AI-Powered Precipitation Prediction System for Small-Scale Farmers

Tushar Bhakat

20.09.2024

Problem Statement

Small-scale farmers often face significant challenges due to unpredictable weather patterns, which can result in poor crop yields and financial losses. Accurate precipitation prediction is essential for these farmers to plan their agricultural activities effectively. The core problem involves enhancing agricultural productivity and sustainability through the integration of Artificial Intelligence (AI). Key issues include unpredictable weather patterns, as traditional forecasting methods are often inadequate for precise agricultural planning. This inadequacy leads to challenges such as improper irrigation scheduling and inefficient resource use, negatively affecting crop yields and farmer income. Additionally, many current irrigation systems are inefficient, lacking the capability to adapt in real-time to changing weather conditions and soil moisture levels, which can cause water wastage or insufficient watering, thereby harming crop health and productivity. Another issue is the limited utilization of available data, as there is often a gap between meteorological data and its practical application in agriculture, leading to suboptimal farming practices. Furthermore, technology accessibility is a challenge, as small-scale and resource-limited farmers may find it difficult to access advanced AI and IoT technologies, which are typically designed for larger operations, creating a disparity in the adoption of technologies that could enhance farming efficiency and resilience. Addressing these issues requires the development of AI-powered solutions that offer accurate, real-time weather predictions, optimize irrigation systems, and are accessible to farmers of all scales.

Step 1: Prototype Selection

a. Feasibility

- The AI-powered precipitation prediction system can be feasibly developed in the next 2-3 years. Machine learning and weather prediction technologies are already well-established, and the availability of historical weather data, satellite feeds, and APIs (like OpenWeatherMap) makes it possible to create reliable prediction models within this timeframe.
- Developing a mobile or web app for farmers to access the predictions is also feasible given the rise in internet penetration and smartphone usage in rural areas.

b. Viability

- The system has long-term viability, as climate change and unpredictable weather patterns are ongoing concerns for farmers worldwide. Over the next 20-30 years, the demand for accurate weather forecasting will only grow as agricultural practices need to adapt to shifting climates.
- The system could evolve to include other features like pest/disease prediction, soil analysis, and crop management, ensuring its relevance and survival in the future.

c. Monetization

- The product is directly monetizable through subscription models. Farmers could be charged a monthly fee for premium features such as advanced weather predictions, crop-specific recommendations, and consulting services. Additionally, partnerships with agro-tech companies and government agencies can further boost revenue streams.

