TASK 1- By Joan Akshita

1. Data preprocessing and visualisation (cleaning , undertstanding the data and EDA)- aggregation, wrangling,one hot encoding, scaling, feature engineering,etc, everything relevant to the data is done .

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer


# Read the CSV file
df = pd.read_csv("/content/PRSA_data_2010.1.1-2014.12.31.csv")


# Display basic info
print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43824 entries, 0 to 43823
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   No      43824 non-null  int64
 1   year    43824 non-null  int64
 2   month   43824 non-null  int64
 3   day     43824 non-null  int64
 4   hour    43824 non-null  int64
 5   pm2.5   41757 non-null  float64
 6   DEWP    43824 non-null  int64
 7   TEMP    43824 non-null  float64
 8   PRES    43824 non-null  float64
```

```
 9   cbwd    43824 non-null  object
 10  Iws     43824 non-null  float64
 11  Is      43824 non-null  int64
 12  Ir      43824 non-null  int64
dtypes: float64(4), int64(8), object(1)
memory usage: 4.3+ MB
None
   No  year  month  day  hour  pm2.5  DEWP  TEMP    PRES cbwd    Iws  Is  Ir
0   1  2010      1    1     0    NaN   -21 -11.0  1021.0   NW   1.79   0   0
1   2  2010      1    1     1    NaN   -21 -12.0  1020.0   NW   4.92   0   0
2   3  2010      1    1     2    NaN   -21 -11.0  1019.0   NW   6.71   0   0
3   4  2010      1    1     3    NaN   -21 -14.0  1019.0   NW   9.84   0   0
4   5  2010      1    1     4    NaN   -20 -12.0  1018.0   NW  12.97   0   0
```

```
#DATA WRANGLING
# Drop unnecessary columns like 'No' (index) & original date parts
df.drop(columns=["No"], inplace=True)


# Check for missing values
print(df.isnull().sum())
```

```
year        0
month       0
day         0
hour        0
pm2.5    2067
DEWP        0
TEMP        0
PRES        0
cbwd        0
Iws         0
Is          0
Ir          0
dtype: int64
```

```
# Handling Missing Values in PM2.5 (Time-Series Interpolation)
df["pm2.5"] = df["pm2.5"].interpolate(method='linear')
```

```
# Check if missing values are filled
print(df.isnull().sum())
```

```
year      0
month     0
day       0
hour      0
pm2.5    24
DEWP      0
TEMP      0
PRES      0
cbwd      0
Iws       0
Is        0
Ir        0
dtype: int64
```

```
# If still missing, fill with median
df["pm2.5"].fillna(df["pm2.5"].median(), inplace=True)

# Verify missing values
print("Missing values after handling:\n", df.isnull().sum())
```

```
Missing values after handling:
 year     0
month    0
day      0
hour     0
pm2.5    0
DEWP     0
TEMP     0
PRES     0
cbwd     0
Iws      0
Is       0
Ir       0
dtype: int64
<ipython-input-9-5639a15389f3>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
```

```
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett

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

  df["pm2.5"].fillna(df["pm2.5"].median(), inplace=True)
```

```
###  3. AGGREGATION ###
# Aggregate data to get daily and monthly mean PM2.5 levels
daily_df = df.groupby(["year", "month", "day"]).agg({"pm2.5": "mean", "TEMP": "mean", "PRES": "mean"}).reset_index()
monthly_df = df.groupby(["year", "month"]).agg({"pm2.5": "mean", "TEMP": "mean", "PRES": "mean"}).reset_index()

print("\nAggregated Daily Data:\n", daily_df.head())
print("\nAggregated Monthly Data:\n", monthly_df.head())
```

```
Aggregated Daily Data:
    year  month  day       pm2.5       TEMP         PRES
0  2010      1    1   73.000000  -6.750000  1017.083333
1  2010      1    2  145.958333  -5.125000  1024.750000
2  2010      1    3   78.833333  -8.541667  1022.791667
3  2010      1    4   31.333333 -11.500000  1029.291667
4  2010      1    5   42.458333 -14.458333  1033.625000

