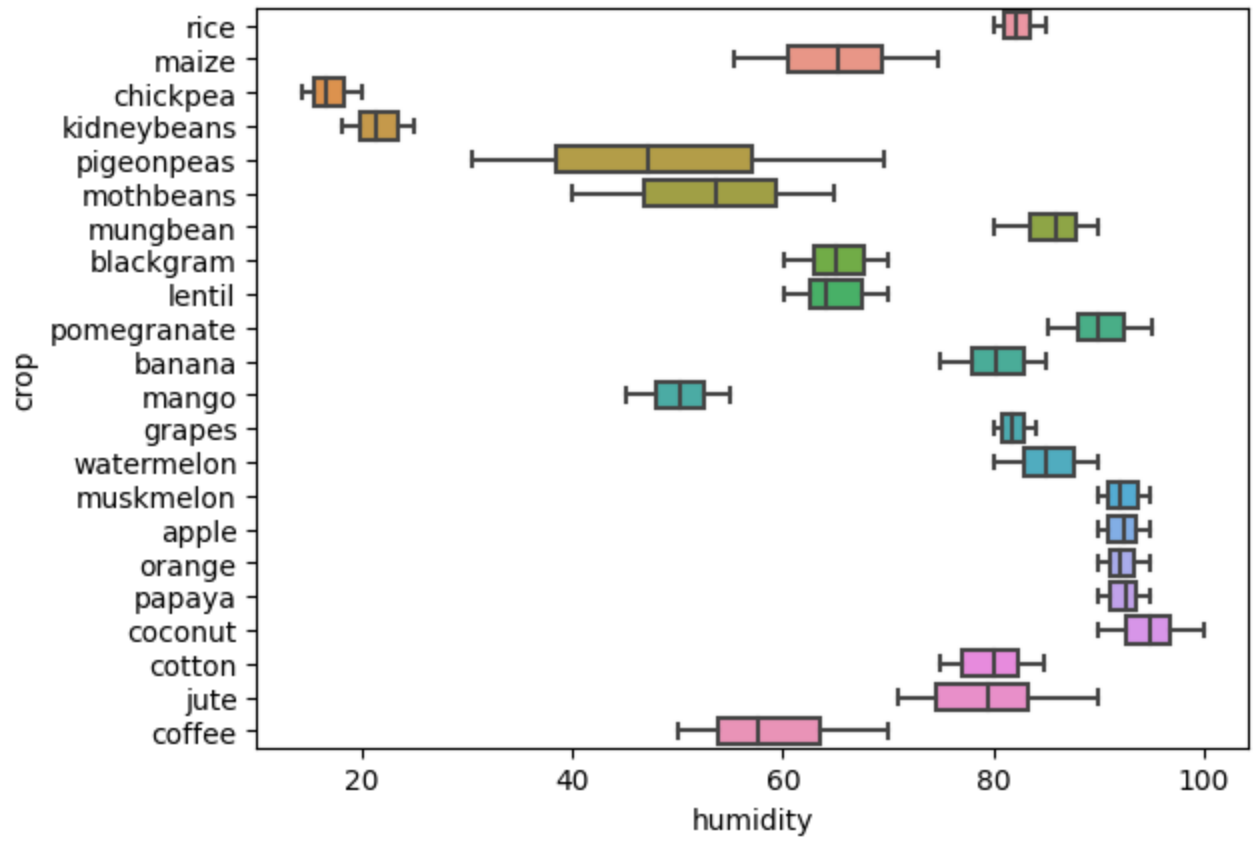


A maior parte dos valores está ao redor da média, que é de aproximadamente 25. Apesar de certos valores apresentarem grande variação, como *grapes* e *orange*, isso não afetou a curva.

In [ ]:
```
# --------------------
#   Humidity x Crop
# --------------------

sns.boxplot(x=df["humidity"], y=df["crop"])
```

