รายละเอียดผลการทดลอง

6610450871 นายชนพัฒน์ โชติกุลรัตน์ หมู่ 200

Model = KNeighborsClassifier
Dataset = IndiaWeather

```python
import pandas as pd
from sklearn.model_selection import train_test_split , GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder, StandardScaler
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Python

- ทำการ import library ต่างๆ

```python
df = pd.read_excel('IndiaWeather.xlsx')
df
```

Python

- อ่านข้อมูลจากไฟล์ Excel ชื่อ "IndiaWeather.xlsx"
- เก็บข้อมูลที่อ่านได้ลงใน DataFrame ของ pandas ชื่อ `df`
- แสดงเนื้อหาของ DataFrame

	อุณหภูมิ	ความชื้น	ปริมาณ PM2.5	ปริมาณ PM10	ปริมาณไนโตรเจน	ปริมาณซัลเฟอร์	ปริมาณคาร์บอน	ระยะห่างจากโรงงาน	ความหนาแน่นประชากร	คุณภาพอากาศ
0	24.7	53.8	2.1	8.7	25.1	21.8	0.88	10.0	310	ปานกลาง
1	25.8	65.6	12.7	18.5	12.3	26	1.02	0.0	297	ดี
2	26.6	55.2	26.6	39.1	?	25.8	0.54	0.6	316	ปานกลาง
3	24.3	63	2.5	13.8	15.9	3.7	1.3	6.6	270	แย่
4	23.3	73.2	19.9	37.2	17.1	19.6	1.15	1.7	319	ดี
...	...	...	...	...	...	...	...	...	...	...
495	27.3	59.5	65.7	73.5	18.5	9.6	0.51	0.2	290	ปานกลาง
496	22.5	58.6	46.4	57.8	10.7	27.6	1.13	4.1	293	ปานกลาง
497	24.4	?	31.5	40.2	12.5	2	0.66	0.3	264	ปานกลาง
498	19.2	50.7	56.8	65.6	14.1	13	0.45	6.4	322	ปานกลาง
499	26.9	58.3	2.7	21.5	?	0.8	1.36	6.1	306	อันตรายต่อสุขภาพ

```python
df.isnull().sum()
```

Python

```
อุณหภูมิ               0
ความชื้น               0
ปริมาณ PM2.5           0
ปริมาณ PM10            0
ปริมาณไนโตรเจน         0
ปริมาณซัลเฟอร์         0
ปริมาณคาร์บอน          0
ระยะห่างจากโรงงาน      0
ความหนาแน่นประชากร     0
คุณภาพอากาศ            0
dtype: int64
```

คำนวณจำนวนของค่าว่าง (missing value) ในแต่ละคอลัมน์ของ DataFrame `df` และแสดงผลลัพธ์ออกมา

ประโยชน์:

- ช่วยในการตรวจสอบคุณภาพข้อมูล
- ช่วยในการตัดสินใจว่าจะจัดการกับค่าว่างอย่างไร เช่น การลบแถวที่มีค่าว่าง การเติมค่าที่เหมาะสม (imputation) เป็นต้น