Step 2: Prototype Development

Small-Scale Code Implementation

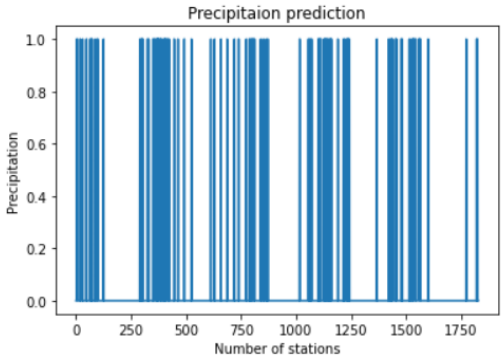
Data Collection: Dataset

```
#importing data
import pandas as pd
df=pd.read_csv('dataset.csv')
print(df)
```

```
         STATION                              NAME        DATE  AWND  PGTM  \
0     USW00093134  LOS ANGELES DOWNTOWN USC, CA US  2016-01-01  2.46   NaN
1     USW00093134  LOS ANGELES DOWNTOWN USC, CA US  2016-01-02  2.01   NaN
2     USW00093134  LOS ANGELES DOWNTOWN USC, CA US  2016-01-03  0.67   NaN
3     USW00093134  LOS ANGELES DOWNTOWN USC, CA US  2016-01-04  1.34   NaN
4     USW00093134  LOS ANGELES DOWNTOWN USC, CA US  2016-01-05  2.46   NaN
...           ...                              ...         ...   ...   ...
1822  USW00093134  LOS ANGELES DOWNTOWN USC, CA US  2020-12-27  1.12   NaN
1823  USW00093134  LOS ANGELES DOWNTOWN USC, CA US  2020-12-28  4.70   NaN
1824  USW00093134  LOS ANGELES DOWNTOWN USC, CA US  2020-12-29  1.57   NaN
1825  USW00093134  LOS ANGELES DOWNTOWN USC, CA US  2020-12-30  0.45   NaN
1826  USW00093134  LOS ANGELES DOWNTOWN USC, CA US  2020-12-31  1.57   NaN

      PRCP  TAVG  TMAX  TMIN   WDF2   WDF5  WSF2  WSF5  WT01  WT02  WT08
0     0.00   NaN    64    43   10.0   30.0   8.1  11.0   NaN   NaN   1.0
1     0.00   NaN    65    47  270.0   30.0   6.0   8.9   NaN   NaN   NaN
2     0.00   NaN    62    44  150.0  150.0  10.1  14.1   NaN   NaN   NaN
3     0.01   NaN    69    55  270.0  280.0   8.1  14.1   NaN   NaN   NaN
4     1.61   NaN    59    49  140.0  140.0  10.1  16.1   1.0   1.0   NaN
...    ...   ...   ...   ...    ...    ...   ...   ...   ...   ...   ...
1822  0.01   NaN    66    55  270.0  260.0   8.9  18.1   1.0   NaN   1.0
1823  1.81   NaN    56    47   90.0  260.0  14.1  21.0   1.0   NaN   NaN
1824  0.00   NaN    65    42  340.0  360.0  10.1  18.1   NaN   NaN   NaN
1825  0.00   NaN    69    44  260.0  260.0   6.9  12.1   NaN   NaN   NaN
1826  0.00   NaN    70    43  350.0  350.0  12.1  19.9   NaN   NaN   NaN

[1827 rows x 16 columns]
```

```
# replacing all PRCP values greater than 0 as 1
df.loc[df['PRCP']>0,'PRCP']=1
print(df)
```

```
%matplotlib inline
from matplotlib import pyplot as plt
import pandas as pd
plt.plot(df['PRCP'])
plt.title('Precipitaion prediction')
plt.xlabel('Number of stations')
plt.ylabel('Precipitation')
plt.show()
```




```
from sklearn.utils import resample,shuffle
df_1 = df[df['PRCP'] == 1]
other_df = df[df['PRCP'] == 0]
#upsample the minority class
df_1_upsampled = resample(df_1,random_state=None,n_samples=None,replace=True)
#concatenate the upsampled dataframe
df_upsampled = pd.concat([df_1_upsampled,other_df])
df_upsampled
```

