Here are several improvements that can be made to the provided code:

1. **Modularization**: Encapsulate functionality into functions for better organization and reusability.
2. **Use of Context Managers**: Utilize **with** statements when reading files to ensure proper resource management.
3. **Improved Data Handling**: Use **.dropna()** with the **inplace=True** parameter to avoid reassignment.
4. **Enhanced Data Inspection**: Include more informative prints and visualizations for better data understanding.
5. **Error Handling**: Add basic error handling when reading the CSV file.
6. **Visualization of Missing Values**: Use a visualization library to plot missing values for better insight.
7. **Consistent Variable Naming**: Use more descriptive variable names where applicable.

Here’s the improved code:

python

Insert CodeEditCopy code

1import numpy as np

2import pandas as pd

3import matplotlib.pyplot as plt

4import seaborn as sns

5import datetime

6from warnings import filterwarnings

7filterwarnings('ignore')

8

9def load\_data(filepath):

10 """Load dataset from a CSV file."""

11 try:

12 with open(filepath, 'r') as file:

13 df = pd.read\_csv(file)

14 print("Data loaded successfully.")

15 return df

16 except Exception as e:

17 print(f"Error loading data: {e}")

18 return None

19

20def inspect\_data(df):

21 """Display basic information about the dataset."""

22 print(df.head())

23 print("Dataset Shape:", df.shape)

24 print("Dataset Information:")

25 print(df.info())

26 print("Missing Values:")

27 print(df.isnull().sum())

28 print("Duplicate Values:", df.duplicated().sum())

29

30def clean\_data(df):

31 """Clean the dataset by dropping rows with missing AQI values."""

32 df.dropna(subset=['AQI'], inplace=True)

33 print("Missing Values after dropping rows:")

34 print(df.isnull().sum().sort\_values(ascending=False))

35 print("Dataset Shape after dropping rows:", df.shape)

36

37def summarize\_data(df):

38 """Display summary statistics and visualize missing values."""

39 print("Summary Statistics:")

40 print(df.describe().T)

41 null\_values\_percentage = (df.isnull().sum()/df.isnull().count()\*100).sort\_values(ascending=False)

42 print("Percentage of Missing Values:")

43 print(null\_values\_percentage)

44

45 # Visualizing missing values

46 plt.figure(figsize=(10, 6))

47 sns.heatmap(df.isnull(), cbar=False, cmap='viridis')

48 plt.title('Missing Values Heatmap')

49 plt.show()

50

51# Main execution

52if \_\_name\_\_ == "\_\_main\_\_":

53 filepath = 'C:\\VS Code\\Python\\air quality data-Copy1.csv'

54 df = load\_data(filepath)

55

56 if df is not None:

57 inspect\_data(df)

58 clean\_data(df)

59 summarize\_data(df)

**Improvements Explained:**

1. **Modularization**: Functions are created for loading, inspecting, cleaning, and summarizing the data, making it easier to maintain and test.
2. **Use of Context Managers**: The **with** statement ensures the file is properly closed after loading.
3. **In-place Data Cleaning**: Using **inplace=True** in **dropna()** avoids unnecessary reassignment.
4. **Enhanced Data Inspection**: The function **summarize\_data** includes a heatmap to visualize missing values, which adds a visual dimension to data inspection.
5. **Error Handling**: Basic error handling is included to manage exceptions when loading the data.
6. **Consistent Variable Naming**: The code uses descriptive function names to clarify their purpose.

These changes collectively improve the readability, maintainability, and functionality of the code.

**Changes Made:**

1. **Function Creation**: Created a function **plot\_histogram** to handle histogram plotting, which reduces code repetition.
2. **Improved Plotting**: Added titles and axis labels to histograms for clarity.
3. **Loop for Histograms**: Used a loop to iterate through the columns for histogram plotting instead of repeating the code for each column.
4. **Consistent Style**: Set the seaborn style only once at the beginning.
5. **Legend Handling**: Removed unnecessary legends for histograms since they are not required.
6. **Code Comments**: Added comments for clarity.

**Improved Code:**

python

VerifyOpen In EditorEditCopy code

1import matplotlib.pyplot as plt

2import seaborn as sns

3

4# Set the style for seaborn

5sns.set(style="darkgrid")

6

7# Function to plot histograms

8def plot\_histogram(data, column):

9 plt.figure(figsize=(10, 5))

10 data[column].plot(kind='hist', edgecolor='black')

11 plt.title(f'Distribution of {column}')

12 plt.xlabel(column)

13 plt.ylabel('Frequency')

14 plt.show()

15

16# List of columns to plot histograms

17columns\_to\_plot = ['Xylene', 'PM10', 'NH3', 'Toluene', 'Benzene',

18 'NOx', 'O3', 'PM2.5', 'SO2', 'CO', 'NO2', 'NO', 'AQI']

19

20# Loop through columns to create histograms

21for column in columns\_to\_plot:

22 plot\_histogram(df, column)

23

24# Distribution of AQI from 2015-2020

25plt.figure(figsize=(10, 5))

