```
1
```

```
1 import pandas as pd
2
```

```
1 data = pd.read_csv("/content/weatherHistory (1).csv")
```

```
1 data
```

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout the day.
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	Partly cloudy throughout the day.
...	...	...	...	...	...	...	...	...	...	...	...	...
96448	2016-09-09 19:00:00.000 +0200	Partly Cloudy	rain	26.016667	26.016667	0.43	10.9963	31.0	16.1000	0.0	1014.36	Partly cloudy starting in the morning.
96449	2016-09-09 20:00:00.000 +0200	Partly Cloudy	rain	24.583333	24.583333	0.48	10.0947	20.0	15.5526	0.0	1015.16	Partly cloudy starting in the morning.
96450	2016-09-09 21:00:00.000 +0200	Partly Cloudy	rain	22.038889	22.038889	0.56	8.9838	30.0	16.1000	0.0	1015.66	Partly cloudy starting in the

```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96453 entries, 0 to 96452
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Formatted Date            96453 non-null  object
 1   Summary                   96453 non-null  object
 2   Precip Type               95936 non-null  object
 3   Temperature (C)           96453 non-null  float64
 4   Apparent Temperature (C)  96453 non-null  float64
 5   Humidity                  96453 non-null  float64
 6   Wind Speed (km/h)         96453 non-null  float64
 7   Wind Bearing (degrees)    96453 non-null  float64
 8   Visibility (km)           96453 non-null  float64
 9   Loud Cover                96453 non-null  float64
 10  Pressure (millibars)      96453 non-null  float64
 11  Daily Summary             96453 non-null  object
dtypes: float64(8), object(4)
memory usage: 8.8+ MB
```

```
1 data.describe()
```

	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)
count	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.0	96453.000000
mean	11.932678	10.855029	0.734899	10.810640	187.509232	10.347325	0.0	1003.235956
std	9.551546	10.696847	0.195473	6.913571	107.383428	4.192123	0.0	116.969906
min	-21.822222	-27.716667	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
25%	4.688889	2.311111	0.600000	5.828200	116.000000	8.339800	0.0	1011.900000
50%	12.000000	12.000000	0.780000	9.965900	180.000000	10.046400	0.0	1016.450000
75%	18.838889	18.838889	0.890000	14.135800	290.000000	14.812000	0.0	1021.090000
max	39.905556	39.344444	1.000000	63.852600	359.000000	16.100000	0.0	1046.380000

```
1 data.isnull().sum()
```

	0
Formatted Date	0
Summary	0
Precip Type	517
Temperature (C)	0
Apparent Temperature (C)	0
Humidity	0
Wind Speed (km/h)	0
Wind Bearing (degrees)	0
Visibility (km)	0
Loud Cover	0
Pressure (millibars)	0
Daily Summary	0

dtype: int64

```
1 data.duplicated().sum()
```

```
24
```

```
1 data.drop_duplicates(inplace = True)
```

```
1 data['Precip Type'].fillna(data['Precip Type'].mode()[0], inplace = True)
```

```
<ipython-input-10-282e4c7ce01b>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting val

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].m

  data['Precip Type'].fillna(data['Precip Type'].mode()[0], inplace = True)
```

```
1 X_class = data.drop(['Precip Type'], axis = 1)
2 y_class = data['Precip Type']
```

```
1 X_class.drop(['Daily Summary'], axis = 1, inplace = True)
```

```
1 X_class.drop(['Formatted Date'], axis = 1, inplace = True)
```

```
1 X_class.drop(['Summary'], axis = 1, inplace = True)
```

```
X_class.drop(['Summary'], axis=1, inplace=True)
```

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
y_class = le.fit_transform(y_class)

```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_scaled_class = scaler.fit_transform(X_class)

```

```
from sklearn.model_selection import train_test_split
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(X_scaled_class, y_class, test_

```

logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train_class, y_train_class)
y_pred_logreg = logreg.predict(X_test_class)
print("Logistic Regression Accuracy:", accuracy_score(y_test_class, y_pred_logreg))
print(classification_report(y_test_class, y_pred_logreg))

```

```
Logistic Regression Accuracy: 0.9892668256766567
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     17087
           1       0.95      0.95      0.95      2199

    accuracy                           0.99     19286
   macro avg       0.97      0.97      0.97     19286
weighted avg       0.99      0.99      0.99     19286
```

svc

```
from sklearn.svm import SVC
svc = SVC()
svc.fit(X_train_class, y_train_class)
y_pred_svc = svc.predict(X_test_class)
print("\nSVC Accuracy:", accuracy_score(y_test_class, y_pred_svc))
print(classification_report(y_test_class, y_pred_svc))

```

```
SVC Accuracy: 0.9905631027688478
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     17087
           1       0.95      0.97      0.96      2199

    accuracy                           0.99     19286
   macro avg       0.97      0.98      0.98     19286
weighted avg       0.99      0.99      0.99     19286
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train_class, y_train_class)
y_pred_dtc = dtc.predict(X_test_class)
print("\nDecision Tree Accuracy:", accuracy_score(y_test_class, y_pred_dtc))
print(classification_report(y_test_class, y_pred_dtc))

```

