TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM

KHOA TOÁN – TIN HỌC

….📖….

BÁO CÁO CUỐI KỲ

Môn Nhập môn Khoa học dữ liệu – MTH10171

Học kỳ II (2021 - 2022)

TÌM HIỂU, XÂY DỰNG VÀ KIỂM THỬ CÁC MÔ HÌNH MÁY HỌC TRONG LĨNH VỰC KHOA HỌC DỮ LIỆU

Tên đề tài: Xây dựng mô hình dự đoán thời tiết – Weather Prediction.

Nhóm: 15

Tên thành viên:

Hồ Ngọc Ân – 20280001.

Trần Tuấn Thái – 20280082.

Huỳnh Quang Trung – 20280108.

Hỏa Ngọc Tú – 20280111.

Giảng viên hướng dẫn: ThS. Hà Văn Thảo.

TP. Hồ Chí Minh, tháng 06 năm 2022

MỤC LỤC

MỤC LỤC HÌNH ẢNH

MỞ ĐẦU

Học phần Nhập môn Khoa học dữ liệu – MTH10171 nhằm cung cấp cho sinh viên các kiến thức cơ bản trong lĩnh vực Khoa học dữ liệu, một lĩnh vực liên ngành về các phương pháp, các quá trình, và các hệ thống có khả năng học và phát hiện tri thức từ lượng dữ liệu ban đầu. Các phương pháp và mô hình trong Khoa học dữ liệu sẽ giúp con người, máy tính đưa ra các quyết định và phán đoán tốt hơn trong thực tế. Trong môn học này, các phương pháp, mô hình từ Học máy (Machine Learning), Khai phá dữ liệu (Data Mining), và Thống kê (Statistics) cũng sẽ được giới thiệu nhằm cung cấp cho sinh viên những kiến thức, kỹ năng cơ bản trong lĩnh vực khoa học dữ liệu.

Đề tài kết thúc môn học lần này nhằm giúp sinh viên nắm được và vận dụng các bước chính khi đào tạo một mô hình học máy bao gồm phân tích dữ liệu, tạo giả thuyết, lấy dữ liệu, tiền xử lý, phân tích, đánh giá chất lượng, đưa ra các phán đoán. Với đề tài "Xây dựng mô hình dự đoán thời tiết – Weather Prediction" nhóm chúng em mong rằng sẽ mang đến được những kết quả tốt nhất có thể để chia sẻ những thông tin hữu ích, đóng góp vào bộ tài nguyên cho ngành học của chúng ta thêm phong phú hơn, làm nguồn tham khảo cho tất cả mọi người.

Nhóm sinh viên thực hiện đề tài.

NỘI DUNG

1. GIỚI THIỆU.

1.1. Giới thiệu chung về đề tài.

Trong bài báo cáo này, nhóm sẽ sử dụng bộ dữ liệu "seattle-weather.csv" từ Kaggle để dùng trong việc phân tích và xây dựng các mô hình dự đoán tình trạng thời tiết dựa trên các điều kiện. Đây là một bộ dữ liệu bao gồm các điều kiện thời tiết dựa trên các mẫu và tình trạng thời tiết gắn với các điều kiện đó. Liên kết đến bộ dữ liệu: https://www.kaggle.com/datasets/ananthr1/weather-prediction. Đây là bộ dữ liệu thời tiết tại bang Seattle Hoa Kỳ được ghi nhận theo từng ngày trong khoảng 4 năm.

Các mô hình được sử dụng trong bài báo cáo: Logistic Regression, Decision Tree, K-Nearest Neighbor Classifier (KNN).

1.2. Mục tiêu của đề tài.

Sau khi thực hiện xong đề tài, sinh viên biết được quy trình, cách thức để xây dựng được một mô hình Machine Learning để có thể ứng dụng vào giải quyết các bài toán phù hợp.

Rèn luyện thêm kỹ năng làm sạch và phân tích dữ liệu, tiền xử lý và hiểu được bộ dữ liệu trước khi tiến hành đào tạo một mô hình.

Bài báo cáo được hoàn thành dựa trên quan điểm tăng độ chính xác của kết quả bài toán nhờ vào việc xây dựng nhiều mô hình Machine Learning khác nhau.

2. NỘI DUNG THỰC HIỆN.

2.1. Các thư viện cần thiết trong quá trình xử lý bài toán.

Việc gọi các thư viện cần sử dụng đến trong quá trình xử lý dữ liệu và đào tạo mô hình là một bước cực kỳ quan trọng cần phải được thực hiện đầu tiên trước khi xử lý các vấn đề khác. Ở bài toán này, chúng ta sẽ cần đến các thư viện

tiêu biểu như: Numpy, Pandas, Matplotlib… và đặc biệt là thư viện Sklearn chứa các mô hình mà chúng ta sẽ sử dụng đến.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
import re
import itertools
import missingno as mso
from scipy import stats
from scipy.stats import ttest_ind
from scipy.stats import pearsonr
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,c
lassification_report
```

2.2. Tìm hiểu và khám phá bộ dữ liệu.

2.2.1. Thông tin về bộ dữ liệu.

Để đọc bộ dữ liệu "seattle-weather.csv" vào không gian làm việc trong Notebook, chúng ta sẽ sử dụng thư viện pandas bằng cách gọi hàm read csv(). Bộ dữ liệu đã được đọc vào bằng URL giúp linh động trong việc thực hiện giữa các người dùng khác nhau mà không cần bận tâm đến việc phải có file mềm của dữ liệu.

Input[1]

```
# Đọc bộ dữ liệu ở dạng RAW thông qua URL với thư viện pandas
data=pd.read_csv("https://raw.githubusercontent.com/trunghq0205
/IntroductionToDataScience_MTH10171/main/seattle-weather.csv")
data.head()
```

Output[1]:

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	2012-01-02	10.9	10.6	2.8	4.5	rain
2	2012-01-03	0.8	11.7	7.2	2.3	rain
3	2012-01-04	20.3	12.2	5.6	4.7	rain
4	2012-01-05	1.3	8.9	2.8	6.1	rain

Input[2]:

```
data.shape
```

Output[2]:

```
(1461, 6)
```

Quan sát bộ dữ liệu thu thập được ở trên, ta có thể thấy có 6 cột với tổng cộng 1461 dòng theo các quan sát trong tập dữ liệu. Trong đó, 4 biến liên tục bao gồm: “precipitation”, “temp_max”, “temp_min”, “wind”; một biến ghi nhận thông tin ngày: “date” có dạng YYYY-MM-DD; một biến chỉ tình trạng thời tiết “weather”.

Giải thích rõ ràng hơn về ý nghĩa của các biến: biến “precipitation” chỉ thông tin lượng mưa của tất cả các dạng nước rơi xuống mặt đất như mưa, mưa đá, tuyết rơi hoặc mưa phùn; biến “temp_max” chỉ nhiệt độ cao nhất trong ngày; biến “temp_min” chỉ nhiệt độ thấp nhất trong ngày; biến “wind” lưu thông tin tốc độ gió trong ngày; biến “weather” là biến chỉ tình trạng thời tiết với các điều kiện mẫu.

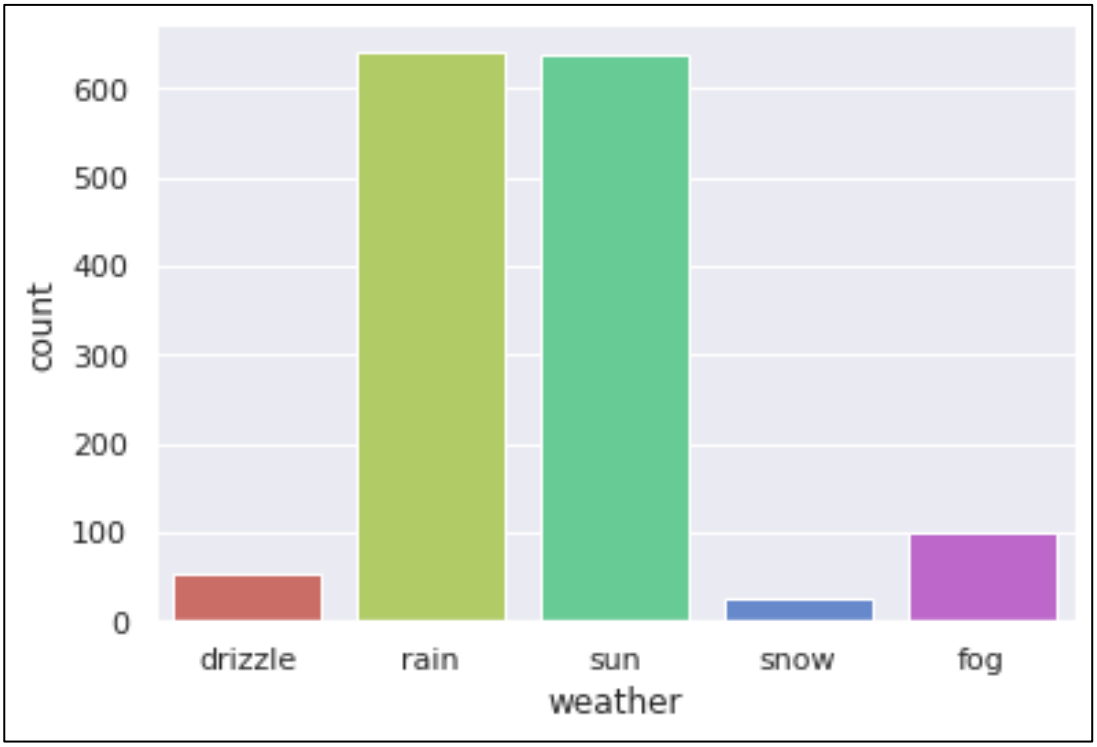
2.2.2. Khám phá các biến dữ liệu.

Ở bước này, chúng ta sẽ tiến hành phân tích các biến trong bộ dữ liệu mà chúng ta đã thu được phía trên. Trước tiên, chúng ta sẽ bắt đầu từ các biến phân loại.

Input[3]:

```
import warnings
warnings.filterwarnings('ignore')
sns.countplot("weather",data=data,palette="hls");
```

Output[3]:



Hình 1. Số lượng các tình trạng thời tiết có trong biến date.

Input[4]:

```
countrain=len(data[data.weather=="rain"])
countsun=len(data[data.weather=="sun"])
countdrizzle=len(data[data.weather=="drizzle"])
countsnow=len(data[data.weather=="snow"])
countfog=len(data[data.weather=="fog"])
print("Percent of Rain:{:2f}%".format((countrain/(len(data
.weather))*100)))
print("Percent of Sun:{:2f}%".format((countsun/(len(data.w
eather))*100)))
print("Percent of Drizzle:{:2f}%".format((countdrizzle/(le
n(data.weather))*100)))
print("Percent of Snow:{:2f}%".format((countsnow/(len(data
.weather))*100)))
print("Percent of Fog:{:2f}%".format((countfog/(len(data.w
eather))*100)))
```

Output[4]:

```
Percent of Rain:43.874059%
Percent of Sun:43.805613%
Percent of Drizzle:3.627652%
Percent of Snow:1.779603%
Percent of Fog:6.913073%
```

Từ đồ thị và các phân tích trên, ta có thể thấy tập dữ liệu chứa phần lớn là tình trạng thời tiết rain và sun với hơn 600 dòng dữ liệu và xấp xỉ nhau khi chiếm 43.3% bộ dữ liệu. Đối với các tình trạng thời tiết như snow, fog và drizzle thì có khoảng dưới 100 dòng dữ liệu với dưới 10% bộ dữ liệu.

Nhận xét chung: Từ việc có ít dữ liệu đề cập đến các tình trạng snow, fog và drizzle thì có thể ảnh hưởng đến độ chính xác của mô hình khi dự đoán các tình trạng thời tiết như snow, fog và drizzle khi quá quá ít dữ liệu để đào tạo cho mô hình.

2.2.3. Tìm hiểu về phân phối của các biến liên tục.

Input[5]:

```
data[["precipitation","temp_max","temp_min","wind"]]
.describe()
```

Output[5]:

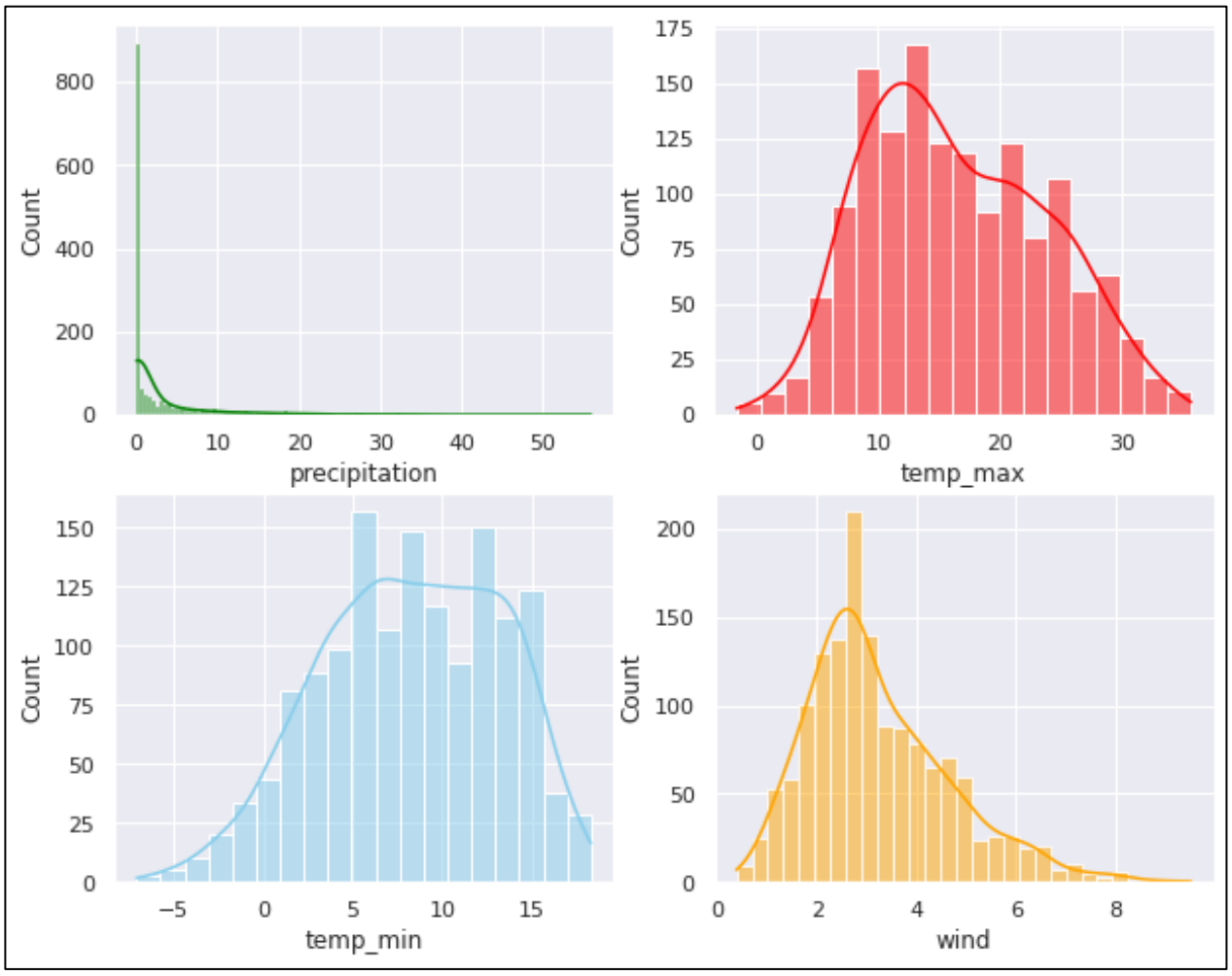
	precipitation	temp_max	temp_min	wind
count	1461.000000	1461.000000	1461.000000	1461.000000
mean	3.029432	16.439083	8.234771	3.241136
std	6.680194	7.349758	5.023004	1.437825
min	0.000000	-1.600000	-7.100000	0.400000
25%	0.000000	10.600000	4.400000	2.200000
50%	0.000000	15.600000	8.300000	3.000000
75%	2.800000	22.200000	12.200000	4.000000
max	55.900000	35.600000	18.300000	9.500000

Để xem và dự đoán được phân phối của các biến dữ liệu liên tục trên, ta sử dụng đồ thị Histogram với hàm vẽ histplot trong thư viện Seaborn.

Input[6]:

```
sns.set(style="darkgrid")
fig,axs=plt.subplots(2,2,figsize=(10,8))
sns.histplot(data=data,x="precipitation",kde=True,ax=axs[0,0],c
olor='green');
sns.histplot(data=data,x="temp_max",kde=True,ax=axs[0,1],color=
'red');
sns.histplot(data=data,x="temp_min",kde=True,ax=axs[1,0],color=
'skyblue');
sns.histplot(data=data,x="wind",kde=True,ax=axs[1,1],color='ora
nge');
```

Output[6]:



Hình 2. Đồ thị Histogram cho các biến precipitation, temp_max, temp_min và wind.

Từ các đồ thị phía trên, ta thấy rõ ràng phân phối của "precipitation", "wind" và có độ lệch tích cực (lệch phải), đuôi bên phải dài hơn đuôi bên trái. Phân phối của "temp_min" có độ lệch tiêu cực (lệch trái). Đối với phân phối của biến "temp_max" cũng có xu hướng lệch phải. Và cả bốn biến đều có một số giá trị ngoại lai (Outlier).

2.2.4. Xác định các điểm ngoại lai (Outlier) và tính lệch của các biến liên tục.

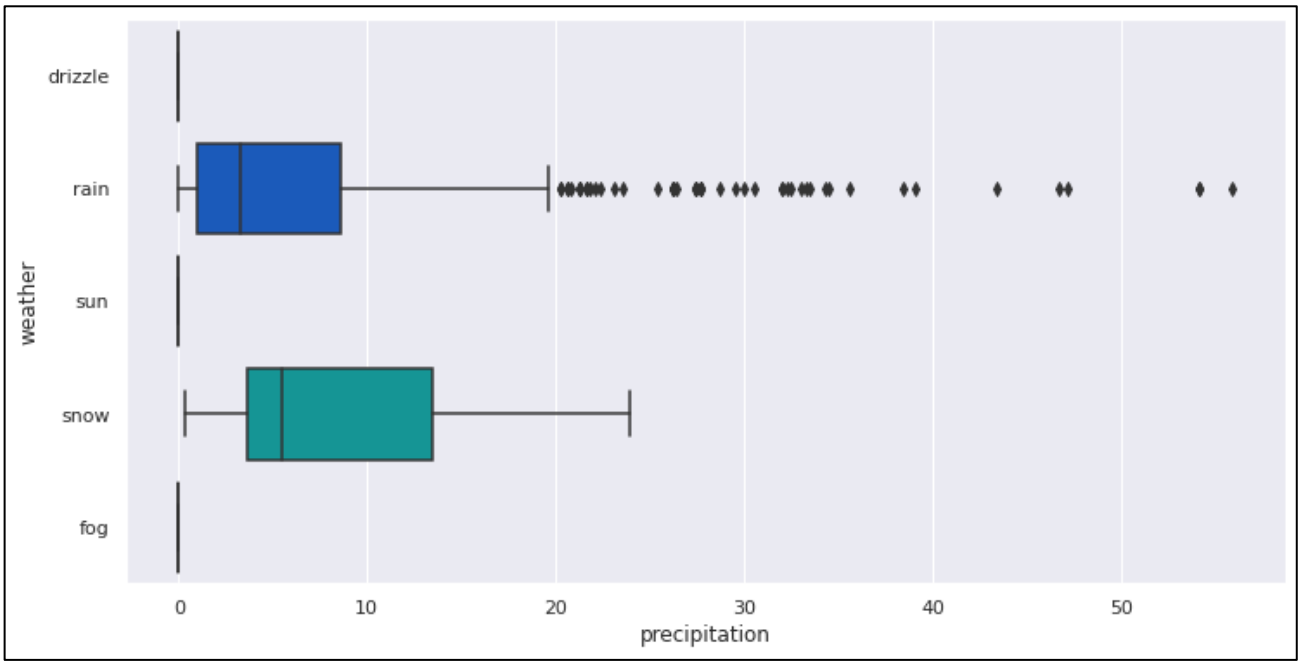
Nhận diện các điểm ngoại lai bằng đồ thị Boxplot là một phương pháp được sử dụng phổ biến dựa trên đặc điểm phân phối chuẩn dữ liệu trong phân tích thống kê. Ở bước này, ta sẽ sử dụng hàm boxplot trong thư viện Seaborn để vẽ đồ thị

histogram tương quan giữa các biến “precipitation”, “wind”, “temp_max”, “temp_min” đối với “weather”.

Input[7]:

```
plt.figure(figsize=(12,6))
sns.boxplot("precipitation","weather",data=data,palette="winter")
;
```

Output[7]:



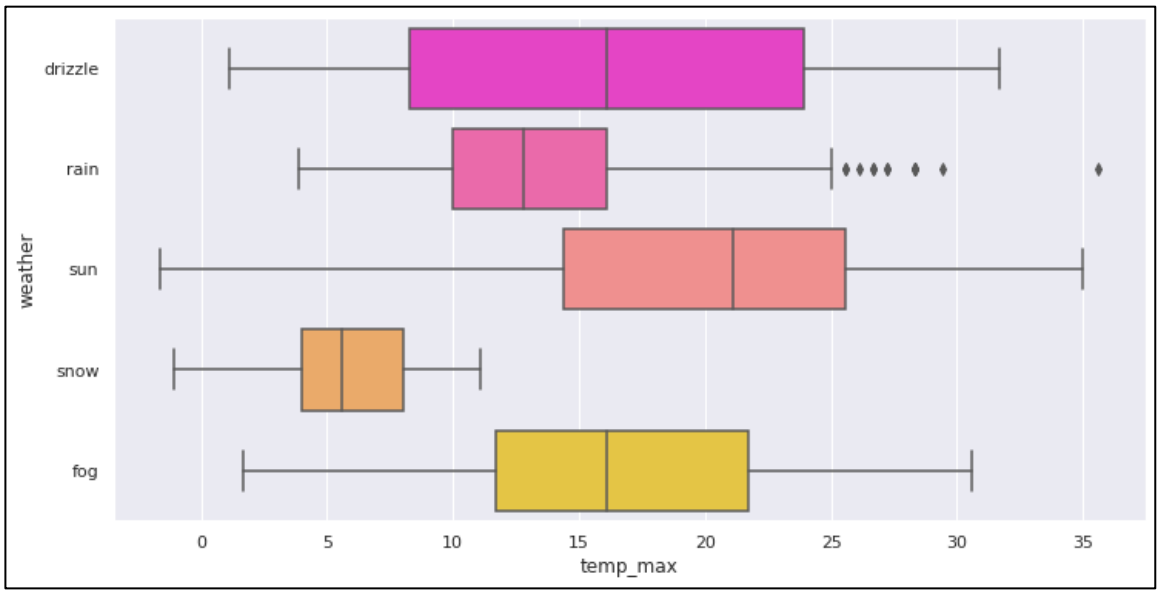
Hình 3. Đồ thị boxplot giữa biến và precipitation và weather.

Từ đồ thị boxplot giữa Weather và Precipitation phía trên, giá trị của Rain có nhiều dữ liệu ngoại lai dương và cả Rain và Snow đều bị lệch phải/có độ lệch dương.

Input[8]:

```
plt.figure(figsize=(12,6))
sns.boxplot("temp_max","weather",data=data,palette="spring");
```

Output[8]:

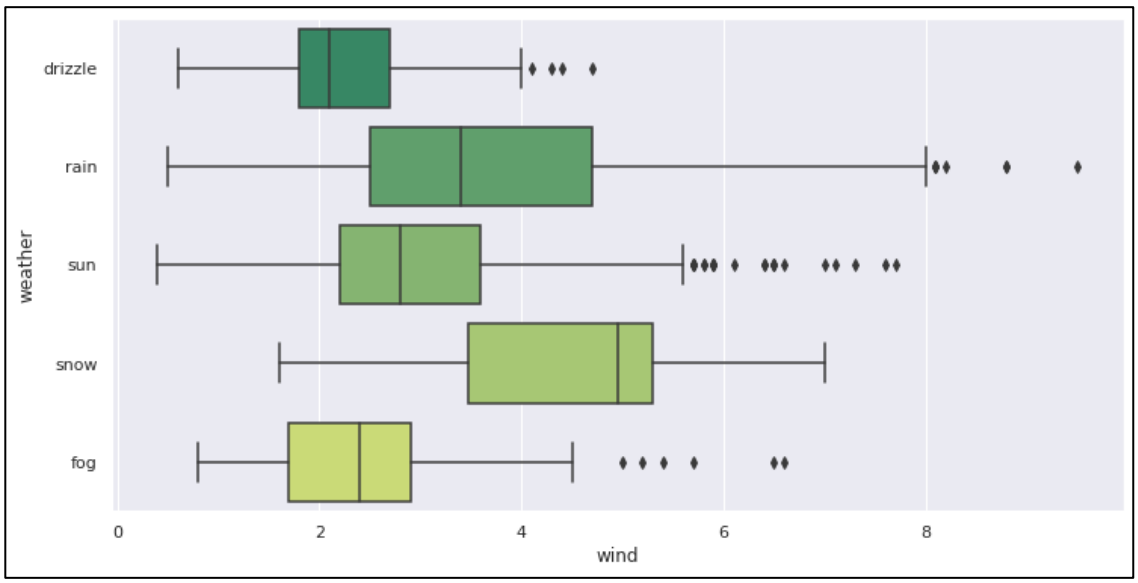


Hình 4. Đồ thị boxplot giữa biến temp_max và weather.

Input[9]:

```
plt.figure(figsize=(12,6))
sns.boxplot("wind","weather",data=data,palette="summer");
```

Output[9]:



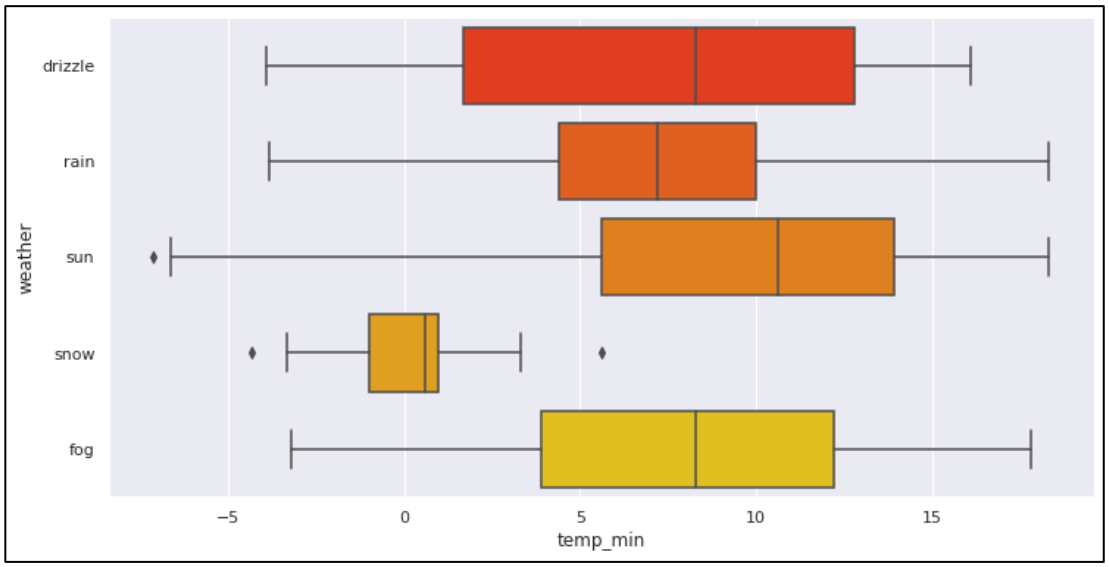
Hình 5. Đồ thị boxplot giữa biến wind và weather.

Từ các đồ thị boxplot phía trên, chúng ta thấy rằng mỗi thuộc tính của "weather" có một vài dữ liệu ngoại lai dương và bao gồm cả hai kiểu lệch trái và lệch phải.

Input[10]:

```
plt.figure(figsize=(12,6))
sns.boxplot("temp_min","weather",data=data,palette="aut
umn");
```

Output[10]:



Hình 6. Đồ thị boxplot giữa biến temp_min và weather.

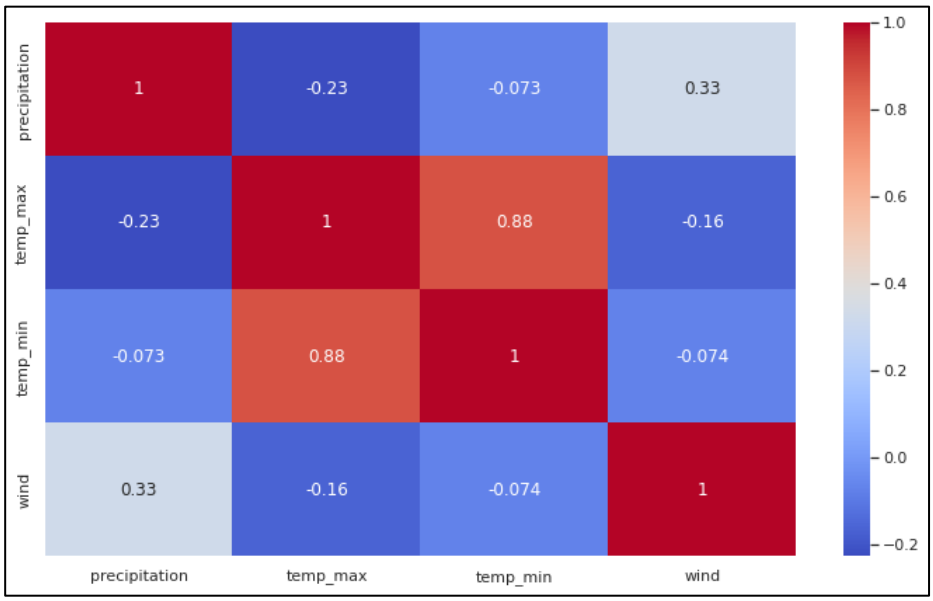
Quan sát được từ đồ thị boxplot giữa "weather" và "temp_min", một vài dữ liệu có giá trị âm và một vài dữ liệu có cả hai loại giá trị ngoại lai âm và dương, trong đó "snow" bị lệch trái.

2.2.5. Heatmap.

Input[11]:

```
plt.figure(figsize=(12,7))
sns.heatmap(data.corr(),annot=True,cmap='coolwarm');
```

Output[11]:



Hình 7. Heatmap.

Dựa vào Heatmap ta thấy có một mối tương quan đồng biến giữa "temp_max" và "temp_min".

2.2.6. Sự tương quan, ý nghĩa thống kê giữa các biến.

Để kiểm tra ý nghĩa thống kế giữa hai biến trong bộ dữ liệu, ta sử dụng hàm ttest_ind(), đây là hàm có sẵn trong thư viện scipy.stats(). Nó hỗ trợ thực kiểm kiểm định giả thuyết thống kê với giả thuyết H0 rằng hai mẫu độc lập có giá trị trung bình (kỳ vọng) giống hệt nhau và đối thuyết rằng giá trị trung bình là khác nhau (kiểm định 2 phía).

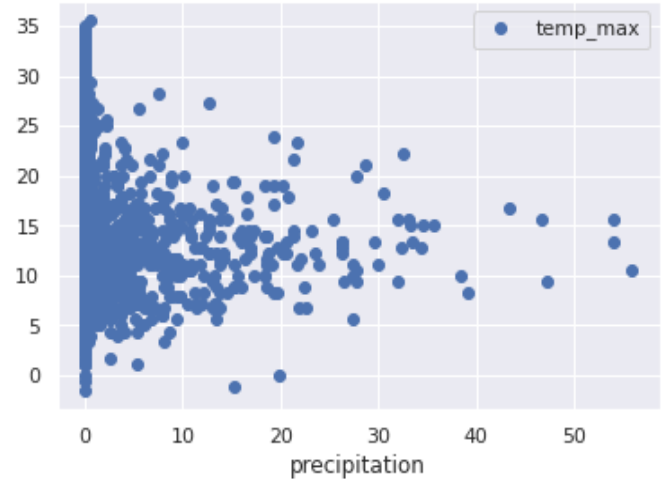
Nếu 2 biến đều có ý nghĩa thống kê, tức là chúng thực sự có trung bình khác nhau có nghĩa là sự hiện diện của nó trong bộ dữ liệu sẽ ảnh hưởng đến kết quả sau này. Ở đây chính là kết quả dự đoán tình trạng thời tiết dựa trên các biến này.

Input[12]:

```
data.plot("precipitation","temp_max",style='o')
print("Pearson correlation:",data["precipitation"].corr(data
["temp_max"]))
print("T Test and P value:",stats.ttest_ind(data["precipitat
ion"],data["temp_max"]))
```

Output[12]:

```
Pearson correlation: -0.22855481643297046
T Test and P value: Ttest_indResult(statistic=-51.60685279531918, pvalue=0.0)
```



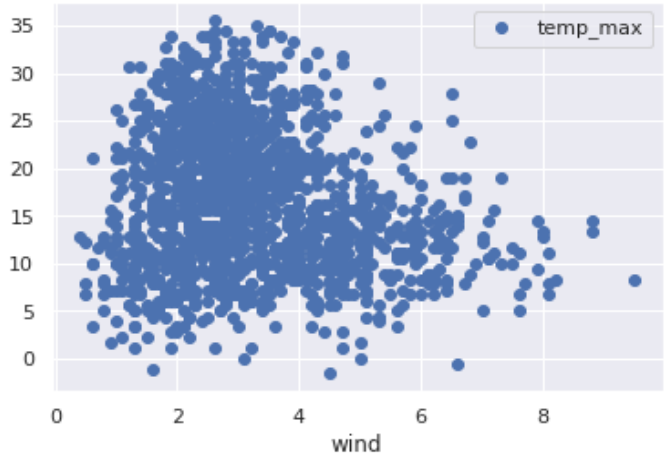
Hình 8. Mối tương quan giữa precipitation và temp_max.

Input[13]:

```
data.plot("wind","temp_max",style='o')
print("Pearson correlation:",data["wind"].corr(data["temp_ma
x"]))
print("T Test and P value:",stats.ttest_ind(data["wind"],dat
a["temp_max"]))
```

Output[13]:

```
Pearson correlation: -0.16485663487495486
T Test and P value: Ttest_indResult(statistic=-67.3601643301846, pvalue=0.0)
```



Hình 9. Mối tương quan giữa wind và temp_max.

Theo kết quả của ttest và p-value tính được bằng 0 từ trên thì chứng tỏ rằng giả thuyết H0 trong các cột tương ứng bị bác bỏ và các cột đều có ý nghĩa thống kê và đều có ảnh hưởng đến kế quả dự đoán.

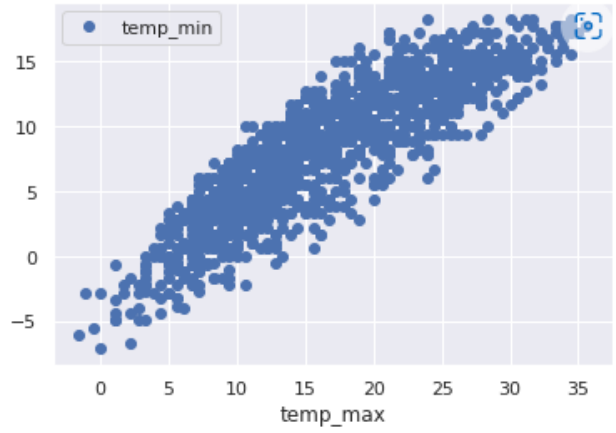
Đồng thời, ta cũng thấy hệ số tương quan giữa các cặp biến trên đều nằm trong khoảng -1 < r < 0, điều này có nghĩa là chúng có mối tương quan yếu với nhau hay có hệ số tương quan âm. Nghĩa là giá trị biến x tăng thì giá trị biến y giảm và ngược lại, giá trị biến y tăng thì giá trị biến x giảm.

Input[14]:

```
data.plot("temp_max","temp_min",style='o')
print("Pearson correlation:",data["temp_max"].corr(data["t
emp_min"]))
print("T Test and P value:",stats.ttest_ind(data["temp_max
"],data["temp_min"]))
```

Output[14]:

```
Pearson correlation: 0.8756866637108172
T Test and P value: Ttest_indResult(statistic=35.22645781239284, pvalue=7.526905087245157e-227)
```



Hình 10. Mối tương quan giữa temp_max và temp_min.

Dựa vào đồ thị Output[14], ta có thể nhận xét rằng là biến "temp_min" và biến "temp_max" có mối quan hệ tương quan dương với nhau và mối quan hệ tuyến tính này khá mạnh. Nghĩa là giá trị biến x tăng thì giá trị biến y tăng và ngược lại, giá trị biến y tăng thì giá trị biến x cũng tăng.

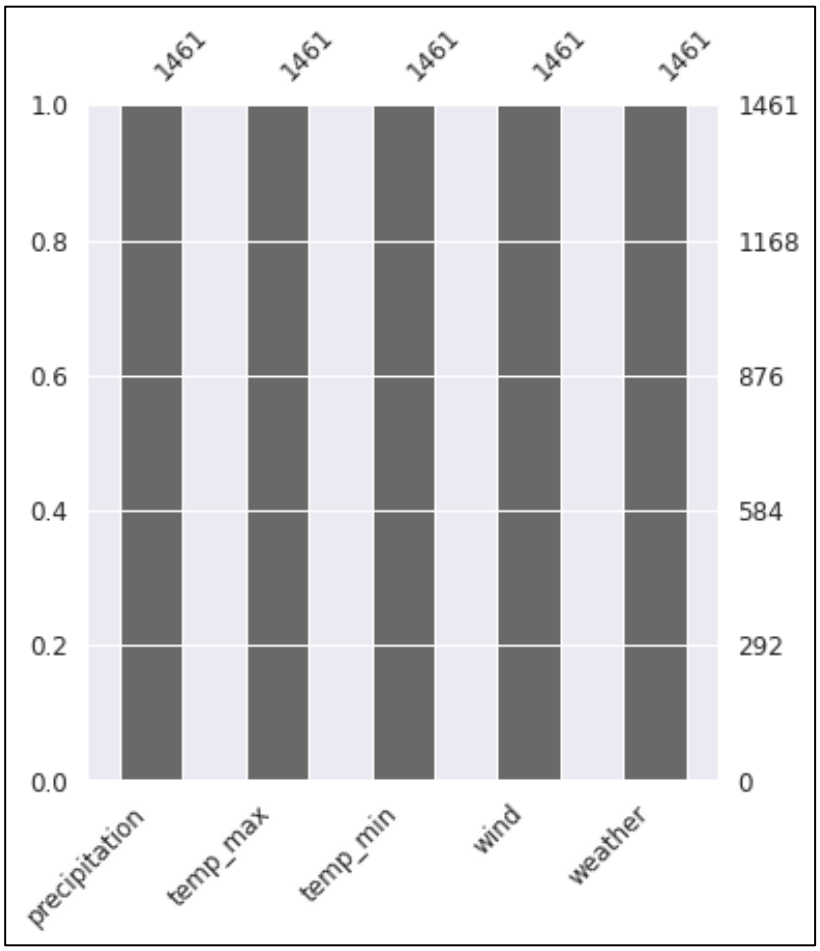
2.2.7. Giá trị NULL.

Tiếp theo, ta tiến hành kiểm tra xem trong bộ dữ liệu có tồn tại giá trị là NULL hay không.

Input[15]:

```
plt.figure(figsize=(12,6))
axz=plt.subplot(1,2,2)
mso.bar(data.drop(["date"],axis=1),ax=axz,fontsize=12);
```

Output[15]:



Hình 11. Kiểm tra giá trị NULL trong bộ dữ liệu.

Bằng cách quan sát biểu đồ ở phía trên, ta có thể kết luận rằng không tồn tại các giá trị NULL ở các biến bởi vì các cột đều có 1461 quan sát bằng đúng với số dòng của dữ liệu.

2.3. Tiền xử lý và làm sạch dữ liệu.

Đây là bước rất quan trọng trước khi đào tạo một mô hình Machine Learning, nó bao gồm nhiều bước thực hiện chẳng hạn như loại bỏ các điểm ngoại lai, loại các biến không cần thiết, không được sử dụng khi xây dựng mô hình hay xử lý các phân phối lệch. Đồng thời cũng thực hiện gắn nhãn dữ liệu và phân tách bộ dữ liệu thành các tập dữ liệu đào tạo (x_train, y_train) và các tập dữ liệu kiểm thử (x_test, y_test).

2.3.1. Loại bỏ các biến không cần thiết.

Giả định đầu tiên ở đây chính là trong bộ dữ liệu này, biến `date` là biến dữ liệu không cần thiết, không cần sử dụng đến, không gây ảnh hưởng đến kết quả trong quá trình xây dựng các mô hình dự báo của chúng ta. Vì vậy trong trường hợp đầu tiên, ta sẽ tiến hành loại bỏ biến này ra khỏi bộ dữ liệu.

Input[16]

```
df=data.drop(["date"],axis=1)
df.head()
```

Output[16]:

	precipitation	temp_max	temp_min	wind	weather
0	0.0	12.8	5.0	4.7	drizzle
1	10.9	10.6	2.8	4.5	rain
2	0.8	11.7	7.2	2.3	rain
3	20.3	12.2	5.6	4.7	rain
4	1.3	8.9	2.8	6.1	rain

2.3.2. Loại bỏ các điểm Outlier và các giá trị vô hạn.

Vì trong bộ dữ liệu trên có chứa các giá trị ngoại lai (Outlier), chúng ta sẽ loại bỏ chúng để làm cho bộ dữ liệu đồng đều hơn.

Input[17]:

```
Q1=df.quantile(0.25)
Q3=df.quantile(0.75)
IQR=Q3-Q1
df=df[~((df<(Q1-1.5*IQR))|(df>(Q3+1.5*IQR))).any(axis=1)]
```

Chúng ta loại bỏ đi những điểm Outlier bằng cách tính khoảng tứ phân vị IQR, sau đó loại đi những giá trị nằm ngoài khoảng (`Q1-1.5*IQR, Q3+1.5*IQR`). Những điểm nằm ngoài khoảng này được gọi là những điểm ngoại lai (Outlier).

2.3.3. Xử lý các phân phối lệch.

Ta xử lý hai biến có phân phối lệch là "precipition" và "wind" bằng cách lấy căn bậc 2 của chúng.

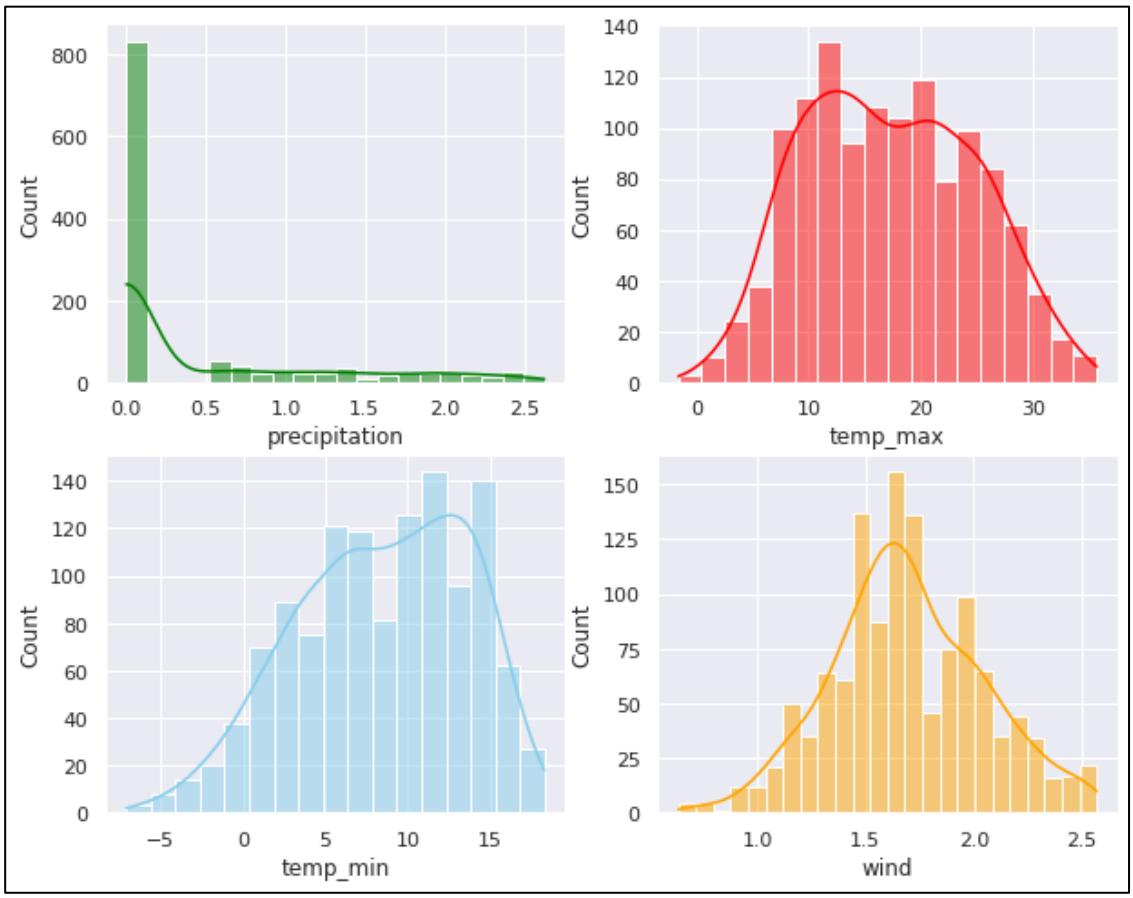
Input[18]:

```
df.precipitation=np.sqrt(df.precipitation)
df.wind=np.sqrt(df.wind)
```

Input[19]:

```
sns.set(style="darkgrid")
fig,axs=plt.subplots(2,2,figsize=(10,8))
sns.histplot(data=df,x="precipitation",kde=True,
ax=axs[0,0],color='green')
sns.histplot(data=df,x="temp_max",kde=True,ax=ax
s[0,1],color='red')
sns.histplot(data=df,x="temp_min",kde=True,ax=ax
s[1,0],color='skyblue')
sns.histplot(data=df,x="wind",kde=True,ax=axs[1,
1],color='orange')
```

Output[19]:



Hình 12. Đồ thị boxpot và phân phối của các biến sau khi xử lý và làm sạch (trường hợp 1).

2.3.4. Mã hóa dữ liệu ở biến weather.

Mã hóa các nhãn dữ liệu ở biến weather thành các giá trị từ 0 - 4 sử dụng hàm LabelEncoder(). Việc này là cần thiết để xây dựng mô hình dự đoán trong bài toán này.

Input[20]:

```
lc=LabelEncoder()
df["weather"]=lc.fit_transform(df["weather"])
```

Output[20]:

	precipitation	temp_max	temp_min	wind	weather
0	0.000000	12.8	5.0	2.167948	0
2	0.894427	11.7	7.2	1.516575	2
4	1.140175	8.9	2.8	2.469818	2
5	1.581139	4.4	2.2	1.483240	2
6	0.000000	7.2	2.8	1.516575	2

Ta có thể thấy các nhãn dữ liệu trong biến "weather" đã được mã hóa như sau: giá trị 0 tương ứng với Dizzle, giá trị 1 tương ứng với Fog, giá trị 2 tương ứng với rain, giá trị 3 tương ứng với Snow, giá trị 4 tương ứng với Sun.

2.3.5. Phân tách tập dữ liệu thành biến phụ thuộc và biến độc lập.

Tiếp theo, ta tiến hành phân tách giá trị các "precipitation", "wind", "temp_max", "temp_min" thành một biến x độc lập với biến "weather" để chuẩn bị cho công đoạn tách bộ dữ liệu thành các tập dữ liệu đào tạo và kiểm thử.

Input[21]:

```
x=((df.loc[:,df.columns!="weather"]).astype(int)).values[:,
0:]
y=df["weather"].values
```

Sau đó, ta chia bộ dữ liệu thành hai tập dữ liệu riêng biệt bao gồm tập dữ liệu đào tạo và tập dữ liệu kiểm thử với tỷ lệ tương ứng là 9:1. Sử dụng hàm train_test_split(), đây là một hàm có sẵn trong thư viện Sklearn.

Input[22]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_siz
e=0.1,random_state=2)
```

2.4. Đào tạo mô hình (trường hợp loại bỏ biến date).

2.4.1. K-Nearest Neighbor Classifier (KNN).

K-Nearest Neighbor Classifer là một bộ phân loại học có giám sát, phi tham số, sử dụng khoảng cách gần nhất để thực hiện phân loại hoặc dự đoán về việc nhóm một điểm dữ liệu riêng lẻ. Được sử dụng trong nhiều ứng dụng khác nhau, một số trường hợp như: xử lý trước dữ liệu, nhận dạng mẫu…

Ta tiến hành đào tạo mô hình sử dụng KNN bằng cách gọi hàm KNeighborsClassifier() từ thư viện sklearn.neighbors.

Input[23]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
knn_score = knn.score(x_test,y_test)
print("KNN Accuracy:", knn_score)
```

Output[23]:

```
KNN Accuracy: 0.75
```

Input[24]:

```
y_pred_knn = knn.predict(x_test)
conf_matrix = confusion_matrix(y_test, y_pred_knn)
print("Confusion Matrix")
print(conf_matrix)
```

Output[24]:

```
Confusion Matrix
[[ 0  0  0  0  1]
 [ 0  2  4  0  5]
 [ 1  0 26  0  6]
 [ 0  0  1  1  1]
 [ 1  5  6  0 64]]
```

Input[25]:

```
print('KNN\n',classification_report(y_test,y_pred_knn,
zero_division=0))
```

Output[25]:

```
KNN
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.29      0.18      0.22        11
           2       0.70      0.79      0.74        33
           3       1.00      0.33      0.50         3
           4       0.83      0.84      0.84        76

    accuracy                           0.75       124
   macro avg       0.56      0.43      0.46       124
weighted avg       0.75      0.75      0.74       124
```

2.4.2. Decision Tree.

Là một công cụ hỗ trợ quyết định sử dụng mô hình quyết định dạng cây và các hệ quả có thể xảy ra của chúng, bao gồm cả kết quả sự kiện may rủi, chi phí tài nguyên và tiện ích. Đó là một cách để hiển thị một thuật toán chỉ chứa các câu lệnh điều khiển có điều kiện.

Cây quyết định thường được sử dụng trong nghiên cứu hoạt động, đặc biệt là trong phân tích quyết định, để giúp xác định chiến lược có nhiều khả năng đạt được mục tiêu nhất, nhưng cũng là một công cụ phổ biến trong học máy.

Ta tiến hành đào tạo mô hình sử dụng Decision Tree bằng cách gọi hàm DecisionTreeClassifier từ thư viện sklearn.tree. Ta tiến hành xây dựng mô hình với các tham số max_depth khác nhau từ 1 đến 7 để tìm lấy mô hình có tham số với độ chính xác tốt nhất.

Input[26]:

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
max_depth_range = list(range(1, 8))
for depth in max_depth_range:
  dec = DecisionTreeClassifier(max_depth = depth, max
_leaf_nodes=15,random_state=0)
  dec.fit(x_train,y_train)
  dec_score = dec.score(x_test,y_test)
  print("Decison Tree Accuracy : ", dec_score)
```

Output[26]:

```
Decison Tree Accuracy :  0.8064516129032258
Decison Tree Accuracy :  0.8145161290322581
Decison Tree Accuracy :  0.8225806451612904
Decison Tree Accuracy :  0.8145161290322581
Decison Tree Accuracy :  0.8306451612903226
Decison Tree Accuracy :  0.8306451612903226
Decison Tree Accuracy :  0.8306451612903226
```

Ta nhận thấy rằng với max_depth nằm trong khoảng từ 5 đến 7 ta sẽ có được mô hình Decision Tree tốt nhất với 0.83 độ tin cậy.

Input[27]:

```
y_pred_dec = dec.predict(x_test)
conf_matrix = confusion_matrix(y_test, y_pred_dec)
print("Confusion Matrix")
print(conf matrix)
```

Output[27]:

```
Confusion Matrix
[[ 0  0  0  0  1]
 [ 0  0  0  0 11]
 [ 0  0 25  0  8]
 [ 0  0  0  3  0]
 [ 0  0  1  0 75]]
```

Input[28]:

```
print('Decision Tree\n',classification_report(y_test,y_pred_dt, zero_division=0))
```

Output[28]:

```
Decision Tree
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.00      0.00      0.00        11
           2       0.96      0.76      0.85        33
           3       1.00      1.00      1.00         3
           4       0.79      0.99      0.88        76

    accuracy                           0.83       124
   macro avg       0.55      0.55      0.54       124
weighted avg       0.76      0.83      0.79       124
```

2.4.3. Logistic Regression.

Phương pháp hồi quy logistic là một mô hình hồi quy học có giám sát nhằm dự đoán giá trị đầu ra rời rạc (discrete target variable) y ứng với một vector đầu vào x.

Mục đích của hồi quy logistic là ước tính xác suất của các sự kiện, bao gồm xác định mối quan hệ giữa các tính năng từ đó đự đoán xác suất của các kết quả, nên đối với hồi quy logistic ta sẽ có:

Input: dữ liệu input (ta sẽ coi có hai nhãn là 0 và 1).

Output: Xác suất dữ liệu input rơi vào nhãn 0 hoặc nhãn 1.

Ta tiến hành đào tạo mô hình sử dụng Logistic Regression bằng cách gọi hàm LogisticRegression từ thư viện sklearn.linear_model.