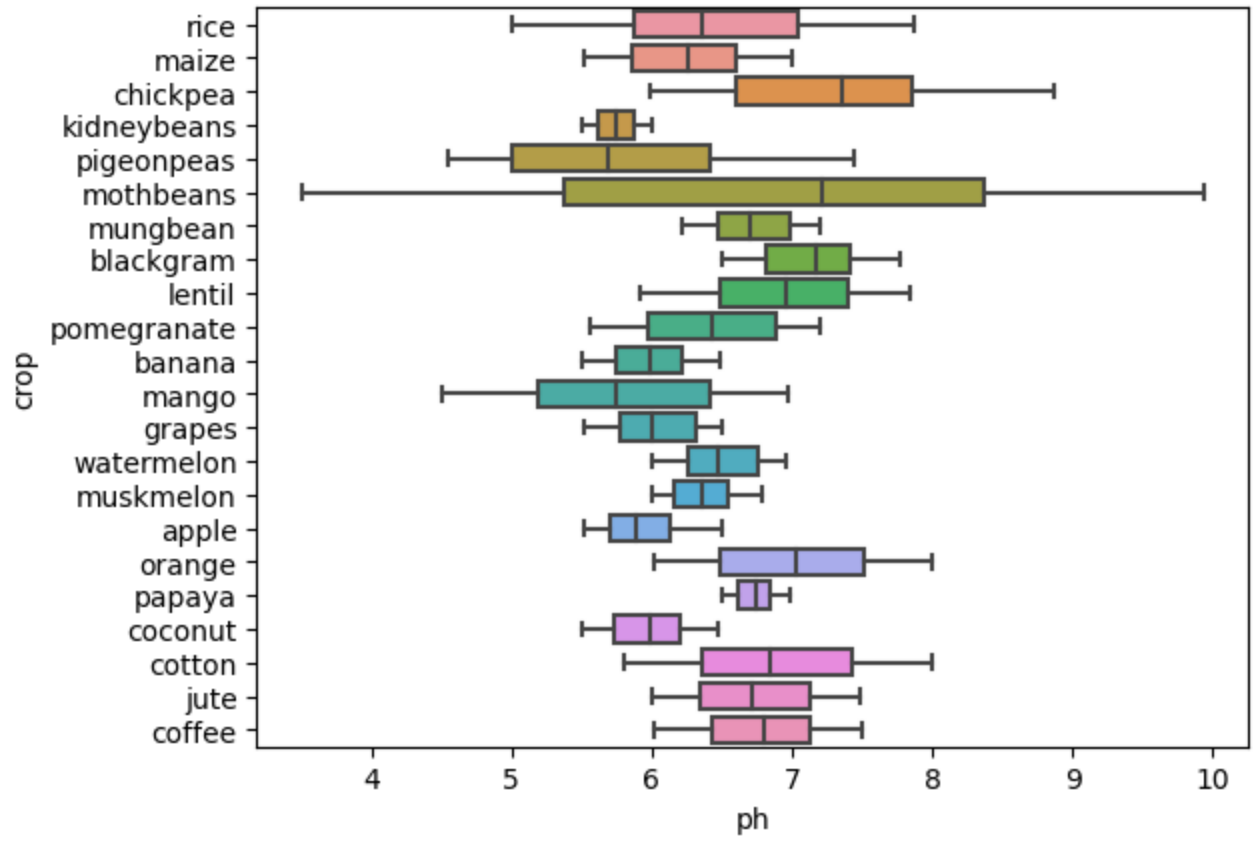
Out[ ]:
```
<Axes: xlabel='humidity', ylabel='crop'>
```



In [ ]:
```
# --------------
#   ph x Crop
# --------------

sns.boxplot(x=df["ph"], y=df["crop"])
```

Out[ ]:
```
<Axes: xlabel='ph', ylabel='crop'>
```



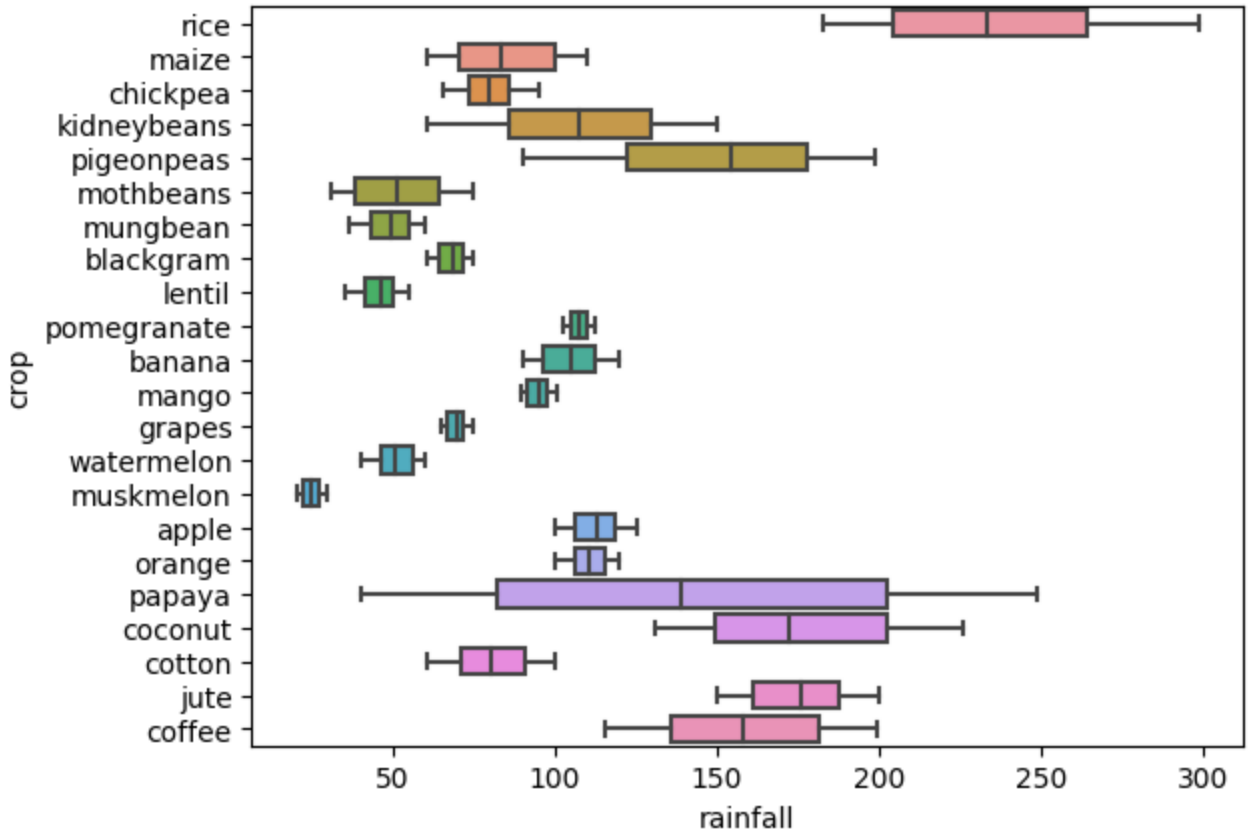
O padrão se repete com o atributo *pH*. A maior parte dos valores está concentrada ao redor da média.

