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

```
from google.colab import files
uploaded = files.upload()
```

Choose Files Algerian_fo…aned (1).csv

- **Algerian_forest_fires_cleaned (1).csv**(text/csv) - 15094 bytes, last modified: 6/4/2024 - 100% done

```
Saving Algerian_forest_fires_cleaned (1).csv to Algerian_forest_fires_cleaned (1) (2).csv
```

```
import pandas as pd
df = pd.read_csv('Algerian_forest_fires_cleaned (1).csv')
df.head()
```

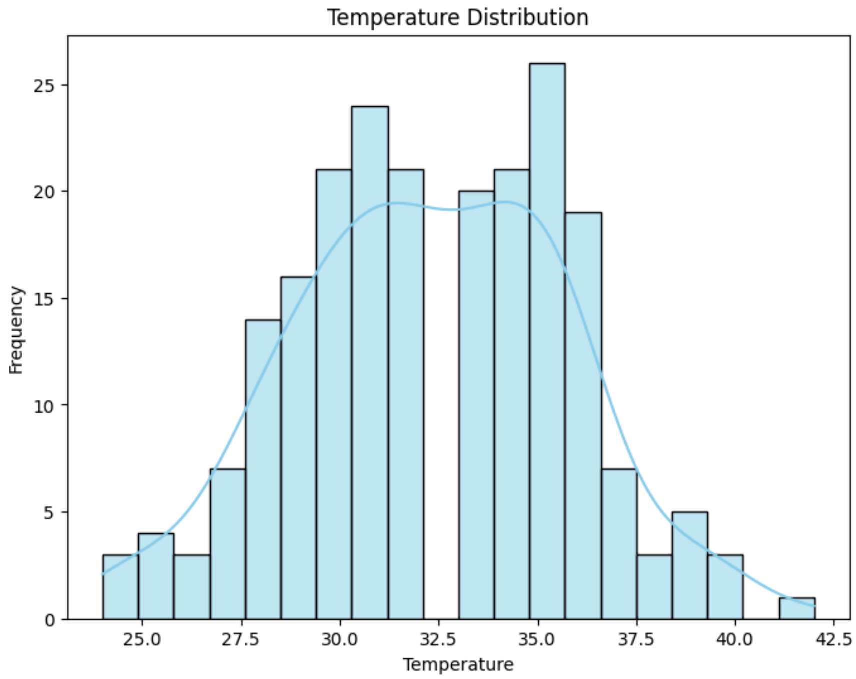
	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Region
0	1	6	2012	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	not fire	0
1	2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	not fire	0
2	3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire	0
3	4	6	2012	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	not fire	0
4	5	6	2012	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	not fire	0

