Sofia Lopez
2/3/2026
CIVE 202

ACD

Lines of Code/ Functions	Explanations
import pandas as pd	The statement import pandas as pd imports the pandas library into Python and assigns it the name pd so its data analysis functions can be used throughout the program.
df = pd.read_csv("AirQuality_Daily_StudentVersion.csv")	The function pd.read_csv function reads the CSV file and stores the dataset in a pandas DataFrame called df, which allows our data to be organized and analyzed.
summary = df.groupby("sensor.name").agg({ "voc": ["mean", "median"], "pm2.5_atm": ["mean", "median"], "pm10.0_atm": ["mean", "median"]})	Using the groupby(), it groups the dataset by sensor location so that statistics can be calculated for each sensor. The agg() function computes the mean and median values for VOC, PM2.5, and PM10 and stores them in a summary table
top5_voc_mean = summary.sort_values(("voc", "mean"), ascending=False).head(5)	The sort_values() function orders the results from highest to lowest, and head(5) selects the top five sensors with the largest VOC mean values. This is the VOC Table:
top5_voc_median = summary.sort_values(("voc", "median"), ascending=False).head(5)	This line repeats the sorting process for VOC median values
top5_pm25_mean = summary.sort_values(("pm2.5_atm", "mean"), ascending=False).head(5)	The sort_values() function orders the results from highest to lowest, and head(5) selects the top five sensors with the largest pm2.5 mean values. This is the Pm2.5 table:
top5_pm25_median = summary.sort_values(("pm2.5_atm", "median"), ascending=False).head(5)	This line repeats the sorting process for pm2.5 median values.

top5_pm10_mean = summary.sort_values(("pm10.0_atm", "mean"), ascending=False).head(5)	The sort_values() function orders the results from highest to lowest, and head(5) selects the top five sensors with the largest pm10 mean values. #This is the pm10 Table:
top5_pm10_median = summary.sort_values(("pm10.0_atm", "median"), ascending=False).head(5)	This line repeats the sorting process for pm2.5 median values.

display(top5_voc_mean[[("voc", "mean")]])
display(top5_voc_median[[("voc", "median")]])
display(top5_pm25_mean[[("pm2.5_atm", "mean")]])
display(top5_pm25_median[[("pm2.5_atm", "median")]])
display(top5_pm10_mean[[("pm10.0_atm", "mean")]])
display(top5_pm10_median[[("pm10.0_atm", "median")]])
	The display() function prints each table in order to read the highest pollution values.

summary_max = pd.DataFrame({	In this line we create a new df that summarizes maximum values of each pollutant, along with the date and location.
"pollutant": ["voc", "pm2.5", "pm10"],	This line labels the pollutants that will appear in the summary

"date": [
        df.loc[df["voc"].idxmax(), "date"],
        df.loc[df["pm2.5_atm"].idxmax(), "date"],
        df.loc[df["pm10.0_atm"].idxmax(), "date"]
	These lines find the date the pollutant reached its max. The idmax() function find the row index of the largest value, and the loc[] function retrieves the date stored in that same row.

"location": [
        df.loc[df["voc"].idxmax(), "sensor.name"],
        df.loc[df["pm2.5_atm"].idxmax(), "sensor.name"],
        df.loc[df["pm10.0_atm"].idxmax(), "sensor.name"]
    ],
	The loc[] function is used again to pull the sensor name.

"value": [df["voc"].max(), df["pm2.5_atm"].max(), df["pm10.0_atm"].max()	The max() function calculates the highest recorded value in each column.
display(summary_max)	This display command will display the completed summary/table.

def humidity_category(h):
if h < 50:
return "Low"
elif h <= 80:
return "High"
else:
return "Very High"
	This function will "def" define our humidity categories by assigning a label based on humidity values passed into the function. The else command means that if the value does not meet the prior restrictions, it auto-groups into the very high category.

df["humidity_cat"] = df["humidity"].apply(humidity_category)	The apply function will run the humidity category on every value in the humidity column and the temp category and can store the result in a new column.

def temp_category(t):
if t < 32:
return "Below freezing"
elif t <= 50:
return "Cool"
elif t <= 70:
return "Warm"
else:
return "Hot"
df["temp_cat"] = df["temperature"].apply(temp_category)
	The def function will define the category temperature by assigning a label based on the temp. value passed into the function.

df["temp_cat"] = df["temperature"].apply(temp_category)	The apply() function runs temp_category on every value then it will store it in a new column.
summary = df.groupby(["sensor.name", "humidity_cat", "temp_cat"])[["voc","pm2.5_atm","pm10.0_atm"]].mean()	The groupby() function will organize the data by sensor,humidity_cat, and temp_Cat, and will prepare them for summary analysis.
display(summary)	This command displays our grouped summary table.

df["pm25_risk"] = df["pm2.5_atm"] >= 35.5	This line creates a new column that flags days where PM2.5 reaches the EPA threshold for unhealthy air quality, based on table client requests.
df["pm10_risk"] = df["pm10.0_atm"] >= 155	This line creates a new column that flags days where PM10 reaches the unhealthy air quality threshold.
dates_risk = df[(df["pm25_risk"]) | (df["pm10_risk"])]	This command filters the dataset to keep only rows where either PM2.5 or PM10 is considered a risk.
risk_summary = dates_risk.groupby("sensor.name")[["pm2.5_atm", "pm10.0_atm"]].max()	This groupby operation summarizes the maximum pollutant values for each sensor on risk days
display(risk_summary)	This command displays the summarized risk table.
display(dates_risk[["date", "sensor.name", "pm2.5_atm", "pm10.0_atm"]])	This command displays the exact dates and locations where unhealthy air quality events occurred.