	STATION	NAME	DATE	AWND	PGTM	PRCP	TAVG	TMAX	TMIN	WDF2	WDF5	WSF2	WSF5	WT01	WT02	WT08
369	USW00093134	LOS ANGELES DOWNTOWN USC, CA US	2017-01-04	1.12	NaN	1.0	NaN	63	47	90.0	100.0	8.1	13.0	1.0	NaN	1.0
1154	USW00093134	LOS ANGELES DOWNTOWN USC, CA US	2019-02-28	2.68	NaN	1.0	NaN	67	56	260.0	280.0	6.9	11.0	1.0	NaN	1.0
1822	USW00093134	LOS ANGELES DOWNTOWN USC, CA US	2020-12-27	1.12	NaN	1.0	NaN	66	55	270.0	260.0	8.9	18.1	1.0	NaN	1.0
1153	USW00093134	LOS ANGELES DOWNTOWN USC, CA US	2019-02-27	1.34	NaN	1.0	NaN	66	50	270.0	270.0	8.9	15.0	1.0	NaN	1.0
1822	USW00093134	LOS ANGELES DOWNTOWN USC, CA US	2020-12-27	1.12	NaN	1.0	NaN	66	55	270.0	260.0	8.9	18.1	1.0	NaN	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1820	USW00093134	LOS ANGELES DOWNTOWN USC, CA US	2020-12-25	1.57	NaN	0.0	NaN	75	52	270.0	270.0	8.9	14.1	NaN	NaN	NaN
1821	USW00093134	LOS ANGELES DOWNTOWN USC, CA US	2020-12-26	0.67	NaN	0.0	NaN	69	46	260.0	250.0	6.9	12.1	NaN	NaN	NaN
1824	USW00093134	LOS ANGELES DOWNTOWN USC, CA US	2020-12-29	1.57	NaN	0.0	NaN	65	42	340.0	360.0	10.1	18.1	NaN	NaN	NaN
1825	USW00093134	LOS ANGELES DOWNTOWN USC, CA US	2020-12-30	0.45	NaN	0.0	NaN	69	44	260.0	260.0	6.9	12.1	NaN	NaN	NaN
1826	USW00093134	LOS ANGELES DOWNTOWN USC, CA US	2020-12-31	1.57	NaN	0.0	NaN	70	43	350.0	350.0	12.1	19.9	NaN	NaN	NaN

```
#dropping features with many null values
df=df[df.columns[df.isnull().mean() < 0.01]]
```

```
#converting the rest of the null values with mode
df['AWND'] = df['AWND'].fillna(df['AWND'].mode()[0])
df['PRCP'] = df['PRCP'].fillna(df['PRCP'].mode()[0])
df['TMAX'] = df['TMAX'].fillna(df['TMAX'].mode()[0])
df['TMIN'] = df['TMIN'].fillna(df['TMIN'].mode()[0])
df['WDF2'] = df['WDF2'].fillna(df['WDF2'].mode()[0])
df['WDF5'] = df['WDF5'].fillna(df['WDF5'].mode()[0])
df['WSF2'] = df['WSF2'].fillna(df['WSF2'].mode()[0])
df['WSF5'] = df['WSF5'].fillna(df['WSF5'].mode()[0])
```

```
#normalizing data
import numpy as np
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df['STATION'] = label_encoder.fit_transform(df['STATION'])
df['NAME'] = label_encoder.fit_transform(df['NAME'])
df['DATE'] = label_encoder.fit_transform(df['DATE'])
df.head()
```

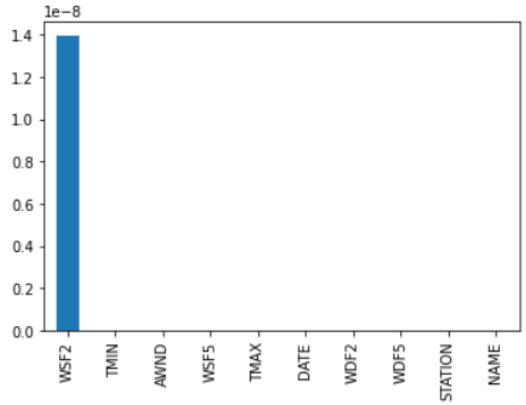
	STATION	NAME	DATE	AWND	PRCP	TMAX	TMIN	WDF2	WDF5	WSF2	WSF5
0	0	0	0	2.46	0.0	64	43	10.0	30.0	8.1	11.0
1	0	0	1	2.01	0.0	65	47	270.0	30.0	6.0	8.9
2	0	0	2	0.67	0.0	62	44	150.0	150.0	10.1	14.1
3	0	0	3	1.34	1.0	69	55	270.0	280.0	8.1	14.1
4	0	0	4	2.46	1.0	59	49	140.0	140.0	10.1	16.1

```
from sklearn.feature_selection import chi2

X = df.drop('PRCP',axis=1)
y = df['PRCP']
scores = chi2(X,y)
scores
```

```
(array([          nan,           nan,  845.54850795,   79.05899558,
        318.33894152,   54.00341786, 2310.12947627, 1828.05882977,
         32.19213439,   83.73695329]),
 array([           nan,            nan, 6.75045568e-186, 6.02826182e-019,
        3.33210470e-071, 2.00141159e-013, 0.00000000e+000, 0.00000000e+000,
        1.39654983e-008, 5.65188246e-020]))
```

```
p_values = pd.Series(scores[1],index = X.columns)
p_values.sort_values(ascending = False , inplace = True)
p_values.plot.bar()
```

```
<AxesSubplot:>
```



```
#Splitting data into test and train datasets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