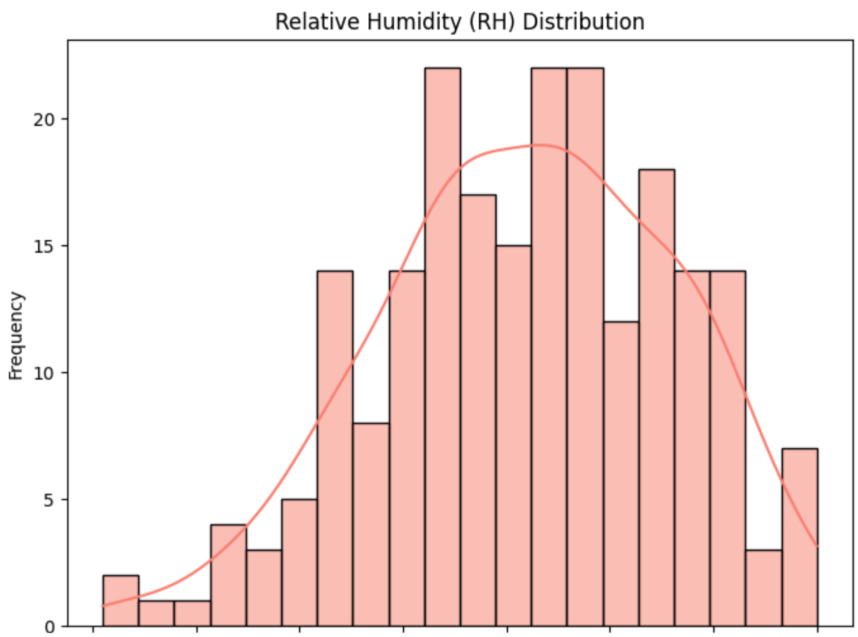
Next steps: Generate code with df View recommended plots

```
import pandas as pd
import numpy as np
from google.colab import files
import matplotlib.pyplot as plt


# Inspect the first few rows of the dataframe
print(df.head())

# Check for missing values
print(df.isnull().sum())



# If there are categorical columns with missing values, fill them with the mode
# df['categorical_column'].fillna(df['categorical_column'].mode()[0], inplace=True)

# Check for data types
print(df.dtypes)

# Convert data types if necessary (Example: day, month, year should be integers)
df['day'] = df['day'].astype(int)
df['month'] = df['month'].astype(int)
df['year'] = df['year'].astype(int)

# Handling outliers (Example: Removing outliers beyond 3 standard deviations)
numeric_columns = ['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI', 'FWI']
for col in numeric_columns:
    mean = df[col].mean()
    std = df[col].std()
    df = df[(df[col] > mean - 3*std) & (df[col] < mean + 3*std)]

# Feature Selection
# Assuming 'Classes' is the target variable and 'Region' might be categorical
selected_features = ['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Region']
df = df[selected_features + ['Classes']]


#  Creating a new feature 'season' based on 'month'
def get_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else:
        return 'Autumn'

df['season'] = df['month'].apply(get_season)

# Convert categorical features to numerical (One-Hot Encoding)
df = pd.get_dummies(df, columns=['season', 'Region'], drop_first=True)

# Final dataset
print(df.head())

# Save the cleaned and preprocessed dataset to a new CSV file
df.to_csv('cleaned_preprocessed_data.csv', index=False)

# Download the cleaned and preprocessed file
files.download('cleaned_preprocessed_data.csv')
```

```
   day  month  year  Temperature  RH  Ws  Rain  FFMC  DMC    DC  ISI  BUI  \
0    1      6  2012           29  57  18   0.0  65.7  3.4   7.6  1.3  3.4   
1    2      6  2012           29  61  13   1.3  64.4  4.1   7.6  1.0  3.9   
2    3      6  2012           26  82  22  13.1  47.1  2.5   7.1  0.3  2.7   
3    4      6  2012           25  89  13   2.5  28.6  1.3   6.9  0.0  1.7   
4    5      6  2012           27  77  16   0.0  64.8  3.0  14.2  1.2  3.9   

   FWI      Classes  Region  
0  0.5  not fire          0  
1  0.4  not fire          0  
2  0.1  not fire          0  
3  0.0  not fire          0  
4  0.5  not fire          0  
day            0
month          0
year           0
Temperature    0
RH             0
Ws             0
Rain           0
FFMC           0
DMC            0
DC             0
ISI            0
BUI            0
FWI            0
Classes        0
Region         0
dtype: int64
day              int64
month            int64
year             int64
Temperature      int64
RH               int64
Ws               int64
Rain           float64
FFMC           float64
DMC            float64
DC             float64
ISI            float64
BUI            float64
FWI            float64
Classes         object
Region           int64
dtype: object
   day  month  year  Temperature  RH  Ws  Rain  FFMC  DMC    DC  ISI   BUI  \
0    1      6  2012           29  57  18   0.0  65.7  3.4   7.6  1.3   3.4   
1    2      6  2012           29  61  13   1.3  64.4  4.1   7.6  1.0   3.9   
4    5      6  2012           27  77  16   0.0  64.8  3.0  14.2  1.2   3.9   
5    6      6  2012           31  67  14   0.0  82.6  5.8  22.2  3.1   7.0   
6    7      6  2012           33  54  13   0.0  88.2  9.9  30.5  6.4  10.9   

   FWI      Classes  season_Summer  Region_1  
0  0.5  not fire              True     False  
1  0.4  not fire              True     False  
4  0.5  not fire              True     False  
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# considering 'df' is your DataFrame containing the dataset

# Plotting Temperature distribution
plt.figure(figsize=(8, 6))
sns.histplot(df['Temperature'], bins=20, kde=True, color='skyblue')
plt.title('Temperature Distribution')
plt.xlabel('Temperature')
plt.ylabel('Frequency')
plt.show()

# Plotting Relative Humidity (RH) distribution
plt.figure(figsize=(8, 6))
sns.histplot(df['RH'], bins=20, kde=True, color='salmon')
plt.title('Relative Humidity (RH) Distribution')
plt.xlabel('RH')
plt.ylabel('Frequency')
plt.show()

# Plotting Wind Speed (Ws) distribution
plt.figure(figsize=(8, 6))
sns.histplot(df['Ws'], bins=20, kde=True, color='lightgreen')
plt.title('Wind Speed (Ws) Distribution')
plt.xlabel('Ws')
plt.ylabel('Frequency')
plt.show()

# Plotting Rain distribution
plt.figure(figsize=(8, 6))
sns.histplot(df['Rain'], bins=20, kde=True, color='gold')
plt.title('Rain Distribution')
plt.xlabel('Rain')
plt.ylabel('Frequency')
plt.show()
```