Input[29]:

```
from sklearn.linear_model import LogisticRegression
lg = LogisticRegression()
lg.fit(x_train,y_train)
lg_score = lg.score(x_test,y_test)
print("Logictic Accuracy : ", lg_score)
```

Ouput[29]:

```
Logistic Accuracy :  0.8064516129032258
```

Input[30]:

```
y_pred_lg = lg.predict(x_test)
conf_matrix = confusion_matrix(y_test, y_pred_lg)
print("Confusion Matrix")
print(conf_matrix)
```

Ouput[30]:

```
Confusion Matrix
[[ 0  0  0  0  1]
 [ 0  0  3  0  8]
 [ 0  0 26  0  7]
 [ 0  0  3  0  0]
 [ 0  0  2  0 74]]
```

Input[31]:

```
print('Logistic Regression\n',classification_report(y_
test,y_pred_lg, zero_division=0))
```

Output[31]:

```
Logistic Regression
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.00      0.00      0.00        11
           2       0.76      0.79      0.78        33
           3       0.00      0.00      0.00         3
           4       0.82      0.97      0.89        76

    accuracy                           0.81       124
   macro avg       0.32      0.35      0.33       124
weighted avg       0.71      0.81      0.75       124
```

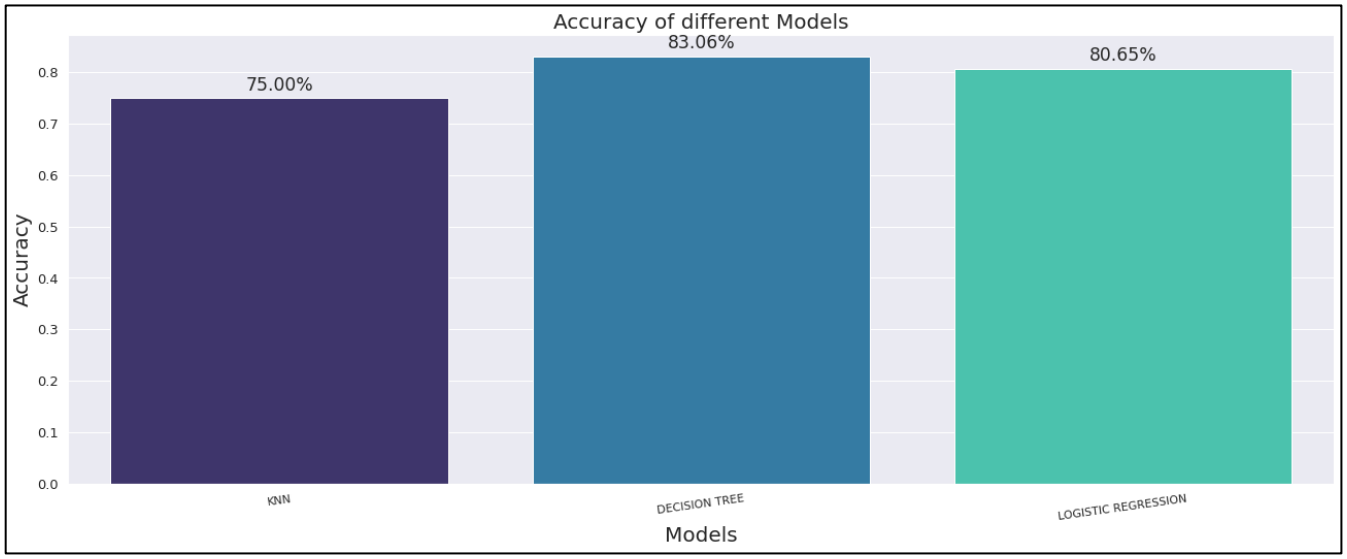
2.4.4. Biểu đồ so sánh các mô hình.

Input[32]:

```
mylist=[]
mylist2=[]
mylist.append(knn_score)
mylist2.append("KNN")
mylist.append(dec_score)
mylist2.append("DECISION TREE")
mylist.append(lg_score)
mylist2.append("LOGISTIC REGRESSION")
plt.rcParams['figure.figsize']=8,6
sns.set_style("darkgrid")
plt.figure(figsize=(22,8))
ax = sns.barplot(x=mylist2, y=mylist, palette = "mako"
, saturation =1.5)
```

```
plt.xlabel("Models", fontsize = 20 )
plt.ylabel("Accuracy", fontsize = 20)
plt.title("Accuracy of different Models", fontsize = 2
0)
plt.xticks(fontsize = 11, horizontalalignment = 'cente
r', rotation = 8)
plt.yticks(fontsize = 13)
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height:.2%}', (x + width/2, y + hei
ght*1.02), ha='center', fontsize = 'x-large')
plt.show()
```

Output[32]:



Hình 13. So sánh độ tin cậy của các mô hình đã xây dựng.

2.5. Đào tạo mô hình (trong trường hợp giữ nguyên biến date).

Câu hỏi tiếp theo đặt ra ở đây đó chính là liệu biến `date` có ảnh hưởng đến độ chính xác của mô hình hay không. Ví dụ như thời tiết của chúng ta bị ảnh hưởng theo từng mùa trong năm chẳng hạn, chính vì vậy ta tiếp tục xây dựng mô hình với bộ dữ liệu không loại bỏ đi biến `date` để kiểm tra giả thuyết này.

Input[33]:

```
df_date = pd.read_csv("https://raw.githubusercontent.com
/trunghq0205/IntroductionToDataScience_MTH10171/main/sea
ttle-weather.csv")
df_date.head()
```

Output[33]:

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	2012-01-02	10.9	10.6	2.8	4.5	rain
2	2012-01-03	0.8	11.7	7.2	2.3	rain
3	2012-01-04	20.3	12.2	5.6	4.7	rain
4	2012-01-05	1.3	8.9	2.8	6.1	rain

Đầu tiên, chúng ta chuyển kiểu dữ liệu ở biến `date` từ dạng chuỗi chuyển sang kiểu dữ liệu Datetime trong Python. Sau đó, loại bỏ đi thuộc tính ngày và năm trong `date`, trích xuất và giữ lại thuộc tính tháng vì thông thường thời tiết thường phụ thuộc theo các mùa trong năm và các mùa trong năm cũng thay đổi theo tháng.

Input[34]:

```
df_date.date = pd.to_datetime(df_date.date).dt.month
df_date.date
```

Ta thực hiện đổi tên biến `date` thành `month` cho phù hợp với trường dữ liệu mà nó lưu giữ.

Input[35]:

```
df_date = df_date.rename(columns = {'date':'month'})
df_date.head()
```

Output[35]:

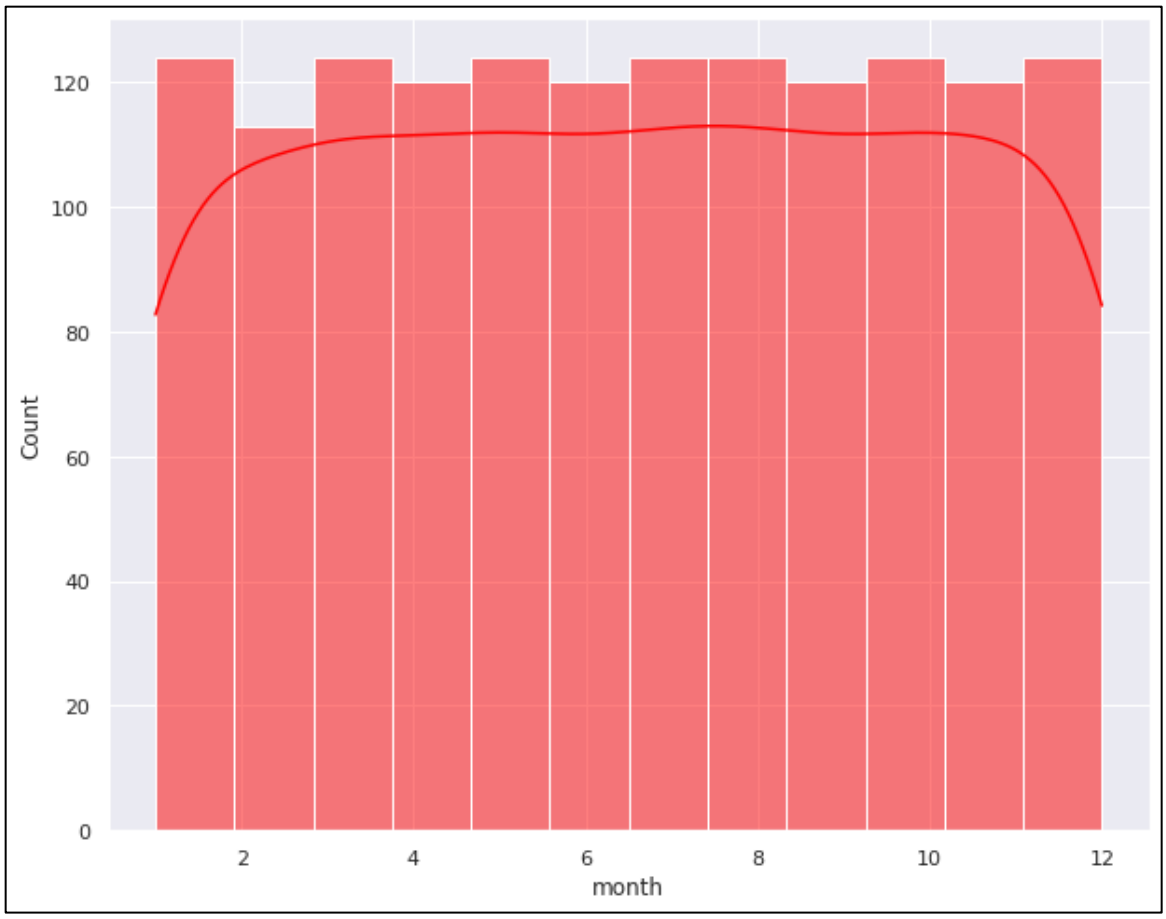
	month	precipitation	temp_max	temp_min	wind	weather
0	1	0.0	12.8	5.0	4.7	drizzle
1	1	10.9	10.6	2.8	4.5	rain
2	1	0.8	11.7	7.2	2.3	rain
3	1	20.3	12.2	5.6	4.7	rain
4	1	1.3	8.9	2.8	6.1	rain

Ta xem phân phối của biến `month` trong bộ dữ liệu bằng đồ thị Histogram.

Input[36]:

```
sns.set(style="darkgrid")
fig,axs = plt.subplots(figsize=(10,8))
plot = sns.histplot(data=df_date,x = "month",kde=True,co
lor='red');
```

Output[36]:



Hình 14. Đồ thị Histogram của biến month.

Ta có thể thấy dữ liệu quan trắc trong bộ dữ liệu này có sự phân bổ đồng đều giữa các tháng trong năm.

2.5.1. Làm sạch dữ liệu trước khi xây dựng các mô hình.

Tương tự như ở phần trước, chúng ta cũng sẽ bắt đầu xử lý và làm sạch dữ liệu trước khi xây dựng các mô hình dự báo. Bước này đã được thực hiện khá chi tiết ở phần trình bày phía trên nên chúng ta sẽ không nhắc lại ở bước này. Bao gồm: loại bỏ các điểm ngoại lai, xử lý các phân phối lệch, mã hóa dữ liệu biến `weather` và phân tách tập dữ liệu thành các tập train và test.

