```
 import numpy as np
 import pandas as pd
 import matplotlib.pyplot as plt
 import seaborn as sns


 #  INITIATING THE HYPERPARAMETERS
class SVM:
    def __init__(self, learning_rate, no_of_iterations, lambda_parameter):
        self.learning_rate = learning_rate
        self.no_of_iterations = no_of_iterations
        self.lambda_parameter = lambda_parameter

    # fitting the dataset to SVM Classifier
    def fit(self, X, Y):
        # m  --> number of Data points --> number of rows
        # n  --> number of input features --> number of columns
        self.m, self.n = X.shape
        # initiating the weight value and bias value
        self.w = np.zeros(self.n)
        self.b = 0
        self.X = X
        self.Y = Y
        # implementing Gradient Descent algorithm for Optimization
        for i in range(self.no_of_iterations): # Now within the fit method
            self.update_weights()


# function for updating the weight and bias value
def update_weights(self):
    # label encoding
    y_label = np.where(self.Y <= 0, -1, 1)
    # gradients ( dw, db)
    for index, x_i in enumerate(self.X): #Fixed: Corrected indentation for this line
        condition = y_label[index] * (np.dot(x_i, self.w) - self.b) >= 1
        if (condition == True):
            dw = 2 * self.lambda_parameter * self.w
            db = 0
        else:
            dw = 2 * self.lambda_parameter * self.w - np.dot(x_i, y_label[index])
            db = y_label[index]
        self.w = self.w - self.learning_rate * dw
        self.b = self.b - self.learning_rate * db


# predict the label for a given input value
def predict(self, X):
        output = np.dot(X, self.w) - self.b
        predicted_labels = np.sign(output)
        y_hat = np.where(predicted_labels <= -1, 0, 1)
        return y_hat


 from sklearn.preprocessing import StandardScaler
 from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score


# LOADING THE DATA FROM CSV FILES TO PANDA DATAFRAME
crop_data = pd.read_csv('/content/projectdata2.csv')


#PRINT FIRST 5 ROWS OF THE DATAFRAME
crop_data.head()
```

	temperature	humidity	ph	rainfall	crop	soil_moisture	soil_type	sunli
0	20.879744	82.002744	6.502985	202.935536	rice	29.446064	2	
1	21.770462	80.319644	7.038096	226.655537	rice	12.851183	3	
2	23.004459	82.320763	7.840207	263.964248	rice	29.363913	2	
3	26.491096	80.158363	6.980401	242.864034	rice	26.207732	3	
4	20.130175	81.604873	7.628473	262.717340	rice	28.236236	2	

```
#NUMBER OF ROW AND COLUMNS IN A DATASET
crop_data.shape
```

```
(2200, 12)
```

```
#GETTING STATISTICAL MEASURES OF THE DATASET
crop_data.describe()
```

	temperature	humidity	ph	rainfall	soil_moisture	soil_type
count	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000
mean	25.616244	71.481779	6.469480	103.463655	20.151388	1.991364
std	5.063749	22.263812	0.773938	54.958389	5.793720	0.812263
min	8.825675	14.258040	3.504752	20.211267	10.024260	1.000000
25%	22.769375	60.261953	5.971693	64.551686	15.179949	1.000000
50%	25.598693	80.473146	6.425045	94.867624	20.088481	2.000000
75%	28.561654	89.948771	6.923643	124.267508	25.255146	3.000000
max	43.675493	99.981876	9.935091	298.560117	29.997860	3.000000

```
#COUNT OF THE DATASET
if 'Days_to_Harvest' not in crop_data.columns:
  print(f"Available columns: {crop_data.columns.tolist()}")
else:
  print(crop_data['Days_to_Harvest'].value_counts())
```

```
Available columns: ['temperature', 'humidity', 'ph', 'rainfall', 'crop', 'soil_moistu
```

```
features = crop_data.drop(columns='irrigation_frequency', axis=1)
target = crop_data['irrigation_frequency']
```

```
print(features)
```

```
      temperature   humidity        ph    rainfall    crop  soil_moisture  \
0       20.879744  82.002744  6.502985  202.935536    rice      29.446064
1       21.770462  80.319644  7.038096  226.655537    rice      12.851183
2       23.004459  82.320763  7.840207  263.964248    rice      29.363913
3       26.491096  80.158363  6.980401  242.864034    rice      26.207732
4       20.130175  81.604873  7.628473  262.717340    rice      28.236236
...           ...        ...       ...         ...     ...            ...