Aggregated Monthly Data:
    year  month      pm2.5       TEMP         PRES
0  2010      1  86.688172  -6.162634  1028.009409
1  2010      2  98.264137  -1.922619  1023.776786
2  2010      3  98.886425   3.293011  1021.811828
3  2010      4  79.884722  10.806944  1017.169444
4  2010      5  86.910618  20.831989  1007.896505
```

```
# Feature Engineering - Creating a 'Season' Column
def get_season(month):
    if month in [12, 1, 2]:
        return "Winter"
    elif month in [3, 4, 5]:
```

```
        return "Spring"
    elif month in [6, 7, 8]:
        return "Summer"
    else:
        return "Autumn"

df["season"] = df["month"].apply(get_season)


# Convert year, month, day, hour into a datetime format
df["datetime"] = pd.to_datetime(df[["year", "month", "day", "hour"]])
df["season"] = df["datetime"].dt.month.apply(get_season)

# Extract time-based features
df["hour"] = df["datetime"].dt.hour
df["day_of_week"] = df["datetime"].dt.dayofweek
df["month"] = df["datetime"].dt.month

# Drop the datetime column after extracting features
df.drop(columns=["datetime"], inplace=True)


# One-Hot Encoding Categorical Features (Wind Direction & Season)
encoder = OneHotEncoder(sparse_output=False, drop="first")
encoded_features = encoder.fit_transform(df[['cbwd', 'season']])
encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_out(['cbwd', 'season']))


# Drop original categorical columns and merge new features
df = df.drop(columns=['cbwd', 'season'])
df = pd.concat([df, encoded_df], axis=1)


# Splitting into Train and Test Sets
X = df.drop(columns=["pm2.5"])  # Drop target
y = df["pm2.5"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Final check
print("Training Set Shape:", X_train.shape)
print("Test Set Shape:", X_test.shape)
```

```
Training Set Shape: (35059, 17)
Test Set Shape: (8765, 17)
```

TASK 2 - By Harshita Khudania.

2. To build classification models (SVM, Logistic Regression, Random Forest, KNN) to predict PM2.5 levels, evaluated them using accuracy, precision, recall, and applied PCA to reduce dimensions. After retraining models on PCA-transformed data, analyzed the impact on performance, tuned KNN for the best k-value, and visualized results with confusion matrices, classification reports, and ROC curves. Debugging ensured correct feature alignment between models and test datasets

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, classification_report,
    confusion_matrix
)
```

```
df = pd.read_csv("/content/PRSA_data_2010.1.1-2014.12.31.csv")
```

```
df.tail()
```

	No	year	month	day	hour	pm2.5	DEWP	TEMP	PRES	cbwd	Iws	Is	Ir
43819	43820	2014	12	31	19	8.0	-23	-2.0	1034.0	NW	231.97	0	0
43820	43821	2014	12	31	20	10.0	-22	-3.0	1034.0	NW	237.78	0	0
43821	43822	2014	12	31	21	10.0	-22	-3.0	1034.0	NW	242.70	0	0
43822	43823	2014	12	31	22	8.0	-22	-4.0	1034.0	NW	246.72	0	0
43823	43824	2014	12	31	23	12.0	-21	-3.0	1034.0	NW	249.85	0	0

```
df.shape
```

(43824, 13)

```
df.describe()
```

	No	year	month	day	hour	pm2.5	DEWP	TEMP	PRES
count	43824.000000	43824.000000	43824.000000	43824.000000	43824.000000	41757.000000	43824.000000	43824.000000	43824.000000
mean	21912.500000	2012.000000	6.523549	15.727820	11.500000	98.613215	1.817246	12.448521	1016.447654
std	12651.043435	1.413842	3.448572	8.799425	6.922266	92.050387	14.433440	12.198613	10.268698
min	1.000000	2010.000000	1.000000	1.000000	0.000000	0.000000	-40.000000	-19.000000	991.000000
25%	10956.750000	2011.000000	4.000000	8.000000	5.750000	29.000000	-10.000000	2.000000	1008.000000
50%	21912.500000	2012.000000	7.000000	16.000000	11.500000	72.000000	2.000000	14.000000	1016.000000
75%	32868.250000	2013.000000	10.000000	23.000000	17.250000	137.000000	15.000000	23.000000	1025.000000
max	43824.000000	2014.000000	12.000000	31.000000	23.000000	994.000000	28.000000	42.000000	1046.000000

```
df.nunique()
```