```
Decision Tree Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17087
           1       1.00      1.00      1.00      2199

    accuracy                           1.00     19286
   macro avg       1.00      1.00      1.00     19286
weighted avg       1.00      1.00      1.00     19286
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train_class, y_train_class)
y_pred_knn = knn.predict(X_test_class)
print("\nKNN Accuracy:", accuracy_score(y_test_class, y_pred_knn))
print(classification_report(y_test_class, y_pred_knn))

```

```
KNN Accuracy: 0.9829409934667634
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     17087
           1       0.94      0.91      0.92      2199

    accuracy                           0.98     19286
   macro avg       0.96      0.95      0.96     19286
weighted avg       0.98      0.98      0.98     19286
```

```
print("Logistic Regression Accuracy:", accuracy_score(y_test_class, y_pred_logreg))
print("Logistic Regression Classification Report:\n", classification_report(y_test_class, y_pred_logreg))

print("\nSVC Accuracy:", accuracy_score(y_test_class, y_pred_svc))
print("SVC Classification Report:\n", classification_report(y_test_class, y_pred_svc))

print("\nDecision Tree Accuracy:", accuracy_score(y_test_class, y_pred_dtc))
print("Decision Tree Classification Report:\n", classification_report(y_test_class, y_pred_dtc))

print("\nKNN Accuracy:", accuracy_score(y_test_class, y_pred_knn))
print("KNN Classification Report:\n", classification_report(y_test_class, y_pred_knn))

```

```
Logistic Regression Accuracy: 0.9892668256766567
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     17087
           1       0.95      0.95      0.95      2199

    accuracy                           0.99     19286
   macro avg       0.97      0.97      0.97     19286
weighted avg       0.99      0.99      0.99     19286
```

```
SVC Accuracy: 0.9905631027688478
SVC Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     17087
           1       0.95      0.97      0.96      2199

    accuracy                           0.99     19286
   macro avg       0.97      0.98      0.98     19286
weighted avg       0.99      0.99      0.99     19286


Decision Tree Accuracy: 1.0
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17087
           1       1.00      1.00      1.00      2199

    accuracy                           1.00     19286
   macro avg       1.00      1.00      1.00     19286
weighted avg       1.00      1.00      1.00     19286


KNN Accuracy: 0.9829409934667634
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     17087
           1       0.94      0.91      0.92      2199

    accuracy                           0.98     19286
   macro avg       0.96      0.95      0.96     19286
weighted avg       0.98      0.98      0.98     19286
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
X = data.drop(['Temperature (C)', 'Formatted Date', 'Summary', 'Daily Summary'], axis=1)
y = data['Temperature (C)']
```

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
le = LabelEncoder()
X['Precip Type'] = le.fit_transform(X['Precip Type'])
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 0.908876900664492
R-squared: 0.9900880796738758
```

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.svm import SVR
linear_regressor = LinearRegression()
linear_regressor.fit(X_train, y_train)
y_pred_linear = linear_regressor.predict(X_test)
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)
print("Linear Regression:")
print("Mean Squared Error:", mse_linear)
print("R-squared:", r2_linear)
```

```
Linear Regression:
Mean Squared Error: 0.908876900664492
R-squared: 0.9900880796738758
```

```
poly_features = PolynomialFeatures(degree=2)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)
poly_regressor = LinearRegression()
poly_regressor.fit(X_train_poly, y_train)
y_pred_poly = poly_regressor.predict(X_test_poly)
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)
print("\nPolynomial Regression:")
print("Mean Squared Error:", mse_poly)
print("R-squared:", r2_poly)
```

```
Polynomial Regression:
Mean Squared Error: 0.277613325683468
R-squared: 0.9969724380016335
```

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
svr_regressor = SVR(kernel='rbf', C=1.0, epsilon=0.0)
svr_regressor.fit(X_train, y_train)
y_pred_svr = svr_regressor.predict(X_test)
mse_svr = mean_squared_error(y_test, y_pred_svr)
r2_svr = r2_score(y_test, y_pred_svr)
print("\nSVR with RBF Kernel (No Epsilon):")
print("Mean Squared Error:", mse_svr)
print("R-squared:", r2_svr)

```

…

```
y
```

…

2222

```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, VotingClassifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train_class, y_train_class)
y_pred_rf = rf_classifier.predict(X_test_class)
print("Random Forest Accuracy:", accuracy_score(y_test_class, y_pred_rf))
print("Random Forest F1 Score:", classification_report(y_test_class, y_pred_rf))

ada_classifier = AdaBoostClassifier(n_estimators=50, random_state=42)
ada_classifier.fit(X_train_class, y_train_class)
y_pred_ada = ada_classifier.predict(X_test_class)
print("\nAdaBoost Accuracy:", accuracy_score(y_test_class, y_pred_ada))
print("AdaBoost F1 Score:", classification_report(y_test_class, y_pred_ada))

voting_classifier = VotingClassifier(estimators=[
    ('lr', logreg),
    ('dt', dtc),
    ('knn', knn)],
    voting='hard')

voting_classifier.fit(X_train_class, y_train_class)
y_pred_voting = voting_classifier.predict(X_test_class)
print("\nVoting Classifier Accuracy:", accuracy_score(y_test_class, y_pred_voting))
print("Voting Classifier F1 Score:", classification_report(y_test_class, y_pred_voting))

```

…

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```