```
# make class predictions for the testing set
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(max_iter=3000).fit(X_train,y_train)
y_pred_class = logreg.predict(X_test)
```

```
# calculate accuracy
from sklearn import metrics
print(metrics.accuracy_score(y_test, y_pred_class))
```

```
0.936542669584245
```

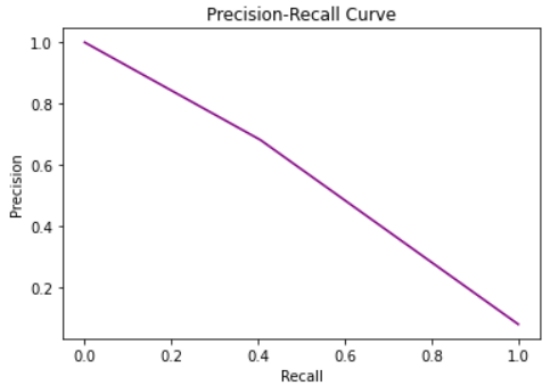
```
# calculate precision
print(metrics.precision_score(y_test, y_pred_class))
```

```
0.6818181818181818
```

```
# calculate recall
print(metrics.recall_score(y_test, y_pred_class))
```

```
0.40540540540540543
```

```
from sklearn.metrics import precision_recall_curve
#calculate precision and recall
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_class)

#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

#add axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

#display plot
plt.show()
```



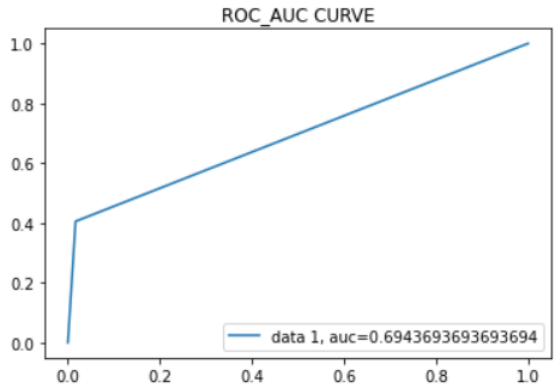
```
# calculate F-1 score
print(metrics.f1_score(y_test, y_pred_class))
```

```
0.5084745762711864
```

```
# calculate ROC_AUC
print(metrics.roc_auc_score(y_test, y_pred_class))
```

```
0.6943693693693694
```

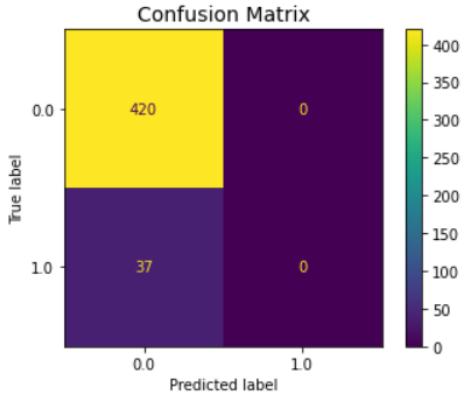
```
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_class)
auc = metrics.roc_auc_score(y_test, y_pred_class)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.title('ROC_AUC CURVE', fontsize=12)
plt.show()
```