2195    26.774637  66.413269  6.780064  177.774507  coffee      10.697757
2196    27.417112  56.636362  6.086922  127.924610  coffee      12.203830
2197    24.131797  67.225123  6.362608  173.322839  coffee      28.989176
2198    26.272418  52.127394  6.758793  127.175293  coffee      13.642305
2199    23.603016  60.396475  6.779833  140.937041  coffee      23.911728

      soil_type  sunlight_exposure  water_source_type  frost_risk  \
0             2           8.677355                  3   95.649985
1             3           5.754288                  2   77.265694
2             2           9.875230                  2   18.192168
3             3           8.023685                  3   82.818720
4             2           8.120512                  3   25.466499
...         ...                ...                ...         ...
2195          1          10.330875                  1   77.719639
2196          3           6.070558                  3   22.336839
2197          3          11.097182                  3   41.782729
2198          2           8.097337                  3   49.619791
2199          3           8.639742                  2   47.271267

      water_usage_efficiency
0                   1.193293
1                   1.752672
2                   3.035541
3                   1.273341
4                   2.578671
...                      ...
2195                4.111619
2196                4.190796
2197                2.447010
2198                4.119388
2199                2.758819

[2200 rows x 11 columns]
```

```
print(target)
```

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

```
       ..
2195    5
2196    6
2197    5
2198    6
2199    3
Name: irrigation_frequency, Length: 2200, dtype: int64
```

```
# Import necessary libraries
import pandas as pd
from sklearn.preprocessing import StandardScaler
# Assuming 'features' is the DataFrame
# Select only numerical features for scaling
numerical_features = features.select_dtypes(include=['number']).columns
features_to_scale = features[numerical_features]
# Now fit the scaler on the numerical features only
scaler = StandardScaler()
scaler.fit(features_to_scale)
```

```
StandardScaler
StandardScaler()
```

```
# Transform only the numerical features
standardized_data_numerical = scaler.transform(features_to_scale)
# 1. Create a new DataFrame with standardized numerical features
standardized_data = pd.DataFrame(standardized_data_numerical, columns=numerical_features
# 2. Concatenate the standardized numerical features with the original non-numerical fea
standardized_data = pd.concat([standardized_data, features.drop(columns=numerical_featur
print(standardized_data)
```

```
      temperature  humidity        ph  rainfall  soil_moisture  soil_type  \
0       -0.935587  0.472666  0.043302  1.810361       1.604632   0.010635
1       -0.759646  0.397051  0.734873  2.242058      -1.260307   1.242044
2       -0.515898  0.486954  1.771510  2.921066       1.590450   0.010635
3        0.172807  0.389805  0.660308  2.537048       1.045567   1.242044
4       -1.083647  0.454792  1.497868  2.898373       1.395768   0.010635
...           ...       ...       ...       ...            ...        ...
2195     0.228814 -0.227709  0.401395  1.352437      -1.632074  -1.220774
2196     0.355720 -0.666947 -0.494413  0.445183      -1.372066   1.242044
2197    -0.293218 -0.191235 -0.138120  1.271418       1.525755   1.242044
2198     0.129612 -0.869518  0.373904  0.431545      -1.123728   0.010635
2199    -0.397667 -0.498020  0.401096  0.682005       0.649185   1.242044

      sunlight_exposure  water_source_type  frost_risk  \
0              0.053402           1.227297    1.586950
1             -1.408916          -0.004479    0.943555
2              0.652660          -0.004479   -1.123843
3             -0.273609           1.227297    1.137894
4             -0.225170           1.227297   -0.869263
...                 ...                ...         ...
2195           0.880605          -1.236255    0.959441
2196          -1.250696           1.227297   -0.978792
2197           1.263964           1.227297   -0.298244
2198          -0.236763           1.227297   -0.023970
2199           0.034585          -0.004479   -0.106161
```

```
      water_usage_efficiency    crop
0                  -1.577155    rice
1                  -1.086204    rice
2                   0.039735    rice
3                  -1.506899    rice
4                  -0.361247    rice
...                      ...     ...
2195                0.984178  coffee
2196                1.053670  coffee
2197               -0.476802  coffee
2198                0.990997  coffee
2199               -0.203136  coffee

[2200 rows x 11 columns]
```

```
features = standardized_data
target = crop_data['irrigation_frequency']
print(features)
print(target)
```

```
      temperature  humidity        ph  rainfall  soil_moisture  soil_type  \
0       -0.935587  0.472666  0.043302  1.810361       1.604632   0.010635
1       -0.759646  0.397051  0.734873  2.242058      -1.260307   1.242044
2       -0.515898  0.486954  1.771510  2.921066       1.590450   0.010635
3        0.172807  0.389805  0.660308  2.537048       1.045567   1.242044
4       -1.083647  0.454792  1.497868  2.898373       1.395768   0.010635
...           ...       ...       ...       ...            ...        ...