```python
df.rename(columns = {'อุณหภูมิ' : 'temperature','ความชื้น' : 'Humidity','ปริมาณ PM2.5' : 'PM2.5','ปริมาณ PM10' : 'PM10','ปริมาณไนโตรเจน':'NO2','ปริมาณซัลเฟ
df
```

Python

	temperature	Humidity	PM2.5	PM10	NO2	SO2	CO	Distance_from_Industrial_Areas	Population_Density	Air_Quality
0	24.7	53.8	2.1	8.7	25.1	21.8	0.88	10.0	310	ปานกลาง
1	25.8	65.6	12.7	18.5	12.3	26	1.02	0.0	297	ดี
2	26.6	55.2	26.6	39.1	?	25.8	0.54	0.6	316	ปานกลาง
3	24.3	63	2.5	13.8	15.9	3.7	1.3	6.6	270	แย่
4	23.3	73.2	19.9	37.2	17.1	19.6	1.15	1.7	319	ดี
...	...	...	...	...	...	...	...	...	...	...
495	27.3	59.5	65.7	73.5	18.5	9.6	0.51	0.2	290	ปานกลาง
496	22.5	58.6	46.4	57.8	10.7	27.6	1.13	4.1	293	ปานกลาง
497	24.4	?	31.5	40.2	12.5	2	0.66	0.3	264	ปานกลาง
498	19.2	50.7	56.8	65.6	14.1	13	0.45	6.4	322	ปานกลาง
499	26.9	58.3	2.7	21.5	?	0.8	1.36	6.1	306	อันตรายต่อสุขภาพ

ทำการเปลี่ยนชื่อคอลัมน์ทั้งหมดใน DataFrame `df` ให้เป็นภาษาอังกฤษ เพื่อให้เข้าใจง่ายขึ้นและสะดวกในการวิเคราะห์ข้อมูลต่อไป เช่น การนำข้อมูลไปใช้กับโมเดล Machine Learning หรือสร้าง Visualization ต่างๆ

ประโยชน์ของการเปลี่ยนชื่อคอลัมน์:

- ทำให้ข้อมูลเป็นระเบียบและอ่านง่ายขึ้น
- ลดความผิดพลาดในการเรียกใช้คอลัมน์
- อำนวยความสะดวกในการวิเคราะห์ข้อมูลต่อไป

```python
df.Air_Quality.value_counts()
```

```
Air_Quality
ดี                195
ปานกลาง           157
แย่                99
อันตรายต่อสุขภาพ    49
Name: count, dtype: int64
```

นับจำนวนครั้งที่แต่ละค่าคุณภาพอากาศ (Air_Quality) ปรากฏใน DataFrame และแสดงผลลัพธ์ออกมาในรูปแบบของ Series

ประโยชน์:

- **วิเคราะห์การกระจายของข้อมูล:** ช่วยให้ทราบว่าค่าคุณภาพอากาศแต่ละระดับมีจำนวนข้อมูลเท่าใด
- **ตรวจสอบความสมดุลของข้อมูล:** สามารถใช้ในการตรวจสอบว่าข้อมูลมีความสมดุลหรือไม่ (balanced)
- **สำรวจข้อมูลเบื้องต้น:** ช่วยในการสำรวจข้อมูลเบื้องต้นก่อนที่จะทำการวิเคราะห์เชิงลึกต่อไป

```python
df.replace('?', np.nan, inplace=True)
for col in df.columns[:-1]:
    if pd.api.types.is_numeric_dtype(df[col]):
        df[col] = df[col].astype(float)
        for label in df['Air_Quality'].unique():
            df.loc[(df['Air_Quality'] == label) & (df[col].isna()), col] = df[col].loc[df['Air_Quality'] == label].
```

ดำเนินการเตรียมข้อมูลเพื่อจัดการกับค่าว่าง (missing values) ใน DataFrame โดยมีขั้นตอนดังนี้

1. แทนที่สัญลักษณ์ '?' ด้วยค่าว่าง (NaN)
2. ตรวจสอบและแปลงชนิดข้อมูลของคอลัมน์เป็นตัวเลข
3. เติมค่าว่างในแต่ละคอลัมน์ด้วยค่าเฉลี่ยของข้อมูลในกลุ่มที่มีคุณภาพอากาศเดียวกัน

ประโยชน์:

- **จัดการค่าว่าง:** ช่วยแก้ปัญหาค่าว่างในข้อมูล ซึ่งอาจส่งผลกระทบต่อการวิเคราะห์และการฝึกสอนโมเดล
- **ปรับปรุงคุณภาพข้อมูล:** ช่วยปรับปรุงคุณภาพข้อมูลโดยการเติมค่าที่สมเหตุสมผลแทนค่าว่าง
- **เตรียมข้อมูลสำหรับการวิเคราะห์:** ช่วยเตรียมข้อมูลให้พร้อมสำหรับขั้นตอนการวิเคราะห์และการฝึกสอนโมเดลต่อไป

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 10 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   temperature                     500 non-null    float64
 1   Humidity                        500 non-null    float64
 2   PM2.5                           500 non-null    float64
 3   PM10                            500 non-null    float64
 4   NO2                             500 non-null    float64
 5   SO2                             500 non-null    float64
 6   CO                              500 non-null    float64
 7   Distance_from_Industrial_Areas  500 non-null    float64
 8   Population_Density              500 non-null    float64
 9   Air_Quality                     500 non-null    object
dtypes: float64(9), object(1)
memory usage: 39.2+ KB
```