```
print(metrics.confusion_matrix(y_test, y_pred_class))
```

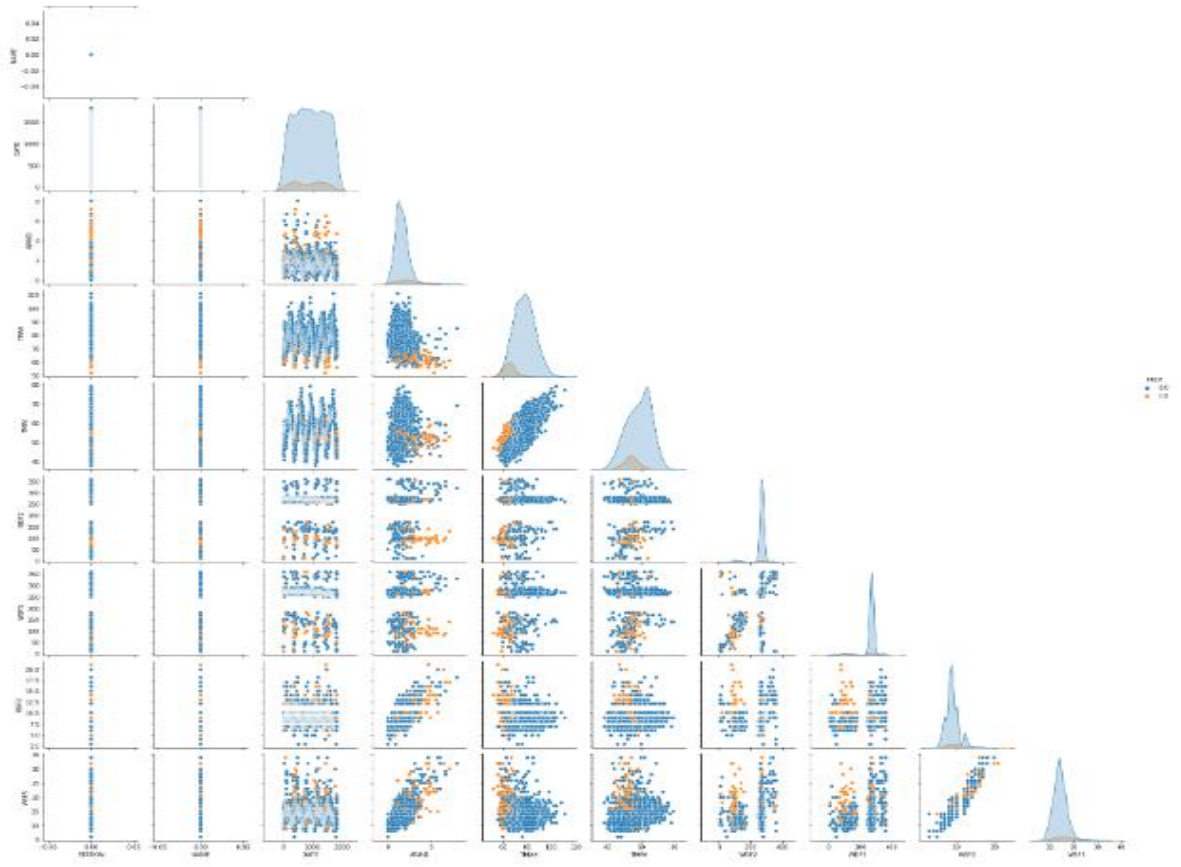
```
[[413   7]
 [ 22  15]]
```

```
from sklearn.datasets import make_classification
from sklearn.metrics import plot_confusion_matrix
from sklearn.svm import SVC
X, y = make_classification(random_state=0)
clf = SVC(random_state=0)
clf.fit(X_train, y_train)
SVC(random_state=0)
plot_confusion_matrix(clf, X_test, y_test)
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```



```
#visualizing using seaborn
import seaborn as sns

sns.pairplot(df.iloc[:, :], hue='PRCP', corner=True)
```

```
<seaborn.axisgrid.PairGrid at 0x253831c7ca0>
```



Step 3: Business Modelling

□ Freemium Model:

- **Basic Access:** Provide a free tier offering essential precipitation forecasts and general weather insights. This attracts a broad user base and introduces them to the system.
- **Premium Features:** Offer advanced functionalities such as detailed weather analytics, personalized alerts, and historical data insights through a subscription model. Premium users could also receive additional support and consultancy services.

 Subscription-Based Model:

- **Monthly/Annual Subscriptions:** Charge users a recurring fee for access to comprehensive features. Different tiers could offer varying levels of detail and additional services like expert consultations and customized forecasts.

 Pay-Per-Use Model:

- **On-Demand Reports:** Allow farmers to pay for individual reports or detailed weather forecasts on an as-needed basis. This model can be attractive for users who prefer to pay only when they need specific insights.

 Advertising and Partnerships:

- **In-App Advertising:** Integrate relevant advertisements from agricultural suppliers, equipment manufacturers, or local service providers. Ensure ads are non-intrusive and relevant to the farmers' needs.