Neste caso, tal resultado é previsível, visto que essa medida varia entre 0 e 14, e 7 representa um meio neutro (a média dos valores foi de aproximadamente 6,4)

In [ ]:
```
# --------------------
#   Rainfall x Crop
# --------------------

sns.boxplot(x=df["rainfall"], y=df["crop"])
```

Out[ ]:
```
<Axes: xlabel='rainfall', ylabel='crop'>
```

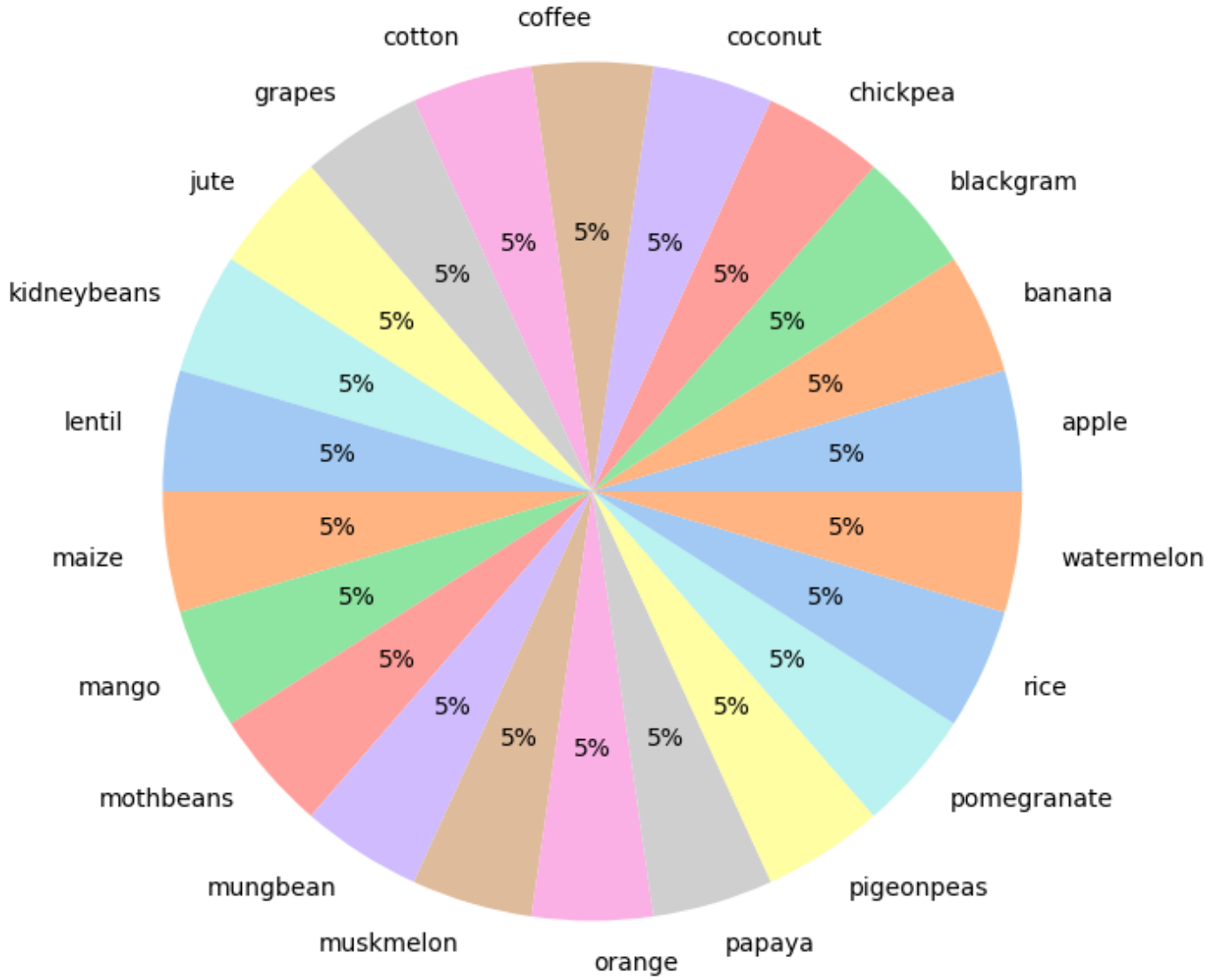


É possível perceber, então, que há certa separação no que tange aos atributos para cada tipo de cultura. Há indícios, portanto, de que é possível classificar o rótulo de novas observações a partir deste conjunto de variáveis.