ตรวจสอบโครงสร้างของ DataFrame: ช่วยให้คุณทราบจำนวนคอลัมน์ ชื่อคอลัมน์ ชนิดข้อมูลของแต่ละคอลัมน์ และจำนวนข้อมูลในแต่ละคอลัมน์

ระบุค่าว่าง: ช่วยในการระบุว่ามีค่าว่างอยู่ในคอลัมน์ใดบ้างและมีจำนวนเท่าใด

สำรวจข้อมูลเบื้องต้น: ช่วยในการสำรวจข้อมูลเบื้องต้นก่อนที่จะทำการวิเคราะห์ข้อมูลเชิงลึกต่อไป

```python
pm25_bins = [0, 38, 51, 91, np.inf]
pm25_labels = ['ดี', 'ปานกลาง', 'แย่', 'อันตรายต่อสุขภาพ']
pm25_filtered = pd.cut(df['PM2.5'], bins=pm25_bins, labels=pm25_labels, right=False)
df = df[pm25_filtered == df['Air_Quality']]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 177 entries, 1 to 496
Data columns (total 10 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   temperature                     177 non-null    float64
 1   Humidity                        177 non-null    float64
 2   PM2.5                           177 non-null    float64
 3   PM10                            177 non-null    float64
 4   NO2                             177 non-null    float64
 5   SO2                             177 non-null    float64
 6   CO                              177 non-null    float64
 7   Distance_from_Industrial_Areas  177 non-null    float64
 8   Population_Density              177 non-null    float64
 9   Air_Quality                     177 non-null    object
dtypes: float64(9), object(1)
memory usage: 15.2+ KB
```

ทำการ:

1. **แบ่งกลุ่มค่า PM2.5:** โดยอ้างอิงจากช่วงที่กำหนดไว้และให้ label ตามคุณภาพอากาศ

2. **กรองข้อมูล:** เลือกเฉพาะข้อมูลที่ค่า PM2.5 ที่แบ่งกลุ่มได้ตรงกับ label คุณภาพอากาศที่ระบุไว้ในคอลัมน์ `Air_Quality`
3. **ตรวจสอบข้อมูล:** แสดงข้อมูลสรุปของ DataFrame หลังจากการกรอง เพื่อให้เห็นภาพรวมของข้อมูลที่เหลือ

วัตถุประสงค์:

- **ตรวจสอบความถูกต้องของข้อมูล:** ตรวจสอบว่าค่า PM2.5 และ label คุณภาพอากาศสอดคล้องกันหรือไม่
- **เตรียมข้อมูลสำหรับวิเคราะห์:** ได้ DataFrame ที่มีข้อมูลเกี่ยวกับ PM2.5 และคุณภาพอากาศที่เชื่อถือได้มากขึ้น

ประโยชน์:

- ช่วยให้การวิเคราะห์ข้อมูลมีความแม่นยำมากขึ้น
- ลดความผิดพลาดในการวิเคราะห์ที่อาจเกิดจากข้อมูลที่ไม่สอดคล้องกัน

```python
MOLAR_VOLUME = 24.45
MW_NO2 = 46.01
MW_SO2 = 64.07
MW_CO = 28.01
def ppb_to_ugm3(ppb, molecular_weight):
    ugm3 = (ppb * molecular_weight * 1e-3) / MOLAR_VOLUME
    return ugm3

def ppm_to_ugm3(ppm, molecular_weight):
    ugm3 = (ppm * molecular_weight) / MOLAR_VOLUME
    return ugm3
```