2195     0.228814 -0.227709  0.401395  1.352437      -1.632074  -1.220774
2196     0.355720 -0.666947 -0.494413  0.445183      -1.372066   1.242044
2197    -0.293218 -0.191235 -0.138120  1.271418       1.525755   1.242044
2198     0.129612 -0.869518  0.373904  0.431545      -1.123728   0.010635
2199    -0.397667 -0.498020  0.401096  0.682005       0.649185   1.242044

      sunlight_exposure  water_source_type  frost_risk  \
0              0.053402           1.227297    1.586950
1             -1.408916          -0.004479    0.943555
2              0.652660          -0.004479   -1.123843
3             -0.273609           1.227297    1.137894
4             -0.225170           1.227297   -0.869263
...                 ...                ...         ...
2195           0.880605          -1.236255    0.959441
2196          -1.250696           1.227297   -0.978792
2197           1.263964           1.227297   -0.298244
2198          -0.236763           1.227297   -0.023970
2199           0.034585          -0.004479   -0.106161

      water_usage_efficiency    crop
0                  -1.577155    rice
1                  -1.086204    rice
2                   0.039735    rice
3                  -1.506899    rice
4                  -0.361247    rice
...                      ...     ...
2195                0.984178  coffee
2196                1.053670  coffee
2197               -0.476802  coffee
2198                0.990997  coffee
2199               -0.203136  coffee
```

```
[2200 rows x 11 columns]
0       4
1       4
2       1
3       1
4       3
       ..
2195    5
2196    6
2197    5
2198    6
2199    3
Name: irrigation_frequency, Length: 2200, dtype: int64
```

```
X_train, X_test, Y_train, Y_test = train_test_split(features, target, test_size=0.2, ran
print(features.shape, X_train.shape, X_test.shape)
```

```
(2200, 11) (1760, 11) (440, 11)
```

```
from sklearn.svm import SVC
classifier = SVC(kernel='linear', C=1/0.01)
!pip install -q pandas scikit-learn
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.impute import SimpleImputer
```

```
categorical_features = ['crop']
numerical_features = features.drop(columns=categorical_features).columns
```

```
# Create a ColumnTransformer to apply OneHotEncoder to categorical features
# and SimpleImputer to handle NaN values
preprocessor = ColumnTransformer(
transformers=[
('num', SimpleImputer(strategy='mean'), numerical_features),  # Impute numerical feature
('cat', OneHotEncoder(sparse_output=False, handle_unknown='ignore'), categorical_feature
])
```

```
# Fit and transform the data
features_encoded = preprocessor.fit_transform(features)