Input[37]:

```
# Loại bỏ các điểm Outlier.
Q1_date = df_date.quantile(0.25)
Q3_date = df_date.quantile(0.75)
IQR_date = Q3_date - Q1_date
df_date = df_date[~((df_date<(Q1_date-
1.5*IQR_date))|(df_date>(Q3_date+1.5*IQR_date))).any(axi
s=1)]
```

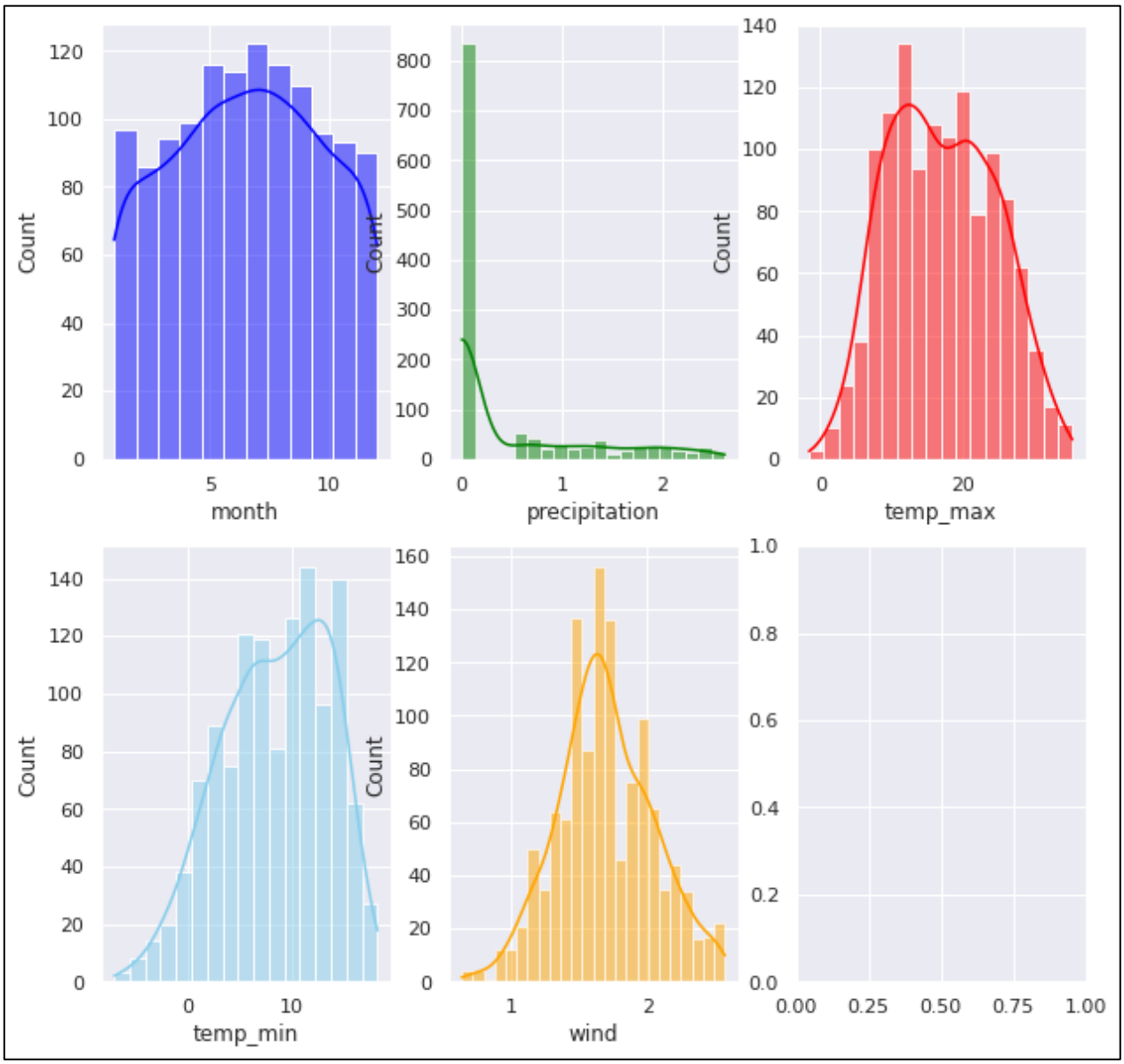
Input[38]:

```
# Xử lý các phân phối lệch.
df_date.precipitation=np.sqrt(df_date.precipitation)
df_date.wind=np.sqrt(df_date.wind)
```

Input[39]:

```
sns.set(style="darkgrid")
fig,axs=plt.subplots(2,2,figsize=(10,8))
sns.histplot(data=df_date,x="precipitation",kde=True,ax=
axs[0,0],color='green');
sns.histplot(data=df_date,x="temp_max",kde=True,ax=axs[0
,1],color='red');
sns.histplot(data=df_date,x="temp_min",kde=True,ax=axs[1
,0],color='skyblue');
sns.histplot(data=df_date,x="wind",kde=True,ax=axs[1,1],
color='orange');
```

Output[39]:



Hình 15. Đồ thị Histogram của các biến sau khi xử lý và làm sạch (trường hợp 2).

Tiếp đến, ta mã hóa các tình trạng thời tiết thành các giá trị từ 0-4, sau đó phân tách dữ liệu thành các tập train và test.

Input[40]:

```
lc_date=LabelEncoder()
df_date["weather"]=lc_date.fit_transform(df_date["weather"])
df_date.head()
```

Input[41]:

```
x_date = ((df_date.loc[:,df_date.columns!="weather"]).astype(in
t)).values[:,0:]
```

Input[42]:

```
x_train_date,x_test_date,y_train_date,y_test_date=train_test_sp
lit(x_date,y_date,test_size=0.1,random_state=2)
```

2.5.2. K-Neighbor Nearest Classifier.

Input[43]:

```
from sklearn.neighbors import KNeighborsClassifier
knn_date = KNeighborsClassifier()
knn_date.fit(x_train_date,y_train_date)
knn_date_score = knn_date.score(x_test_date,y_test_date)
print("KNN Accuracy (with month column):", knn_date_score)
```

Output[43]:

```
KNN Accuracy (with month column): 0.7338709677419355
```

Input[44]:

```
y_pred_knn_date = knn_date.predict(x_test_date)
conf_matrix_knn_date = confusion_matrix(y_test_date, y_pred_knn
_date)
print("Confusion Matrix (with month column)")
print(conf matrix knn date)
```

Output[44]:

```
Confusion Matrix (with month column)
[[ 0  0  0  0  1]
 [ 0  3  3  0  5]
 [ 0  2 24  0  7]
 [ 1  0  0  1  1]
 [ 1  3  9  0 63]]
```

Input[45]:

```
print('KNN (with month column)\n',classification_report(
y_test_date,y_pred_knn_date, zero_division=0))
```

Output[45]:

```
KNN (with month column)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.38      0.27      0.32        11
           2       0.67      0.73      0.70        33
           3       1.00      0.33      0.50         3
           4       0.82      0.83      0.82        76

    accuracy                           0.73       124
   macro avg       0.57      0.43      0.47       124
weighted avg       0.74      0.73      0.73       124
```

Ta có thể nhận xét rằng việc có thêm biến `month` trong việc đào tạo mô hình ở trường hợp này đã làm giảm độ tin cậy của mô hình sử dụng KNN từ 0.75 xuống còn xấp xỉ 0.734.

2.5.3. Decision Tree.

Input[46]:

```
from sklearn.tree import DecisionTreeClassifier
max_depth_range_date = list(range(1, 8))
for depth in max_depth_range_date:
  dec_date = DecisionTreeClassifier(max_depth = depth, max_leaf
_nodes = 15,random_state=0)
  dec_date.fit(x_train_date,y_train_date)
  dec_date_score = dec_date.score(x_test_date,y_test_date)
  print("Decison Tree Accuracy (with month column): ", dec_date
_score)
```

Output[46]:

```
Decison Tree Accuracy (with month column):  0.8064516129032258
Decison Tree Accuracy (with month column):  0.8145161290322581
Decison Tree Accuracy (with month column):  0.8225806451612904
Decison Tree Accuracy (with month column):  0.8306451612903226
Decison Tree Accuracy (with month column):  0.7983870967741935
Decison Tree Accuracy (with month column):  0.8225806451612904
Decison Tree Accuracy (with month column):  0.7983870967741935
```

Input[47]:

```
y_pred_dec_date = dec_date.predict(x_test_date)
conf_matrix_dec_date = confusion_matrix(y_test_date, y_pred_dec
_date)
print("Confusion Matrix (with month column)")
print(conf_matrix_dec_date)
```

Output[47]:

```
Confusion Matrix (with month column)
[[ 0  0  0  0  1]
 [ 0  0  2  0  9]
 [ 0  0 26  0  7]
 [ 0  0  1  2  0]
 [ 0  0  5  0 71]]
```

Input[48]:

```
print('Decision Tree (with month column)\n',classification_re
port(y_test_date,y_pred_dec_date, zero_division=0))
```

Output[48]:

```
Decision Tree (with month column)
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.00      0.00      0.00        11
           2       0.76      0.79      0.78        33
           3       1.00      0.67      0.80         3
           4       0.81      0.93      0.87        76

    accuracy                           0.80       124
   macro avg       0.51      0.48      0.49       124
weighted avg       0.72      0.80      0.76       124
```

Ta có thể nhận xét rằng trong trường hợp đào tạo mô hình có thêm biến month thì độ tin cậy của mô hình sử dụng cây quyết định đã có sự thay đổi. Khi max_depth = 7 thì độ tin cậy của mô hình đã giảm đi so với trường hợp không có biến month (từ 0.83 xuống còn xấp xỉ 0.799). Nhưng trong trường hợp này, khi max_depth = 4 thì độ tin cậy đạt được là 0.83, bằng đúng với trường hợp ở phía trên (trường hợp không có biến `month`).

2.5.4. Logistic Regression.

Input[49]:

```
from sklearn.linear_model import LogisticRegression
lg_date = LogisticRegression()
lg_date.fit(x_train_date,y_train_date)
lg_date_score = lg_date.score(x_test_date,y_test_date)
print("Logistic Accuracy (with month column): ", lg_da
te_score)
```

Output[49]:

```
Logistic Accuracy (with month column):
0.8064516129032258
```

Input[50]:

```
y_pred_lg_date = lg_date.predict(x_test_date)
conf_matrix_date = confusion_matrix(y_test_date, y_pred_
lg_date)
print("Confusion Matrix (with month column)")
print(conf_matrix_date)
```

Output[50]:

```
Confusion Matrix (with month column)
[[ 0  0  0  0  1]
 [ 0  0  2  0  9]
 [ 0  0 26  0  7]
 [ 0  0  3  0  0]
 [ 0  0  2  0 74]]
```

Input[51]:

```
print('Logistic Regression (with month column)\n',classi
fication_report(y_test_date,y_pred_lg_date, zero_divisio
n=0))
```

Output[51]:

```
Logistic Regression (with month column)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.00      0.00      0.00        11
           2       0.79      0.79      0.79        33
           3       0.00      0.00      0.00         3
           4       0.81      0.97      0.89        76

    accuracy                           0.81       124
   macro avg       0.32      0.35      0.33       124
weighted avg       0.71      0.81      0.75       124
```

Dựa vào trên ta thấy việc có thêm biến `month` vẫn không làm thay đổi độ tin cậy của mô hình sử dụng Logistic Regression.

Như vây, việc sử dụng thêm biến month được trích xuất từ biến date để đào tạo mô hình cho kết quả không mấy khả quan khi có 1 mô hình bị giảm độ tin cậy (KNN), 2 mô hình không thay đổi độ tin cậy là Decision Tree và Logistic Regression (trong đó, đối với Decision Tree ta phải thay đổi max_depth từ 7 thành 4 mới có thể giữ nguyên được độ tin cậy của mô hình như trước đó.).

2.5.5. Trường hợp giữ nguyên biến date theo định dạng YYYY-MM-DD.

Đối với trường hợp này, ta sẽ không tách lấy trường dữ liệu tháng trong biến date nữa và giữ nguyên biến date theo định dạng YYY-MM-DD như ban đầu của bộ dữ liệu để xây dựng các mô hình dự đoán sau đó so sánh sự khác biệt.