Python

```python
df['NO2'] = ppm_to_ugm3(df['NO2'], MW_NO2)
df['SO2'] = ppm_to_ugm3(df['SO2'], MW_SO2)
df['CO'] = ppm_to_ugm3(df['CO'], MW_CO)
```

Python

ทำหน้าที่แปลงหน่วยความเข้มข้นของก๊าซ NO2, SO2 และ CO จากหน่วย ppm และ ppb เป็นหน่วย µg/m³ ซึ่งเป็นหน่วยที่ใช้กันทั่วไปในการวัดมลพิษทางอากาศ

วัตถุประสงค์:

- **ทำให้ข้อมูลมีความสอดคล้องกัน:** ปรับหน่วยของข้อมูลให้เป็นมาตรฐานเดียวกันเพื่อความสะดวกในการวิเคราะห์และเปรียบเทียบ
- **เตรียมข้อมูลสำหรับการวิเคราะห์เชิงลึก:** ข้อมูลในหน่วย µg/m³ สามารถนำไปใช้ในการวิเคราะห์เชิงลึก เช่น การสร้างกราฟ การวิเคราะห์ความสัมพันธ์ระหว่างตัวแปรต่างๆ ได้อย่างมีประสิทธิภาพ

ประโยชน์:

- ช่วยให้การวิเคราะห์ข้อมูลมีความแม่นยำและเชื่อถือได้มากขึ้น
- ทำให้การนำเสนอผลการวิเคราะห์เป็นไปอย่างชัดเจนและเข้าใจง่ายขึ้น

```python
df.Air_Quality.value_counts()
```

```
Air_Quality
ดี                146
ปานกลาง            14
แย่                14
อันตรายต่อสุขภาพ    3
Name: count, dtype: int64
```

นับจำนวนครั้งที่เหลือ ที่แต่ละค่าคุณภาพอากาศ (Air_Quality) ปรากฏใน DataFrame และแสดงผลลัพธ์ออกมาในรูปแบบของ Series

```python
le = LabelEncoder()
df['Air_Quality'] = le.fit_transform(df['Air_Quality'])

X = df.drop('Air_Quality', axis=1)
y = df['Air_Quality']

scaler = StandardScaler()
X = scaler.fit_transform(X)
```

ดำเนินการเตรียมข้อมูลเพื่อนำไปใช้ในการฝึกสอนโมเดล Machine Learning โดยมีขั้นตอนดังนี้

1. แปลงค่าคุณภาพอากาศ (Air_Quality) ให้เป็นตัวเลขเพื่อให้โมเดลสามารถประมวลผลได้
2. แบ่งข้อมูลออกเป็นชุดข้อมูลต้นแบบ (X) และชุดข้อมูลเป้าหมาย (y)
3. มาตรฐานข้อมูลในชุดข้อมูลต้นแบบ (X) เพื่อปรับปรุงประสิทธิภาพของโมเดล

ประโยชน์:

- **การแปลงข้อมูล:** ช่วยให้โมเดลสามารถประมวลผลข้อมูลที่มีค่าเป็นหมวดหมู่ได้
- **การแบ่งข้อมูล:** ช่วยแยกแยะระหว่างข้อมูลที่ใช้ในการฝึกสอนโมเดลและข้อมูลที่ใช้ในการทดสอบโมเดล
- **การมาตรฐานข้อมูล:** ช่วยปรับปรุงประสิทธิภาพของโมเดลโดยเฉพาะอย่างยิ่งกับโมเดลที่ไวต่อสเกลของข้อมูล (เช่น โมเดล K-Nearest Neighbors)