- **Partnerships:** Collaborate with agricultural organizations, local governments, or NGOs that focus on farming support. These partners can subsidize costs for farmers or offer the system as part of a larger support package.

 Data and Insights Sales:

  Aggregated Data: Sell anonymized, aggregated weather data and insights to research institutions, agricultural agencies, or commercial entities interested in market trends and agricultural patterns.

Step 4: Financial Modelling

1. **Time Series/Regression Forecasting**: Since we're focused on predicting precipitation patterns (for an AI-powered precipitation prediction system), we will perform time series forecasting using the **PRCP** (precipitation) column. We'll use regression or time series models like ARIMA to forecast future precipitation trends.

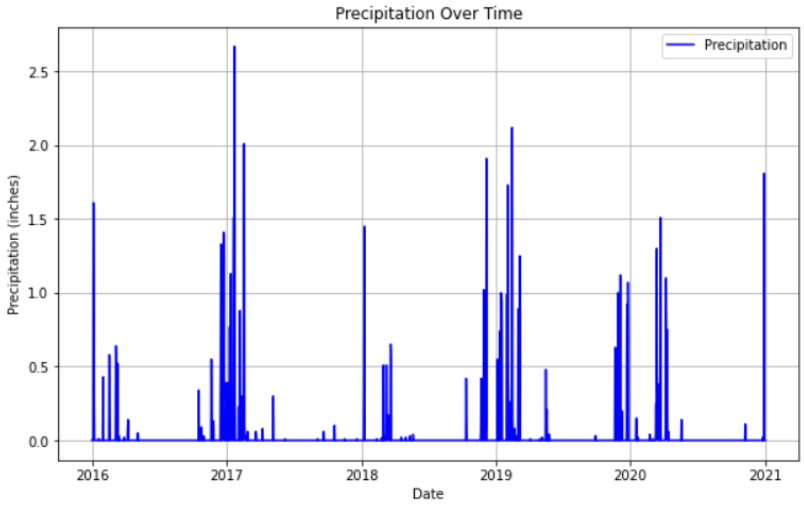
   Let's clean the data, ensuring the DATE column is properly formatted, and then build a time series model.

```
data1=pd.read_csv('dataset.csv')
# Convert the DATE column to datetime for time series analysis
data1['DATE'] = pd.to_datetime(data1['DATE'])

# Sort the data by date for proper time series analysis
data1 = data1.sort_values('DATE')

# Since we are focusing on precipitation (PRCP), let's check if there are any missing values in PRCP
prcp_missing = data1['PRCP'].isna().sum()

# Display the number of missing values in PRCP
prcp_missing
```

```
0
```

There are no missing values in the **PRCP** (precipitation) column, which means the data is ready for time series forecasting. Let's proceed by visualizing the precipitation trend over time and then applying a time series forecasting model like ARIMA to predict future precipitation.

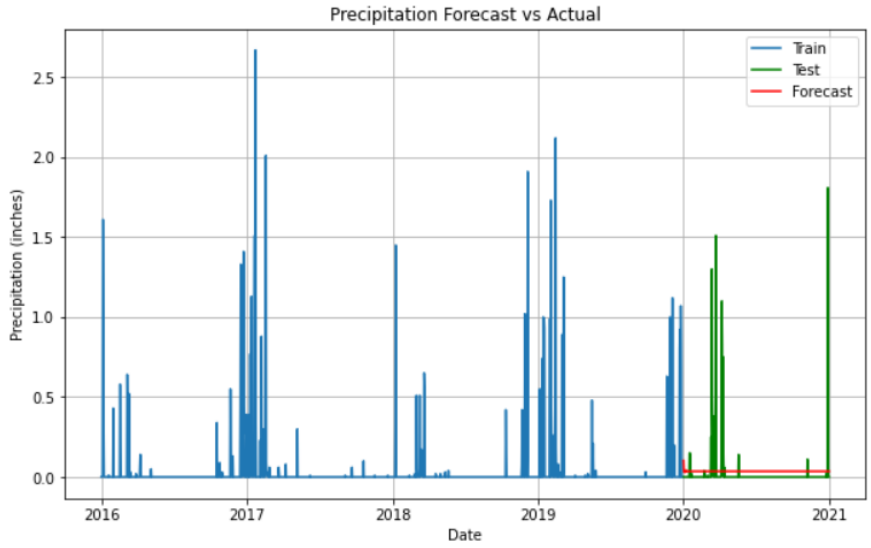
```
# Plotting the time series of precipitation
plt.figure(figsize=(10,6))
plt.plot(data1['DATE'], data1['PRCP'], color='blue', label='Precipitation')
plt.title('Precipitation Over Time')
plt.xlabel('Date')
plt.ylabel('Precipitation (inches)')
plt.legend()
plt.grid(True)
plt.show()
```