Tương tự như các bước thực hiện ở các phần trên, trước khi xây dựng các mô hình chúng ta cần phải chuẩn bị và làm sạch bộ dữ liệu. Tại đây chúng ta chỉ đi sơ qua quá trình xử lý và làm sạch dữ liệu (về chi tiết đã được trình bày tương tự ở các phần khác).

Input[52]:

```
df3 = pd.read_csv("https://raw.githubusercontent.com/
trunghq0205/IntroductionToDataScience_MTH10171/main/s
eattle-weather.csv")
df3.head()
```

Output[52]:

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	2012-01-02	10.9	10.6	2.8	4.5	rain
2	2012-01-03	0.8	11.7	7.2	2.3	rain
3	2012-01-04	20.3	12.2	5.6	4.7	rain
4	2012-01-05	1.3	8.9	2.8	6.1	rain

Input[53]:

```
# Loại bỏ các điểm Outlier.
Q1 = df3.quantile(0.25)
Q3 = df3.quantile(0.75)
IQR = Q3 - Q1
df3 = df3[~((df3<(Q1-
1.5*IQR))|(df3>(Q3+1.5*IQR))).any(axis=1)]
```

Input[54]:

```
# Xử lý các phân phối lệch.
df3.precipitation=np.sqrt(df3.precipitation)
df3.wind=np.sqrt(df3.wind)
```

Input[55]:

```
lc = LabelEncoder()
df3["weather"]=lc.fit_transform(df3["weather"])
df3.head()
```

Output[55]:

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.000000	12.8	5.0	2.167948	0
2	2012-01-03	0.894427	11.7	7.2	1.516575	2
4	2012-01-05	1.140175	8.9	2.8	2.469818	2
5	2012-01-06	1.581139	4.4	2.2	1.483240	2
6	2012-01-07	0.000000	7.2	2.8	1.516575	2

Input[56]:

```
df3.date = pd.to_datetime(df3.date)
df3.info()
```

Output[56]:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1233 entries, 0 to 1460
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   date           1233 non-null   datetime64[ns]
 1   precipitation  1233 non-null   float64       
 2   temp_max       1233 non-null   float64       
 3   temp_min       1233 non-null   float64       
 4   wind           1233 non-null   float64       
 5   weather        1233 non-null   int64         
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 67.4 KB
```

Input[57]:

```
x_df3 = ((df3.loc[:,df3.columns!="weather"]).astype(np.
int64)).values[:,0:]
y_df3 = df3["weather"].values
```

Input[58]:

```
x_train_df3,x_test_df3,y_train_df3,y_test_df3 = train_
test_split(x_df3,y_df3,test_size=0.1,random_state=2)
```

2.5.5.1. K-Neighbor Nearest Classifier.

Input[59]:

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn_df3 = KNeighborsClassifier()
knn_df3.fit(x_train_df3,y_train_df3)
knn_score_df3 = knn_df3.score(x_test_df3,y_test_df3)
print("KNN Accuracy:", knn_score_df3)
```

Output[59]:

```
KNN Accuracy: 0.6290322580645161
```

Mô hình K-Neighbor Nearest Classifier đã giảm độ chính xác xuống chỉ còn 0.629 So với 2 trường hợp ở phía trên thì trường hợp này cho kết quả xấu nhất.

2.5.5.2. Decision Tree.

Input[60]:

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
max_depth_range = list(range(1, 8))
for depth in max_depth_range:
  dec_df3 = DecisionTreeClassifier(max_depth = depth, max_leaf_
nodes=15,random_state=0)
  dec_df3.fit(x_train_df3,y_train_df3)
  dec_score_df3 = dec_df3.score(x_test_df3,y_test_df3)
  print("Decison Tree Accuracy : ", dec_score_df3)
```

Output[60]:

```
Decison Tree Accuracy :  0.8064516129032258
Decison Tree Accuracy :  0.8145161290322581
Decison Tree Accuracy :  0.7903225806451613
Decison Tree Accuracy :  0.8467741935483871
Decison Tree Accuracy :  0.8145161290322581
Decison Tree Accuracy :  0.8145161290322581
Decison Tree Accuracy :  0.8145161290322581
```

Mô hình Decision Tree với biến `date` được giữ nguyên ở định dạng YYYY-MM-DD đã cho mô hình với độ tin cậy là 0.8467 có tham số max_depth = 4. Đây là mô hình có độ tin cậy tốt nhất trong số kết quả mà chúng ta đã có.

2.5.5.3. Logistic Regression.

Input[61]:

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lg_df3 = LogisticRegression()
lg_df3.fit(x_train_df3,y_train_df3)
lg_score_df3 = lg_df3.score(x_test_df3, y_test_df3)
print("Logistic Accuracy : ", lg_score_df3)
```

Output[61]:

```
Logistic Accuracy :  0.008064516129032258
```

Mô hình trên chỉ cho ra 0.008 độ chính xác, đây là một kết quả vô cùng thấp.

Nhận xét: Khi giữ nguyên biến date ở dịnh dạng YYYY-MM-DD thì chúng ta đã có được độ chính xác cao hơn so với các trường hợp khác là 0.847 ở mô hình sử dụng Decision Tree. Nhưng ở trường hợp này sẽ có 1 điều không được hợp lý đó là chúng ta dự đoán thời tiết nhưng lại dựa vào chính xác vào một ngày-tháng-năm(YYYY-MM-DD) nào đó, điều này có hơi không thực tế so với khi chỉ dựa vào thông tin tháng (MM).

3. KIỂM THỬ KẾT QUẢ DỰ ĐOÁN CỦA MÔ HÌNH.

Ở đây, chúng ta sẽ sử dụng một mô hình tiêu biểu từ số các mô hình đã được xây dựng ở phía trên để làm kiểm thử kết quả. Ta sẽ chọn mô hình được xây dựng với Decision Tree có biến month lưu thông tin tháng được trích xuất từ biến date, có tham số max_depth = 4. Mô hình này có độ chính xác là 0.83.

Input[62]:

```
dec_test = DecisionTreeClassifier(max_depth = 4, max_leaf_no
des = 15,random_state=0)
dec_test.fit(x_train_date,y_train_date)
dec_test_score = dec_test.score(x_test_date,y_test_date)
print("Decison Tree Accuracy (with month column): ", dec_tes
t_score)
```

Output[62]:

```
Decison Tree Accuracy (with month column):
```

Input[63]:

```
for i in (range(len(y_test_df3))):
    print("-----------------------------------")
    ot = dec_test.predict([x_test_df3[i]])
    if(ot==0):
        print("The weather predict is: Drizzle")
    elif(ot==1):
        print("The weather predict is: Fog")
    elif(ot==2):
        print("The weather predict is: Rain")
    elif(ot==3):
        print("The weather predict is: Snow")
    else:
        print("The weather predict is: Sun")
```

```
    ac = y_test_df3[i]
    if(ac==0):
        print("The weather actual is: Drizzle")
    elif(ac==1):
        print("The weather actual is: Fog")
    elif(ac==2):
        print("The weather actual is: Rain")
    elif(ac==3):
        print("The weather actual is: Snow")
    else:
        print("The weather actual is: Sun")
```

Output[63]: Một vài dòng kết quả được xuất ra.

```
---------------------------------
The weather predict is: Sun
The weather actual is: Sun
---------------------------------
The weather predict is: Snow
The weather actual is: Snow
---------------------------------
The weather predict is: Sun
The weather actual is: Sun
---------------------------------
The weather predict is: Sun
The weather actual is: Sun
---------------------------------
The weather predict is: Snow
The weather actual is: Snow
---------------------------------
The weather predict is: Rain
The weather actual is: Rain
---------------------------------
The weather predict is: Sun
The weather actual is: Sun
---------------------------------
The weather predict is: Sun
The weather actual is: Sun
---------------------------------
The weather predict is: Rain
The weather actual is: Rain
---------------------------------
```

Kiểm thử kết quả với dữ liệu được nhập thủ công:

Input[64]:

```
input=[[10,0.3,15.6,0.0,2.5]]
ot = dec_test.predict(input)
print("The weather is:")
if(ot==0):
    print("Drizzle")
elif(ot==1):
    print("Fog")
elif(ot==2):
    print("Rain")
elif(ot==3):
    print("Snow")
else:
    print("Sun")
```

Output[64]:

```
The weather is:
Sun
```

4. THÔNG TIN BÀI NỘP (BẢN TRÌNH CHIẾU).

- Đường dẫn đến bài báo cáo trên nền tảng Google Colab: https://colab.research.google.com/drive/1xF1djO43fKVzlKZNV0FfUqmXBlMRsIhV?usp=sharing
- Đường dẫn đến bài báo cáo trên nền tảng Github: https://github.com/trunghq0205/IntroductionToDataScience_MTH10171

  hoặc :

  https://github.com/trunghq0205/IntroductionToDataScience_MTH10171/blob/main/WeatherPrediction_Group15.ipynb

KẾT LUẬN

Qua việc xây dựng các mô hình trên, ta thấy rằng khi dự đoán về các tình trạng thời tiết như snow, fog và dizzle các mô hình dự đoán không được hiệu quả. Nguyên do là trong tập dữ liệu có quá ít dữ liệu đề cập về 3 tình trạng thời tiết này, làm ảnh hưởng đến độ chính xác của mô hình. Trong khi đó, mô hình dự đoán khá tốt khi làm việc với tình trạng thời tiết rain và sun.

Qua việc xây dựng các mô hình ở hai trường hợp, đó là loại bỏ dữ liệu về ngày quan trắc và sử dụng dữ liệu về ngày quan trắc (biến date) ta thấy rằng khi giữ lại biến date trước khi đào tạo các mô hình cho kết quả không mấy khả quan hơn so với khi bỏ đi biến date (Đối với KNN độ tin cậy đã giảm đi, hai mô hình còn lại Decision Tree và Logistic Regression cần phải thay đổi tham số mới có thể giữ nguyên được độ chính xác như ở trường hợp trước (max_depth từ 5 hoặc 6 hoặc 7 giảm xuống còn 4 ở Decision Tree)).

Riêng về trường hợp thử nghiệm khi ta giữ nguyên biến date theo định dạng YYYY-MM-DD thì ta nhận được độ chính xác là 0.8467741935483871 ở mô hình sử dụng Decision Tree (có max_depth = 4). Tuy nhiên ở trường hợp này, ta có thể đưa ra một nhận xét chủ quan rằng mô hình này sẽ không được hợp lý khi mà dự báo một tình trạng thời tiết lại dựa vào một ngày chi tiết ngày-tháng-năm như thế.

Biến date đã thực sự gây ảnh hưởng đến kết quả của chúng ta nhưng không mấy tích cực (có nhiều mô hình khi được xây dựng lại đã giảm đi độ tin cậy so với trường hợp đầu tiên - loại bỏ date).

Ta cũng nhận thấy được rằng dữ liệu khi được đào tạo với mô hình Decision Tree cho độ chính xác cao nhất trong số 3 mô hình được lựa chọn với độ chính xác thường gặp là 83.06% (có trường hợp 84.67%). Vì vậy, ta có thể kết luận rằng Decision Tree là mô hình phù hợp nhất đối với tập dữ liệu của chúng ta so với các mô hình được sử dụng trong bài báo cáo này.