	0
No	43824
year	5
month	12
day	31
hour	24
pm2.5	581
DEWP	69
TEMP	64
PRES	60
cbwd	4
Iws	2788
Is	28
Ir	37

dtype: int64

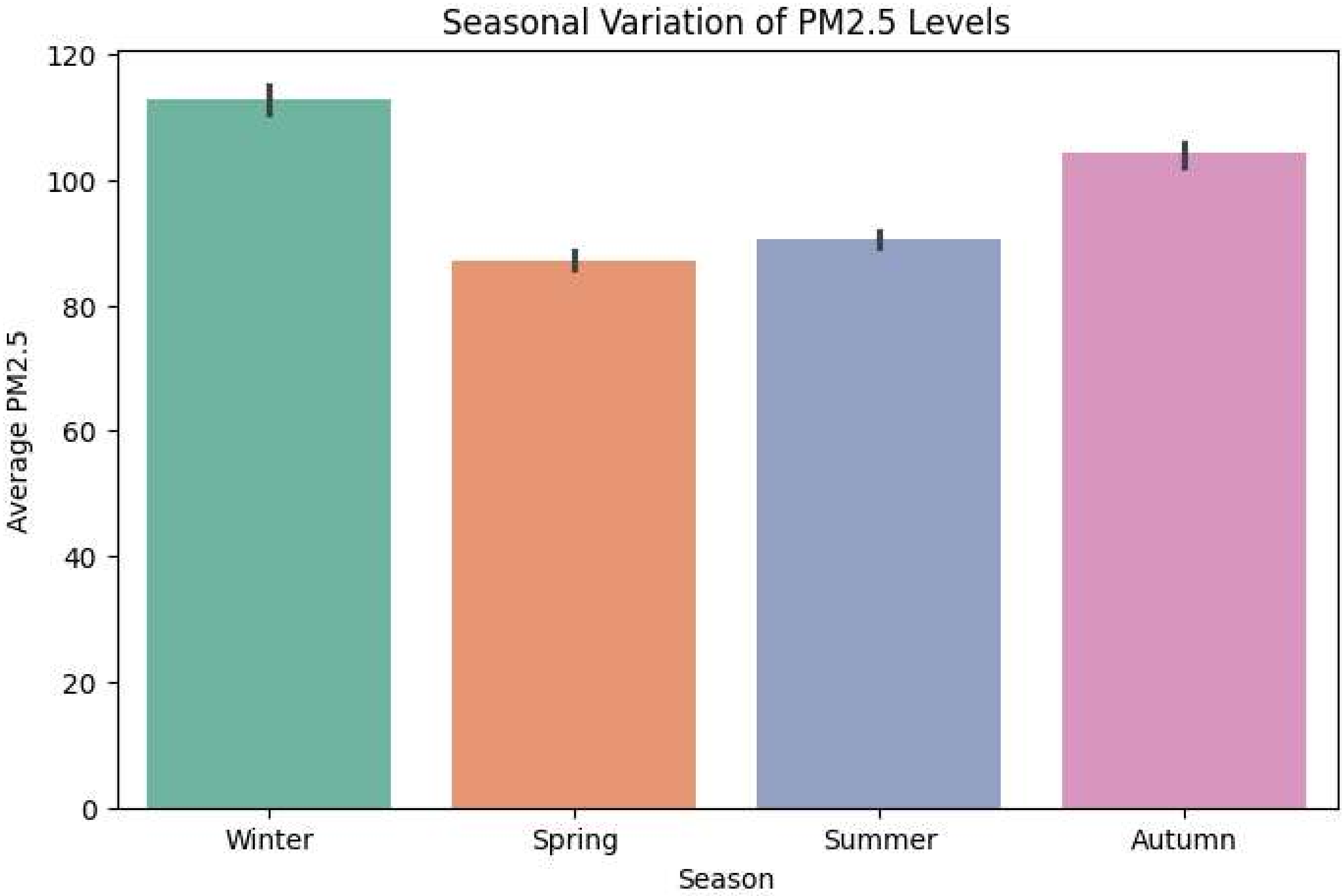
```
df.columns
```

```
Index(['No', 'year', 'month', 'day', 'hour', 'pm2.5', 'DEWP', 'TEMP', 'PRES',
       'cbwd', 'Iws', 'Is', 'Ir'],
      dtype='object')
```

```
# Handling missing values
df = df.dropna()
```

```python
# Categorizing PM2.5 levels (classification)
df['pm2.5_category'] = pd.cut(df['pm2.5'], bins=[-1, 35, 75, np.inf], labels=[0, 1, 2])


df["season"] = df["month"].map({12: "Winter", 1: "Winter", 2: "Winter",
                                3: "Spring", 4: "Spring", 5: "Spring",
                                6: "Summer", 7: "Summer", 8: "Summer",
                                9: "Autumn", 10: "Autumn", 11: "Autumn"})

plt.figure(figsize=(8,5))
sns.barplot(x=df["season"], y=df["pm2.5"], estimator=np.mean, palette="Set2")
plt.xlabel("Season")
plt.ylabel("Average PM2.5")
plt.title("Seasonal Variation of PM2.5 Levels")
plt.show()
```

```
<ipython-input-14-b4fb368560fb>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(x=df["season"], y=df["pm2.5"], estimator=np.mean, palette="Set2")
```