Temperature Distribution
Frequency
Temperature
0
5
10
15
20
25
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5


Relative Humidity (RH) Distribution
Frequency
0
5
10
15
20

```
from sklearn.preprocessing import LabelEncoder

# Encode categorical target variable
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)

# One-hot encode the target variable
from sklearn.preprocessing import OneHotEncoder
onehot_encoder = OneHotEncoder(sparse=False)
y_train_encoded = y_train_encoded.reshape(len(y_train_encoded), 1)
y_train_onehot = onehot_encoder.fit_transform(y_train_encoded)

# Linear Regression
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train_onehot)

# Polynomial Regression
poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train_onehot)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWa
  warnings.warn(
▾ LinearRegression
LinearRegression()
```

```
from sklearn.preprocessing import LabelEncoder

# Encode categorical target variable
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)

# Lasso Regression
lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X_train_scaled, y_train_encoded)

# Ridge Regression
ridge_reg = Ridge(alpha=0.1)
ridge_reg.fit(X_train_scaled, y_train_encoded)
```

```
▾ Ridge
Ridge(alpha=0.1)
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import Lasso, Ridge
from sklearn.metrics import mean_squared_error

# Split the data into features (X) and target variable (y)
X = df[['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI', 'FWI']]
y = df['Classes']

# Encode the target variable (assuming non-numeric)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

# Create a pipeline with scaling and a regression model (Lasso or Ridge)
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('regression', Lasso())  # Change to Ridge if desired
])

# Define a parameter grid for hyperparameter tuning
param_grid = {
    'regression__alpha': [0.1, 0.5, 1.0]  # Adjust alpha values as needed
}

# Perform GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Evaluate the best model on the test set
y_pred = best_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)

print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.10700365666244754
```

Effective use of cross validation : Cross-validation is like testing how good your cooking skills are by making different dishes multiple times. Instead of just making one dish and hoping it turns out well, you make several dishes using different ingredients and recipes. This way, you get a better idea of how well you cook overall.

Similarly, in machine learning, cross-validation is like testing how good a model is by training it on different parts of the data multiple times. Instead of just training once and hoping for the best, you train the model multiple times on different parts of the data and see how well it performs on average.

This technique helps in choosing the best model, tweaking its settings (like adjusting the heat when cooking), and making sure it doesn't just work well on the data it was trained on, but also on new data it hasn't seen before. Overall, cross-validation helps in making sure your model is