4.2.3. Balaceamento dos Dados

In [ ]:
```
plt.figure(figsize=(8, 8))
colors = sns.color_palette('pastel')

plt.pie(df['crop'].groupby(df['crop']).count(), labels = labels, colors = colors, autopc
plt.show()
```

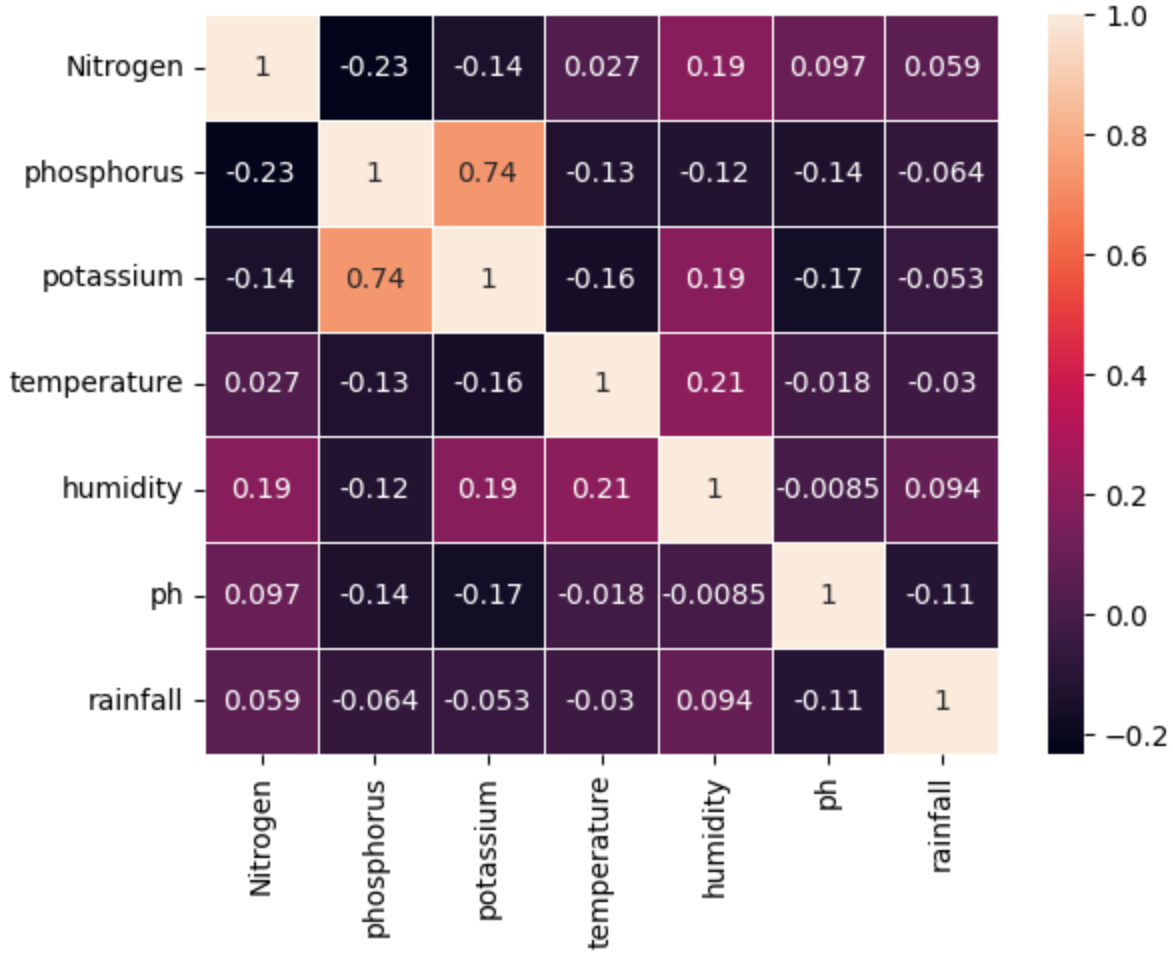


É possível perceber que a distribuição dos valores é idêntica.

4.3. Correlação entre as Variáveis

In [ ]:
```
# Mapa de calor com as correlações entre os atributos
sns.heatmap(df.drop(columns=["crop"]).corr(), annot=True, linewidths=0.5)
```