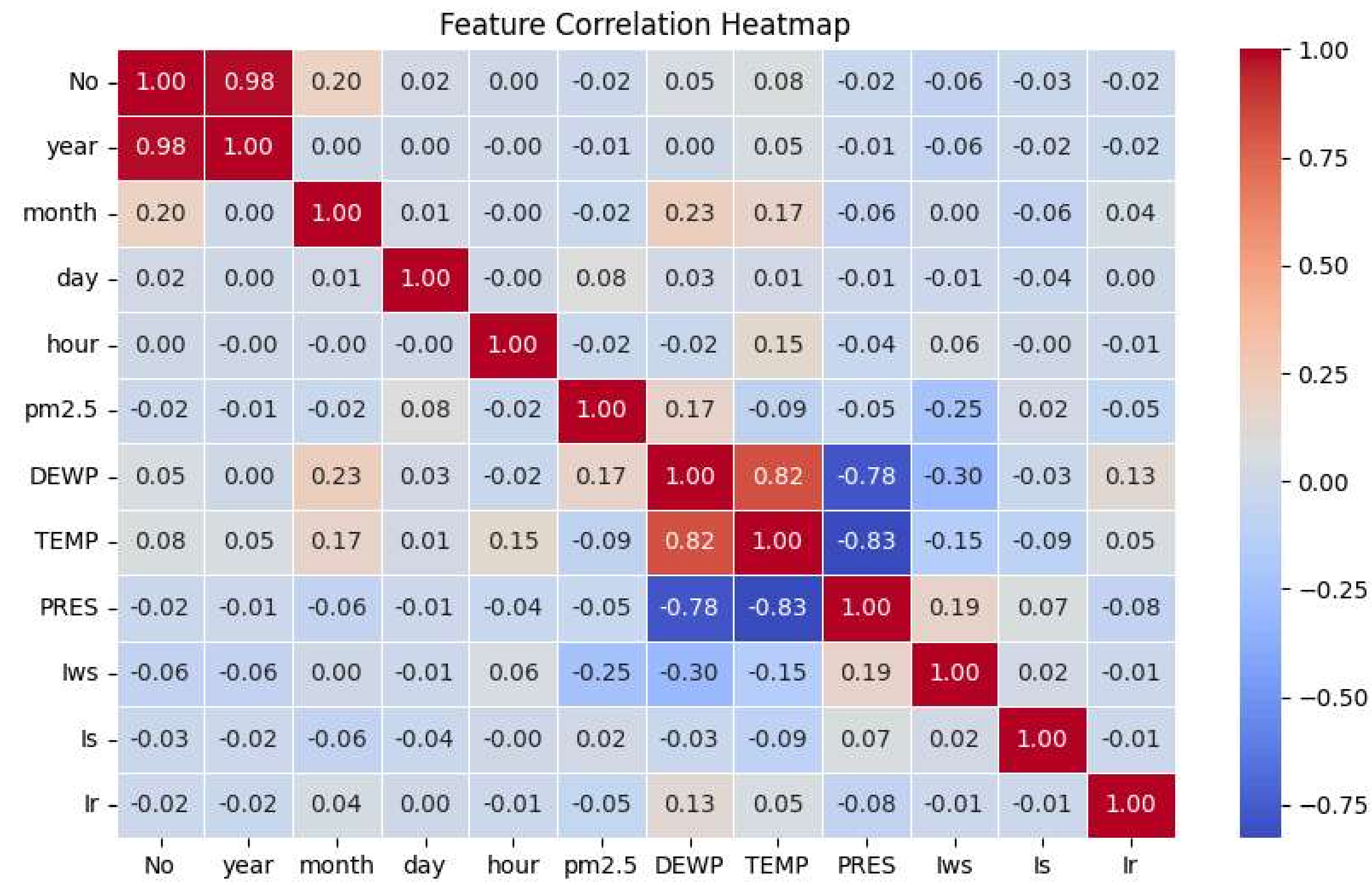



```
correlation_matrix = df.select_dtypes(include=[np.number]).corr()
print(correlation_matrix)
```

```
             No          year         month           day          hour  \
No     1.000000  9.797958e-01  1.993007e-01  1.880803e-02  5.471695e-04
year   0.979796  1.000000e+00  3.070661e-13  3.200526e-15 -3.318729e-15
```

```
month  0.199301  3.070661e-13  1.000000e+00  1.079604e-02 -1.525086e-16
day    0.018808  3.200526e-15  1.079604e-02  1.000000e+00 -5.312012e-17
hour   0.000547 -3.318729e-15 -1.525086e-16 -5.312012e-17  1.000000e+00
pm2.5 -0.017706 -1.469020e-02 -2.406878e-02  8.278849e-02 -2.311644e-02
DEWP   0.047668  1.121574e-03  2.339746e-01  2.855899e-02 -2.098769e-02
TEMP   0.078159  4.552854e-02  1.700926e-01  1.479104e-02  1.500656e-01
PRES  -0.024224 -1.257001e-02 -6.218507e-02 -7.070048e-03 -4.192788e-02
Iws   -0.062427 -6.424368e-02  3.043299e-03 -8.953566e-03  5.661776e-02
Is    -0.029464 -1.700207e-02 -6.167206e-02 -3.682638e-02 -2.373592e-03
Ir    -0.016563 -2.438290e-02  3.673715e-02  2.681328e-03 -6.286241e-03