# Split the encoded data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(features_encoded, target, test_size=0
```

```
# Create and train the SVC classifier
classifier = SVC(kernel='linear', C=1/0.01)
classifier.fit(X_train, Y_train)
print(features_encoded.shape, X_train.shape, X_test.shape)
```

```
(2200, 32) (1760, 32) (440, 32)
```

```
 # accuracy on training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
print('Accuracy score on training data = ', training_data_accuracy)
```

```
Accuracy score on training data =  0.23636363636363636
```

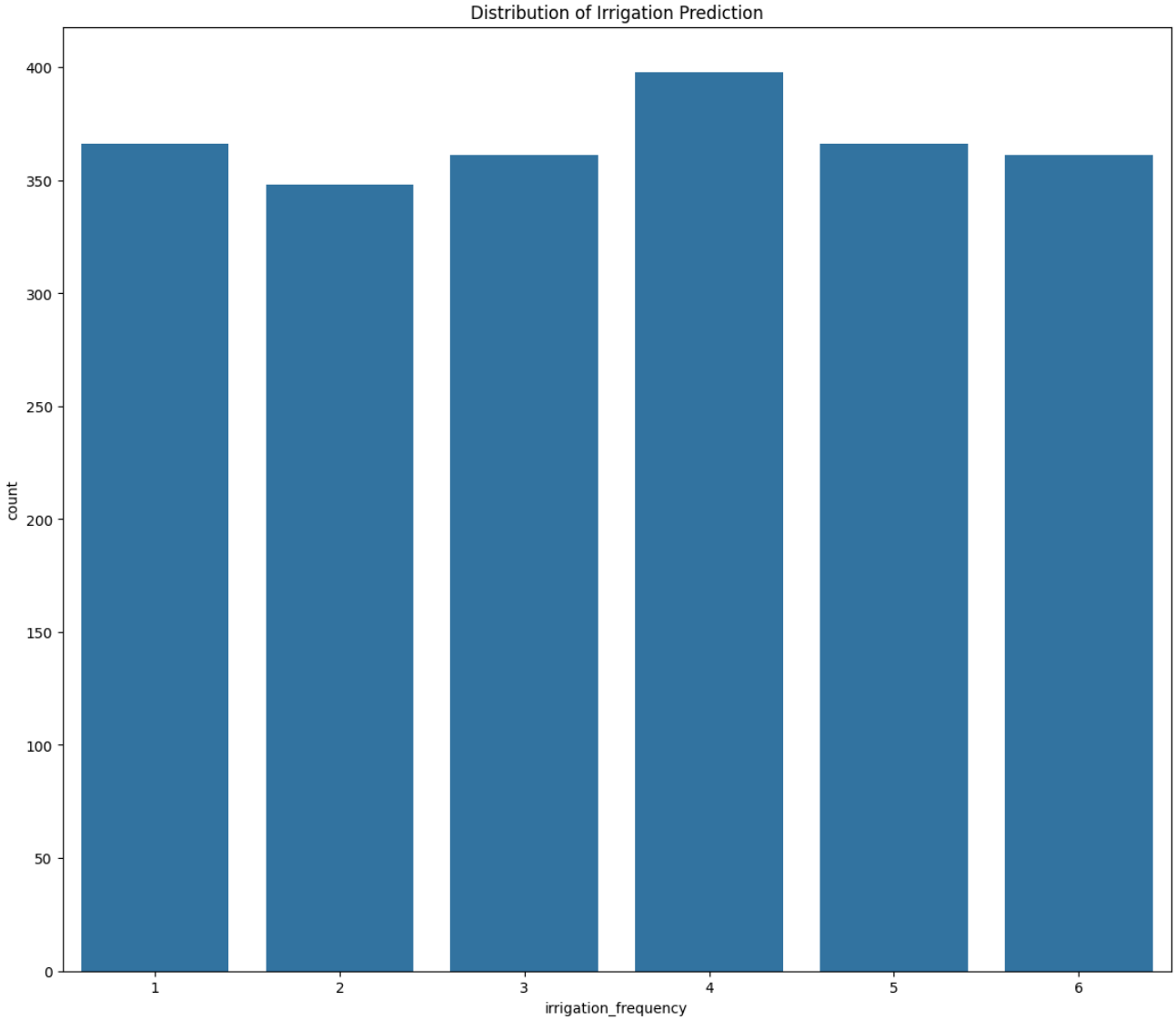
```
 # accuracy on training data
 X_test_prediction = classifier.predict(X_test)
 test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
 print('Accuracy score on test data = ', test_data_accuracy)
```

```
Accuracy score on test data =  0.18863636363636363
```

```
 # BUILD A PREDICTIVE SYSTEM
 # Input sequence from the user
input_sequence = input("Enter the crop: ")
# Create a DataFrame with all expected columns
input_data = pd.DataFrame({
    'crop': [input_sequence],
    'temperature': [0],
    'humidity':[0],
     'ph': [0],
     'rainfall':[0],
    'sunlight_exposure':[0],
    'soil_moisture':[0],
    'soil_type':[0],
    'wind_speed':[0],
    'water_source_type':[0],
    'frost_risk':[0],
    'water_usage_efficiency':[0]
})
 # Preprocess the input data using the same preprocessor used for training
 try:
  input_encoded = preprocessor.transform(input_data)
    # Make prediction using the trained classifier
  prediction = classifier.predict(input_encoded)
    # Print the predicted irrigation
  print("Predicted irrigation:", prediction[0])
except ValueError as e:
    print(f"Error during prediction: {e}")
    print("Please ensure the input sequence and other features are in the correct format.
```

```
Enter the crop: maize
 Predicted irrigation: 2
```

```
 # Distribution of the target variable
plt.figure(figsize=(14, 12))
sns.countplot(x=target)
plt.title('Distribution of Irrigation Prediction')
plt.show()
```


Distribution of Irrigation Prediction
count
irrigation_frequency
0
50
100
150
200
250
300
350
400
1
2
3
4
5
6

```
#Comparison of training and testing accuracy
accuracy_scores = {'Training': training_data_accuracy, 'Testing': test_data_accuracy}
plt.figure(figsize=(14, 12))
plt.bar(accuracy_scores.keys(), accuracy_scores.values(), color=['lightblue', 'pink'])
plt.title('Model Accuracy Comparison')
plt.ylabel('Accuracy Score')
plt.ylim([0, 1]) # Set y-axis limit to 0-1 for accuracy
plt.show()
```