The plot above shows the trend of precipitation over time. With this visual, we can see the fluctuations in precipitation levels.

Next, we will apply a time series model such as ARIMA (AutoRegressive Integrated Moving Average) to predict future precipitation values based on the historical data.

```python
from statsmodels.tsa.arima.model import ARIMA

# Selecting the precipitation column (PRCP) for time series analysis
prcp_series = data1.set_index('DATE')['PRCP']

# Splitting data into training and test sets (80% train, 20% test)
train, test = train_test_split(prcp_series, test_size=0.2, shuffle=False)

# Building the ARIMA model on the training data
model = ARIMA(train, order=(5, 1, 0))  # ARIMA(p,d,q) with (5,1,0) as initial parameters
arima_model = model.fit()

# Forecasting future values on the test set
forecast = arima_model.forecast(steps=len(test))

# Plotting the actual vs forecasted values
plt.figure(figsize=(10,6))
plt.plot(train.index, train, label='Train')
plt.plot(test.index, test, label='Test', color='green')
plt.plot(test.index, forecast, label='Forecast', color='red')
plt.title('Precipitation Forecast vs Actual')
plt.xlabel('Date')
plt.ylabel('Precipitation (inches)')
plt.legend()
plt.grid(True)
plt.show()
```



2. Financial Equation:

To design a **financial equation** for our AI-powered precipitation prediction system targeting small-scale farmers, let's break it down based on the example and adapt it to our product.

Variables:

- **Product Unit Cost (P):** The price we charge per unit of your product/service.
- **Total Number of Sales (x):** The number of units sold in a given time period (e.g., month, quarter, or year).

- **Fixed Costs (C):** The cost to run the business (e.g., salaries, infrastructure, etc.) that does not depend on the number of units sold.

The basic financial equation becomes: Total Revenue (y) = P*x - C

i. Example:

- **Assume Product Unit Cost (P):** ₹ 500 per farmer subscription per month.
- **Fixed Costs (C):** ₹ 10,000 per month (this includes infrastructure costs, development costs, maintenance, etc.).
- **Assume 300 farmers (x) subscribe in a given month.**

  Now, the equation for that month will be:

  Revenue (y) = 500×300−10,000 = ₹ 1,40,000

ii. General Equation:

For general forecasting purposes:

$$y = 500x - 10,000$$

Where:

- **x:** Number of sales/subscriptions.
- **500:** Price per subscription.
- **10,000:** Monthly operating cost.

iii. Incorporating Market Growth:

If the market grows and we expect an increase in subscribers, say we anticipate a 10% monthly growth in subscribers, the financial equation can incorporate this trend over time.

Let:

- x_0**:** Initial number of subscribers (e.g., 300).
- **g:** Monthly growth rate (e.g., 10%, or 0.1).

For month **t**:

$$x(t) = x_0 \times (1+g)^t$$

We can now substitute this in the revenue equation for **t** months:

$$y(t) = P \times x_0 \times (1+g)^t - C$$

iv. Profit:

Once we compute the revenue, we can further calculate profit by subtracting additional operational costs or taxes, depending on your specific situation.

Conclusion:

This equation allows us to track total revenue over time and adjust based on market trends and sales.

GitHub link: https://github.com/Tushar264/AI-Powered-Precipitation-Prediction-System