          pm2.5      DEWP      TEMP      PRES       Iws        Is        Ir
No    -0.017706  0.047668  0.078159 -0.024224 -0.062427 -0.029464 -0.016563
year  -0.014690  0.001122  0.045529 -0.012570 -0.064244 -0.017002 -0.024383
month -0.024069  0.233975  0.170093 -0.062185  0.003043 -0.061672  0.036737
day    0.082788  0.028559  0.014791 -0.007070 -0.008954 -0.036826  0.002681
hour  -0.023116 -0.020988  0.150066 -0.041928  0.056618 -0.002374 -0.006286
pm2.5  1.000000  0.171423 -0.090534 -0.047282 -0.247784  0.019266 -0.051369
DEWP   0.171423  1.000000  0.824633 -0.778346 -0.296399 -0.034410  0.125090
TEMP  -0.090534  0.824633  1.000000 -0.826690 -0.154623 -0.092601  0.049121
PRES  -0.047282 -0.778346 -0.826690  1.000000  0.185355  0.069028 -0.079843
Iws   -0.247784 -0.296399 -0.154623  0.185355  1.000000  0.021883 -0.010122
Is     0.019266 -0.034410 -0.092601  0.069028  0.021883  1.000000 -0.009548
Ir    -0.051369  0.125090  0.049121 -0.079843 -0.010122 -0.009548  1.000000
```

```
plt.figure(figsize=(10,6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```