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

แบ่งข้อมูล X และ y ออกเป็นสองส่วน โดย 80% ของข้อมูลจะถูกใช้สำหรับฝึกสอนโมเดล และ 20% ของข้อมูลจะถูกใช้สำหรับทดสอบโมเดล การแบ่งข้อมูลในลักษณะนี้มีความสำคัญในการประเมินผลการทำงานของโมเดลอย่างเป็นกลางและป้องกันการ overfitting

ประโยชน์:

- **ประเมินผลการทำงานของโมเดล:** ช่วยประเมินว่าโมเดลสามารถทำนายข้อมูลที่ไม่เคยเห็นมาก่อนได้ดีเพียงใด
- **ป้องกัน overfitting:** ช่วยป้องกันปัญหา overfitting ซึ่งเกิดขึ้นเมื่อโมเดลเรียนรู้ข้อมูลฝึกสอนได้ดีเกินไปจนไม่สามารถทำนายข้อมูลใหม่ได้อย่างถูกต้อง
- **เลือกโมเดลที่ดีที่สุด:** ช่วยในการเลือกโมเดลที่มีประสิทธิภาพที่ดีที่สุด

```python
grid_params = {
    'n_neighbors': list(range(1,100)),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
}


gs = GridSearchCV(KNeighborsClassifier(), grid_params, verbose=3, cv=10, n_jobs=-1)
gs_results = gs.fit(X_train, y_train)
knn = gs_results.best_estimator_
```

ใช้ GridSearchCV เพื่อค้นหาค่าของพารามิเตอร์ต่างๆ ที่เหมาะสมที่สุดสำหรับโมเดล KNN โดยทำการประเมินประสิทธิภาพของโมเดลบนชุดข้อมูลฝึกสอน (X_train, y_train) ด้วยการแบ่งข้อมูลแบบ Cross-validation ผลลัพธ์ที่ได้คือโมเดล KNN ที่มีประสิทธิภาพดีที่สุด

```
knn
```

```
KNeighborsClassifier
KNeighborsClassifier(metric='manhattan', n_neighbors=4, weights='distance')
```

แสดงให้เห็นโมเดล KNN ที่ได้รับการปรับแต่งโดยมีพารามิเตอร์ที่เหมาะสมที่สุด ซึ่งจะถูกนำไปใช้ในการทำนายค่าของข้อมูลใหม่

ประโยชน์:

- **เข้าใจโครงสร้างของโมเดล:** ช่วยให้เข้าใจรายละเอียดของโมเดล KNN ที่ได้รับการฝึกสอน เช่น วิธีการวัดระยะห่าง จำนวนเพื่อนบ้าน และการกำหนดน้ำหนัก
- **ใช้โมเดลในการทำนาย:** สามารถนำโมเดลนี้ไปใช้ในการทำนายค่าของข้อมูลใหม่ได้โดยตรง

```python
y_pred = knn.predict(X_test)
```

Python

ใช้โมเดล KNN ที่ได้รับการฝึกสอนแล้วเพื่อทำนายค่าของชุดข้อมูลทดสอบ (X_test) ผลลัพธ์การทำนายจะถูกเก็บไว้ในตัวแปร `y_pred` ซึ่งจะนำไปใช้ในการประเมินผลการทำงานของโมเดลต่อไป

ประโยชน์:

- **ประเมินผลการทำงานของโมเดล:** ใช้ค่าทำนาย (`y_pred`) เพื่อเปรียบเทียบกับค่าจริง (`y_test`) เพื่อประเมินความแม่นยำของโมเดล
- **ใช้โมเดลในการทำนายข้อมูลใหม่:** สามารถนำโมเดลนี้ไปใช้ในการทำนายค่าของข้อมูลใหม่ที่ไม่เคยเห็นมาก่อน

```python
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Python