Out[ ]:
```
<Axes: >
```

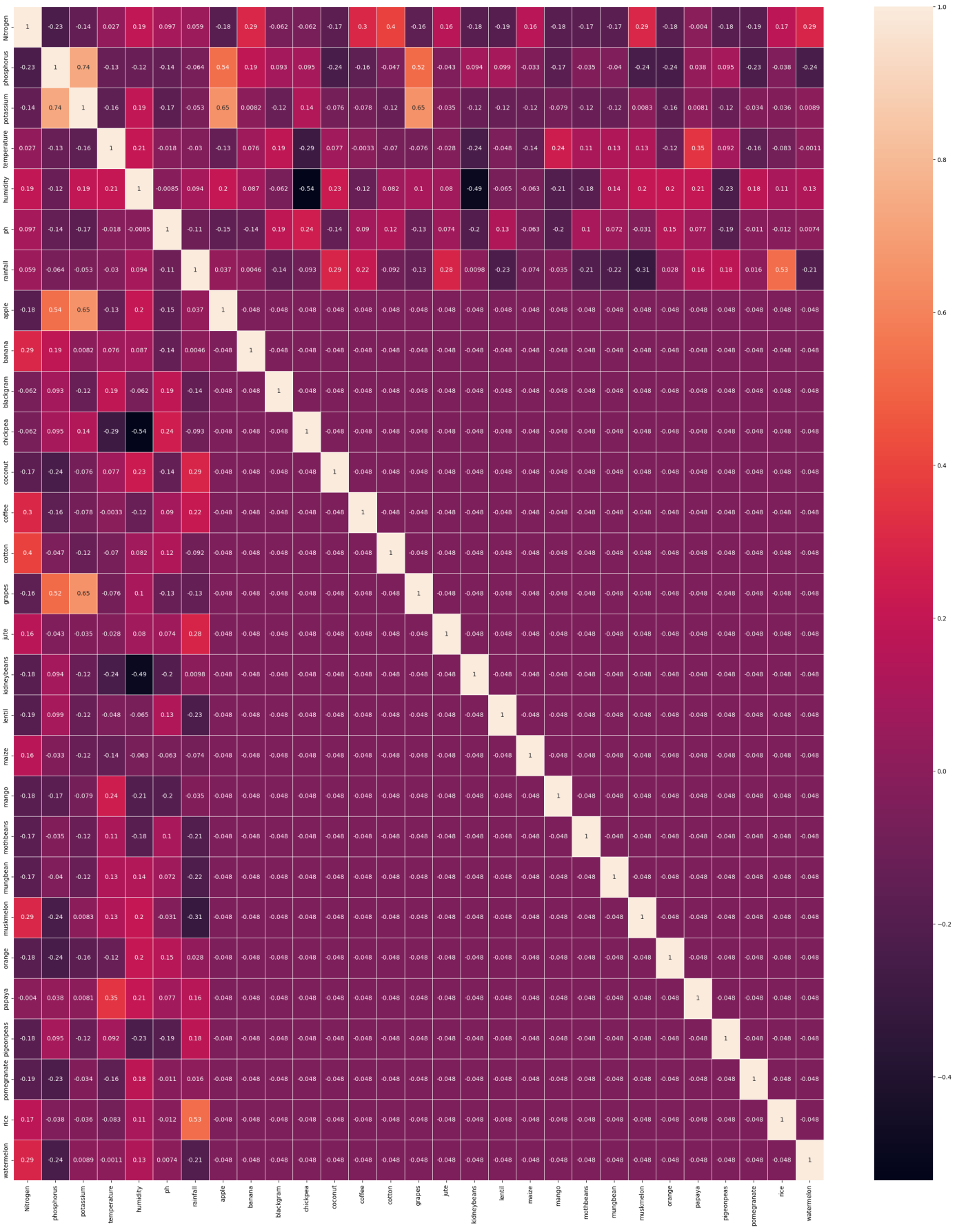


A maior parte das variáveis **não** está relacionada entre si, com exceção dos atributos *potassium* e *phosphorus*, que possuem uma correlação positiva considerável. Para que a classificação dos dados seja mais efetiva, um dos atributos pode ser removido. Abaixo é possível visualizar a **matriz de dispersão** entre todos as culturas possíveis (os rótulos) e os atributos, o que é necessário para avaliar qual das duas variáveis relacionadas poderia ser removida.

In [ ]:
```
dummies = pd.get_dummies(df['crop'])
dummy_df = pd.concat([df, dummies], axis=1)

plt.figure(figsize=(30, 35))
corr = dummy_df.drop(columns=["crop"]).corr()
sns.heatmap(corr, annot=True, linewidths=0.5)
```