26sns.displot(df, x="AQI", color="purple")

27plt.title('Distribution of AQI from 2015-2020')

28plt.show()

29

30# Count plots for City

31graph = sns.catplot(x="City", kind="count", data=df, height=5, aspect=3)

32graph.set\_xticklabels(rotation=90)

33plt.title('Count of Entries by City')

34plt.show()

35

36# Count plots for City by AQI Bucket

37graph = sns.catplot(x="City", kind="count", data=df, col="AQI\_Bucket", col\_wrap=2, height=3.5, aspect=3)

38graph.set\_xticklabels(rotation=90)

39plt.title('Count of Entries by City and AQI Bucket')

40plt.show()

41

42# Box plots for different pollutants

43pollutants = ['PM2.5', 'NO2', 'O3', 'SO2', 'NOx', 'NO']

44for pollutant in pollutants:

45 graph = sns.catplot(x="City", y=pollutant, kind="box", data=df, height=5, aspect=3)

46 graph.set\_xticklabels(rotation=90)

47 plt.title(f'Box Plot of {pollutant} by City')

48 plt.show()

49

50# Count plot for AQI Bucket

51graph = sns.catplot(x="AQI\_Bucket", data=df, kind="count", height=6, aspect=3)

52graph.set\_xticklabels(rotation=90)

53plt.title('Count of Entries by AQI Bucket')

54plt.show()

**Key Benefits of Changes:**

* **Readability**: The code is cleaner and easier to understand.
* **Maintainability**: If you need to add more columns to the histogram or box plots, you can simply update the list instead of duplicating code.
* **Clarity**: Each plot now has a title and labels, making it clear what is being displayed.

**Improvements Made:**

1. **Function Creation**: Created a function **evaluate\_model** to handle model training, prediction, and evaluation. This reduces code duplication.
2. **Consistent Variable Naming**: Ensured consistent variable names for predictions and RMSE calculations across different models.
3. **Removed Redundant Code**: Removed repeated code segments for each model's training and evaluation.
4. **Use of train\_test\_split**: Added the import statement for **train\_test\_split** from **sklearn.model\_selection** since it was used but not imported.
5. **Consolidated Imports**: Grouped all necessary imports at the top for better organization.
6. **Improved Output Formatting**: Enhanced the print statements for better clarity and organization.
7. **Added Comments**: Included comments to explain different sections of the code for better understanding.

**Improved Code:**

python

VerifyOpen In EditorEditCopy code

1import numpy as np

2import pandas as pd

3from sklearn.model\_selection import train\_test\_split

4from sklearn.linear\_model import LinearRegression

5from sklearn.metrics import mean\_squared\_error

6from sklearn.tree import DecisionTreeRegressor

7from sklearn.ensemble import RandomForestRegressor

8from sklearn.neighbors import KNeighborsRegressor

9

10# Data Preparation for Modeling

11x = df[["PM2.5", "PM10", "NO", "NO2", "NOx", "NH3", "CO", "SO2", "O3", "Benzene", "Toluene", "Xylene"]]

12y = df["AQI"]

13

14# Splitting the data into training and testing data

15X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=70)

16print(f"Training set shape: {X\_train.shape}, Test set shape: {X\_test.shape}")

17

18# Function to evaluate models

19def evaluate\_model(model, X\_train, Y\_train, X\_test, Y\_test):

20 model.fit(X\_train, Y\_train)

21 train\_pred = model.predict(X\_train)

22 test\_pred = model.predict(X\_test)

23

24 RMSE\_train = np.sqrt(mean\_squared\_error(Y\_train, train\_pred))

25 RMSE\_test = np.sqrt(mean\_squared\_error(Y\_test, test\_pred))

26

27 print(f"Model: {model.\_\_class\_\_.\_\_name\_\_}")

28 print(f"RMSE Training Data: {RMSE\_train:.2f}")

29 print(f"RMSE Test Data: {RMSE\_test:.2f}")

30 print(f'R-Squared value on train: {model.score(X\_train, Y\_train):.2f}')

31 print(f'R-Squared value on test: {model.score(X\_test, Y\_test):.2f}')

32 print('-' \* 50)

33

34# Evaluate different models

35models = {

36 "Linear Regression": LinearRegression(),

37 "K-Neighbors Regressor": KNeighborsRegressor(),

38 "Decision Tree Regressor": DecisionTreeRegressor(),

39 "Random Forest Regressor": RandomForestRegressor()

40}

41

42for model\_name, model in models.items():

43 evaluate\_model(model, X\_train, Y\_train, X\_test, Y\_test)

**Key Benefits of Changes:**

* **Reduced Redundancy**: The use of a function significantly reduces repeated code, making it easier to manage and update.
* **Improved Readability**: The code is more organized and easier to read, with clear sections for data preparation, model evaluation, and results.
* **Easier Maintenance**: If you need to add or modify models, you can do so in a single location without affecting the overall structure.
* **Enhanced Clarity**: The output is formatted for better readability, making it easier to interpret the results.