```
Accuracy: 0.9166666666666666
```

คำนวณค่าความแม่นยำ (accuracy) ของโมเดล KNN โดยเปรียบเทียบค่าทำนาย (`y_pred`) กับค่าจริง (`y_test`) และแสดงผลลัพธ์บนหน้าจอ

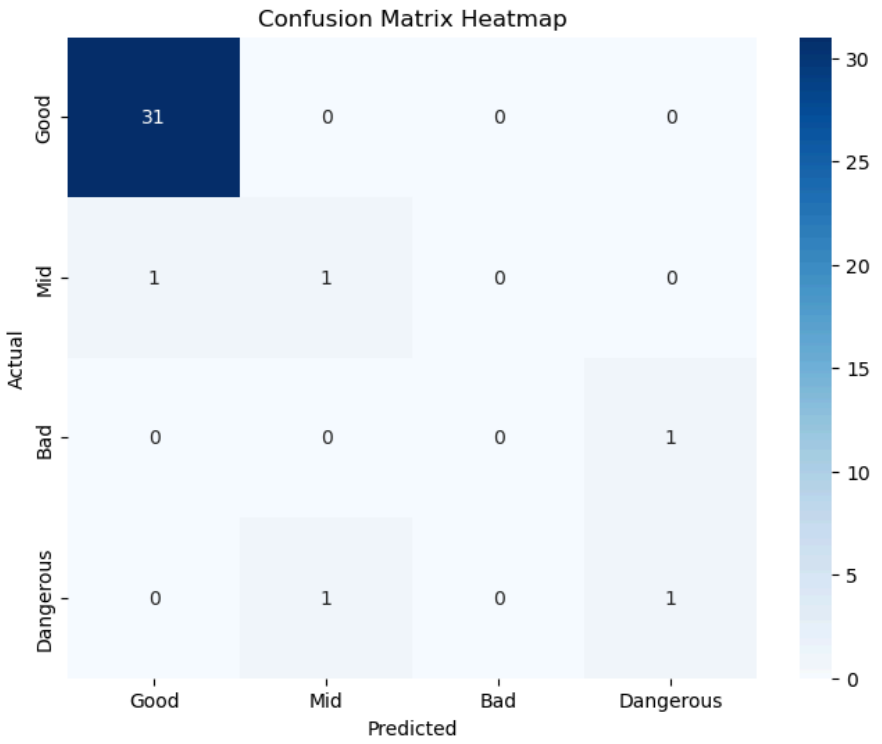
Accuracy = 0.91666

```python
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        31
           1       0.50      0.50      0.50         2
           2       0.00      0.00      0.00         1
           3       0.50      0.50      0.50         2

    accuracy                           0.92        36
   macro avg       0.49      0.50      0.50        36
weighted avg       0.89      0.92      0.90        36
```

แสดงรายงานสรุปผลการจำแนกประเภทของโมเดล KNN โดยรายงานจะแสดงค่า precision, recall, f1-score, support และค่าความแม่นยำโดยรวมสำหรับแต่ละคลาสและโดยรวม ซึ่งช่วยในการประเมินประสิทธิภาพของโมเดลอย่างละเอียด

```python
cm = confusion_matrix(y_test, y_pred)
# class_labels = le.classes_
class_labels = ['Good', 'Mid', 'Bad', 'Dangerous']


plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix Heatmap")
plt.show()
```



สร้าง Confusion Matrix Heatmap เพื่อแสดงผลการจำแนกประเภทของโมเดล KNN โดยใช้สีที่เข้มแสดงจำนวนข้อมูลมาก สีจางแสดงจำนวนข้อมูลน้อย การวิเคราะห์ Confusion Matrix Heatmap ช่วยให้เข้าใจประสิทธิภาพของโมเดลได้ดียิ่งขึ้น เช่น

- ตรวจสอบว่าโมเดลมีการทำนายผิดพลาดในคลาสใดบ้าง
- ประเมินประสิทธิภาพของการจำแนกแต่ละคลาส

สรุป Accuracy ที่ได้คือ 0.92