Out[ ]:
```
<Axes: >
```

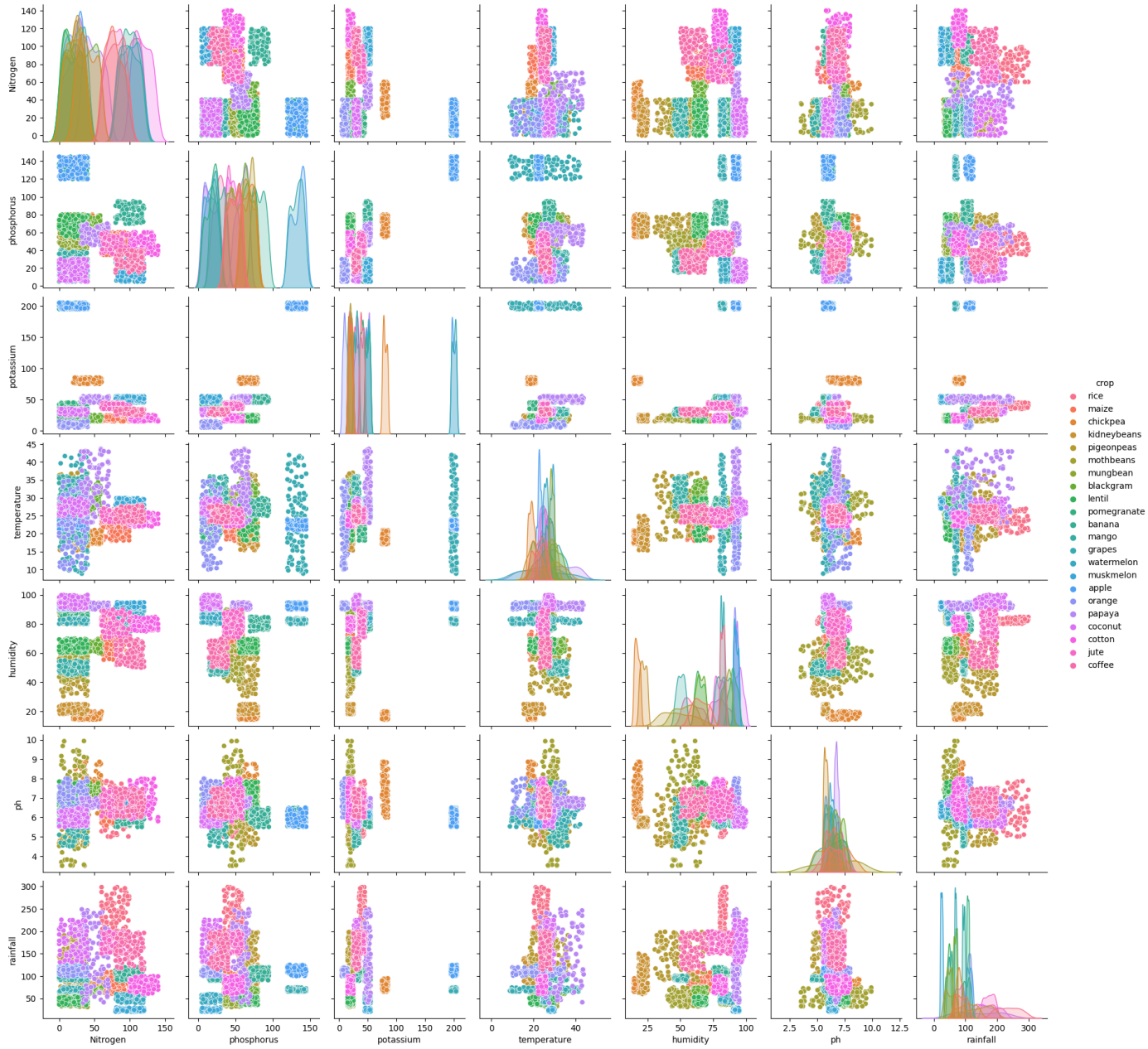


É possível perceber que dentre elas a que possui maior relação com os atributos-alvo é a *potassium*. Sendo assim, uma possível saída para o problema seria remover a variável *phosporus* do conjunto de dados, visto que *potassium* tem maior relação com o atributo-alvo.

In [ ]:
```
# Interação entre os atributos preditivos
sns.pairplot(df, hue='crop')
```

```
<seaborn.axisgrid.PairGrid at 0x7f2d1a4967d0>
```

Out[ ]:



5. Modelos de Aprendizado Supervisionado (Classificação)

In [58]:
```
X = df.drop(["crop", "crop_int"], axis=1)
y = df["crop_int"]
```

In [59]:
```
# Separando em conjunto de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .2, random_state =
```

In [60]:
```
# Normalização dos dados
scaler = StandardScaler()
X_train_norm = scaler.fit_transform(X_train)
X_test_norm = scaler.fit_transform(X_test)

minmax = MinMaxScaler()
X_train_MinMax = minmax.fit_transform(X_train)
X_test_MinMax = minmax.fit_transform(X_test)
```

5.1. K-Nearest Neighbors

In [61]:
```
# Inicializando Classificador kNN
knn = KNeighborsClassifier()

# Hiper-parâmetros do kNN
params_knn = {
    "n_neighbors": tuple(range(1, 31)),
    "p": tuple(range(1, 6))
}
```

In [69]:
```
# Cross-validation Holdout
knn_holdout = GridSearchCV(estimator=knn, param_grid=params_knn, cv=StratifiedShuffleSpl

# Treinando modelo
knn_holdout.fit(X_train_MinMax, y_train)

print(f'Os melhores parâmetros para o kNN com o Holdout foram: {knn_holdout.best_params_

# Predizendo conjunto de teste
y_prediction = knn_holdout.predict(X_test_MinMax)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

```
Os melhores parâmetros para o kNN com o Holdout foram: {'n_neighbors': 1, 'p': 4}
Precision: 0.972488038277512
Recall: 0.9704545454545453
F1 Score:  0.9702885838374226
Accuracy: 0.9704545454545455
```

In [72]:
```
# Cross-Validation K-Fold
knn_kfold = GridSearchCV(estimator=knn, param_grid=params_knn, cv=5)

# Treinando modelo
knn_kfold.fit(X_train_MinMax, y_train)

print(f'Os melhores parâmetros para o kNN com o 5-fold foram: {knn_kfold.best_params_}')

# Predizendo conjunto de teste
y_prediction = knn_kfold.predict(X_test_MinMax)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

```
Os melhores parâmetros para o kNN com o 5-fold foram: {'n_neighbors': 3, 'p': 1}
Precision: 0.9788600288600289
Recall: 0.9772727272727271
```

```
F1 Score:  0.9772614925433224
Accuracy: 0.9772727272727273
```

In [74]:
```python
# Cross-Validation Leave-One-Out
knn_loo = GridSearchCV(estimator=knn, param_grid=params_knn, cv=LeaveOneOut(), n_jobs=-1

# Treinando modelo
knn_loo.fit(X_train_MinMax, y_train)

print(f'Os melhores parâmetros para o kNN com o Leave-One-Out foram: {knn_loo.best_param

# Predizendo conjunto de teste
y_prediction = knn_loo.predict(X_test_MinMax)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

```
Os melhores parâmetros para o kNN com o Leave-One-Out foram: {'n_neighbors': 5, 'p': 1}
Precision: 0.9803915781188508
Recall: 0.9795454545454544
F1 Score:  0.9794670348906549
Accuracy: 0.9795454545454545
```

5.2. Árvore de Decisão

In [77]:
```python
# Inicializando Classificador Decision Tree
decision_tree = DecisionTreeClassifier(random_state=50)

# Hiper-parâmetros do Decision tree
params_dt = {
    "max_depth": range(5, 21),
    "criterion": ["gini", "log_loss", "entropy"]
}
```

In [78]:
```python
# Decision Tree Cross-validation Holdout
dt_holdout = GridSearchCV(estimator=decision_tree, param_grid=params_dt, cv=StratifiedSh

# Treinando modelo
dt_holdout.fit(X_train_norm, y_train)

print(f'Os melhores parâmetros para o Decision Tree com o Holdout foram: {dt_holdout.bes

# Predizendo conjunto de teste
y_prediction = dt_holdout.predict(X_test_norm)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

```
Os melhores parâmetros para o Decision Tree com o Holdout foram: {'criterion': 'gini',
'max_depth': 12}
Precision: 0.9788469106650924
Recall: 0.9772727272727271
F1 Score:  0.9771376106421035
Accuracy: 0.9772727272727273
```

In [79]:
```
# Decision Tree Cross-validation K-Fold
dt_kfold = GridSearchCV(estimator=decision_tree, param_grid=params_dt, cv=5)

# Treinando modelo
dt_kfold.fit(X_train_norm, y_train)

print(f'Os melhores parâmetros para o Decision Tree com o 5-fold foram: {dt_kfold.best_p

# Predizendo conjunto de teste
y_prediction = dt_kfold.predict(X_test_norm)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

```
Os melhores parâmetros para o Decision Tree com o 5-fold foram: {'criterion': 'gini', 'm
ax_depth': 13}
Precision: 0.9808146399055488
Recall: 0.9795454545454544
F1 Score:  0.9794203086886014
Accuracy: 0.9795454545454545
```

5.3. Multi-Layer Perceptron

In [87]:
```
# Inicializando MLP Classifier
mlp = MLPClassifier(random_state=50)

# Hiper-parâmetros do MLP
params_mlp = {
    "hidden_layer_sizes": [[], [5], [10], [5, 5], [10, 10]]
}
```

In [91]:
```
# MLP Cross-validation Holdout
mlp_holdout = GridSearchCV(estimator=mlp, param_grid=params_mlp, cv=StratifiedShuffleSpl

# Treinando modelo
mlp_holdout.fit(X_train_norm, y_train)

print(f'Os melhores parâmetros para o MLP com o Holdout foram: {mlp_holdout.best_params_

# Predizendo conjunto de teste
y_prediction = mlp_holdout.predict(X_test_norm)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
```

```
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
Os melhores parâmetros para o MLP com o Holdout foram: {'hidden_layer_sizes': [10, 10]}
Precision: 0.963021534552635
Recall: 0.9613636363636361
F1 Score:  0.9612955080862654
Accuracy: 0.9613636363636363
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
```

In [92]:

```
# MLP Cross-validation K-Fold
dt_kfold = GridSearchCV(estimator=decision_tree, param_grid=params_dt, cv=5)

# Treinando modelo
dt_kfold.fit(X_train_norm, y_train)

print(f'Os melhores parâmetros para o MLP com o 5-fold foram: {dt_kfold.best_params_}')

# Predizendo conjunto de teste
y_prediction = dt_kfold.predict(X_test_norm)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

```
Os melhores parâmetros para o MLP com o 5-fold foram: {'criterion': 'gini', 'max_depth':
13}
Precision: 0.9808146399055488
Recall: 0.9795454545454544
F1 Score:  0.9794203086886014
Accuracy: 0.9795454545454545
```

5.4. Support Vector Machine

In [83]:
```
# Inicializando SVM
svm = SVC()

# K-fold
kfold = StratifiedKFold(n_splits=5)

# Acurácia do Treinamento
scores = []

# Aplicando Cross-Validation K-Fold
for train_index, test_index in kfold.split(X_train, y_train):
  # Separando conjunto de treinamento com base no k-fold
  train_data, test_data = X_train_norm[train_index], X_train_norm[test_index]
  train_labels, test_labels = y_train.values[train_index], y_train.values[test_index]

  # Treinando modelo
  svm.fit(train_data, train_labels)

  # Calculando acurácia do conjunto de teste do k-fold
  accuracy = svm.score(test_data, test_labels)

  # Coletando acurácia do treinamento
  scores.append(accuracy)

print('Acurácia no treino com 5-fold', np.mean(scores))
print('Acurácia no teste: ', svm.score(X_test_norm, y_test))
```

```
Acurácia no treino com 5-fold 0.9761363636363637
Acurácia no teste:  0.9818181818181818
```

In [ ]:
```
# Predizendo conjunto de teste
y_prediction = dt_kfold.predict(X_test_norm)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

5.5. Random Forest

In [84]:
```
# Inicializando Random Forest Classifier
random_forest = RandomForestClassifier(random_state=50)

# Hiper-parâmetros do Random Forest
params_rf = {
    "max_depth": range(5, 21)
}
```

In [85]:
```
# Random Forest Cross-validation Holdout
rf_holdout = GridSearchCV(estimator=random_forest, param_grid=params_rf, cv=StratifiedSh

# Treinando modelo
rf_holdout.fit(X_train_norm, y_train)

print(f'Os melhores parâmetros para o Random Forest com o Holdout foram: {rf_holdout.bes

# Predizendo conjunto de teste
```

```
y_prediction = rf_holdout.predict(X_test_norm)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

```
Os melhores parâmetros para o Random Forest com o Holdout foram: {'max_depth': 8}
Precision: 0.991538764266037
Recall: 0.990909090909091
F1 Score:  0.9908948561066662
Accuracy: 0.990909090909091
```

In [90]:
```
# Random Forest Cross-validation K-Fold
rf_kfold = GridSearchCV(estimator=random_forest, param_grid=params_rf, cv=5)

# Treinando modelo
rf_kfold.fit(X_train_norm, y_train)

print(f'Os melhores parâmetros para o Random Forest com o 5-fold foram: {rf_kfold.best_p

# Predizendo conjunto de teste
y_prediction = rf_kfold.predict(X_test_norm)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

```
Os melhores parâmetros para o Random Forest com o 5-fold foram: {'max_depth': 14}
Precision: 0.9895710350255804
Recall: 0.9886363636363636
F1 Score:  0.9886121580601682
Accuracy: 0.9886363636363636
```

5.6. Naive Bayes

In [93]:
```
# Inicializando o Naive Bayes
nb = GaussianNB()

# Determinando os parâmetros do Grid Search
# https://medium.com/analytics-vidhya/how-to-improve-naive-bayes-9fa698e14cba
params_nb = {
    'var_smoothing': np.logspace(0, -9, num=100)
}
```

In [94]:
```
# Naive Bayes Cross-validation Holdout
nb_holdout = GridSearchCV(estimator=nb, param_grid=params_nb, cv=5)

# Treinando modelo
nb_holdout.fit(X_train_norm, y_train)
```

```
print(f'Os melhores parâmetros para o Naive Bayes com o Holdout foram: {nb_holdout.best_

# Predizendo conjunto de teste
y_prediction = nb_holdout.predict(X_test_norm)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

```
Os melhores parâmetros para o Naive Bayes com o Holdout foram: {'var_smoothing': 0.00053
36699231206307}
Precision: 0.9937032664305392
Recall: 0.993181818181818
F1 Score:  0.9931690047222781
Accuracy: 0.9931818181818182
```

In [95]:
```
# Naive Bayes Cross-validation K-Fold
nb_kfold = GridSearchCV(estimator=nb, param_grid=params_nb, cv=5)

# Treinando modelo
nb_kfold.fit(X_train_norm, y_train)

print(f'Os melhores parâmetros para o Naive Bayes com o 5-fold foram: {nb_kfold.best_par

# Predizendo conjunto de teste
y_prediction = nb_kfold.predict(X_test_norm)

# Calculando métricas para avaliação
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction, average=None)
recall = recall_score(y_test, y_prediction, average=None)
f1 = f1_score(y_test, y_prediction, average=None)

print("Precision:", np.mean(precision))
print("Recall:", np.mean(recall))
print("F1 Score: ", np.mean(f1))
print("Accuracy:", np.mean(accuracy))
```

```
Os melhores parâmetros para o Naive Bayes com o 5-fold foram: {'var_smoothing': 0.000533
6699231206307}
Precision: 0.9937032664305392
Recall: 0.993181818181818
F1 Score:  0.9931690047222781
Accuracy: 0.9931818181818182
```

6. Referências Bibliográficas

- SISTEMA DE PRODUÇÃO DE MELÃO. [S. l.]: Embrapa Semiárido, ISSN 1807-0027. Mensal. Disponível em: http://www.cpatsa.embrapa.br:8080/sistema_producao/spmelao/manejo_do_solo.html. Acesso em: 1 maio 2023;

- SOUSA, Rafaela. Rotação de culturas. [S. l.]: Brasil Escola. Disponível em: https://brasilescola.uol.com.br/geografia/rotacao-culturas.htm. Acesso em: 1 maio 2023.

In [ ]:
```
!jupyter nbconvert --to webpdf --allow-chromium-download "Análise de Culturas.ipynb"
```

```
[NbConvertApp] Converting notebook Análise de Culturas.ipynb to webpdf
[NbConvertApp] Building PDF
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 2532278 bytes to Análise de Culturas.pdf
```