```
# Selecting features (removing unnecessary columns)
X = df.drop(columns=['No', 'year', 'day', 'pm2.5', 'pm2.5_category'])
y = df['pm2.5_category']
y
```

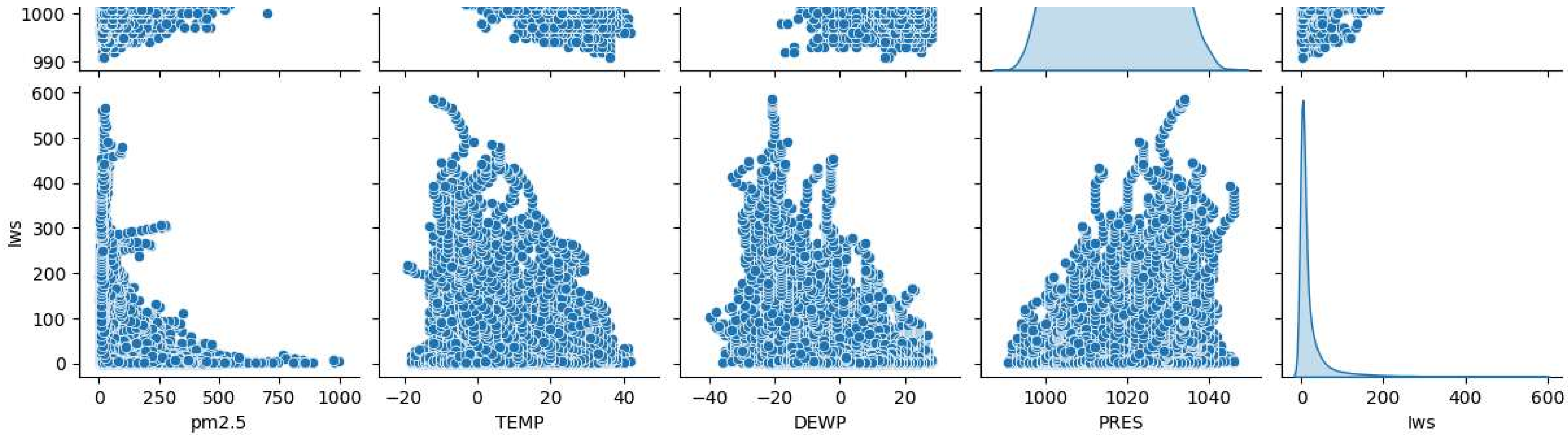
	pm2.5_category
24	2
25	2
26	2
27	2
28	2
...	...
43819	0
43820	0
43821	0
43822	0
43823	0

41757 rows × 1 columns

dtype: category

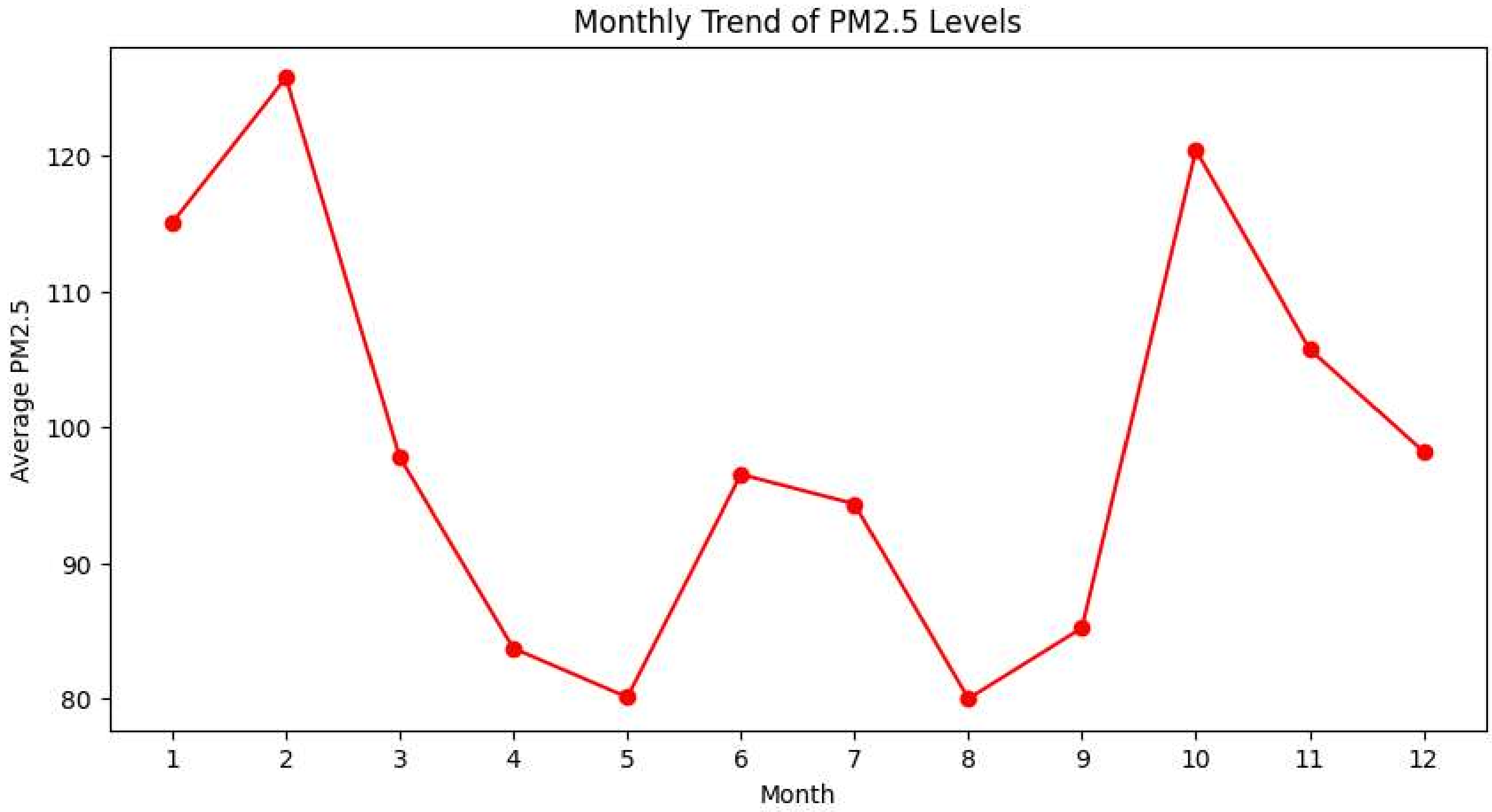
```
sns.pairplot(df[["pm2.5", "TEMP", "DEWP", "PRES", "Iws"]], diag_kind="kde")
plt.show()
```

pm2.5
TEMP
DEWP
PRES
1000
800
600
400
200
0
40
30
20
10
0
−10
−20
30
20
10
0
−10
−20
−30
−40
1040
1030
1020
1010



Iws
pm2.5
TEMP
DEWP
PRES
Iws

```
plt.figure(figsize=(10,5))
df.groupby("month")["pm2.5"].mean().plot(marker="o", linestyle="-", color="red")
plt.xlabel("Month")
plt.ylabel("Average PM2.5")
plt.title("Monthly Trend of PM2.5 Levels")
plt.xticks(range(1,13))
plt.show()
```



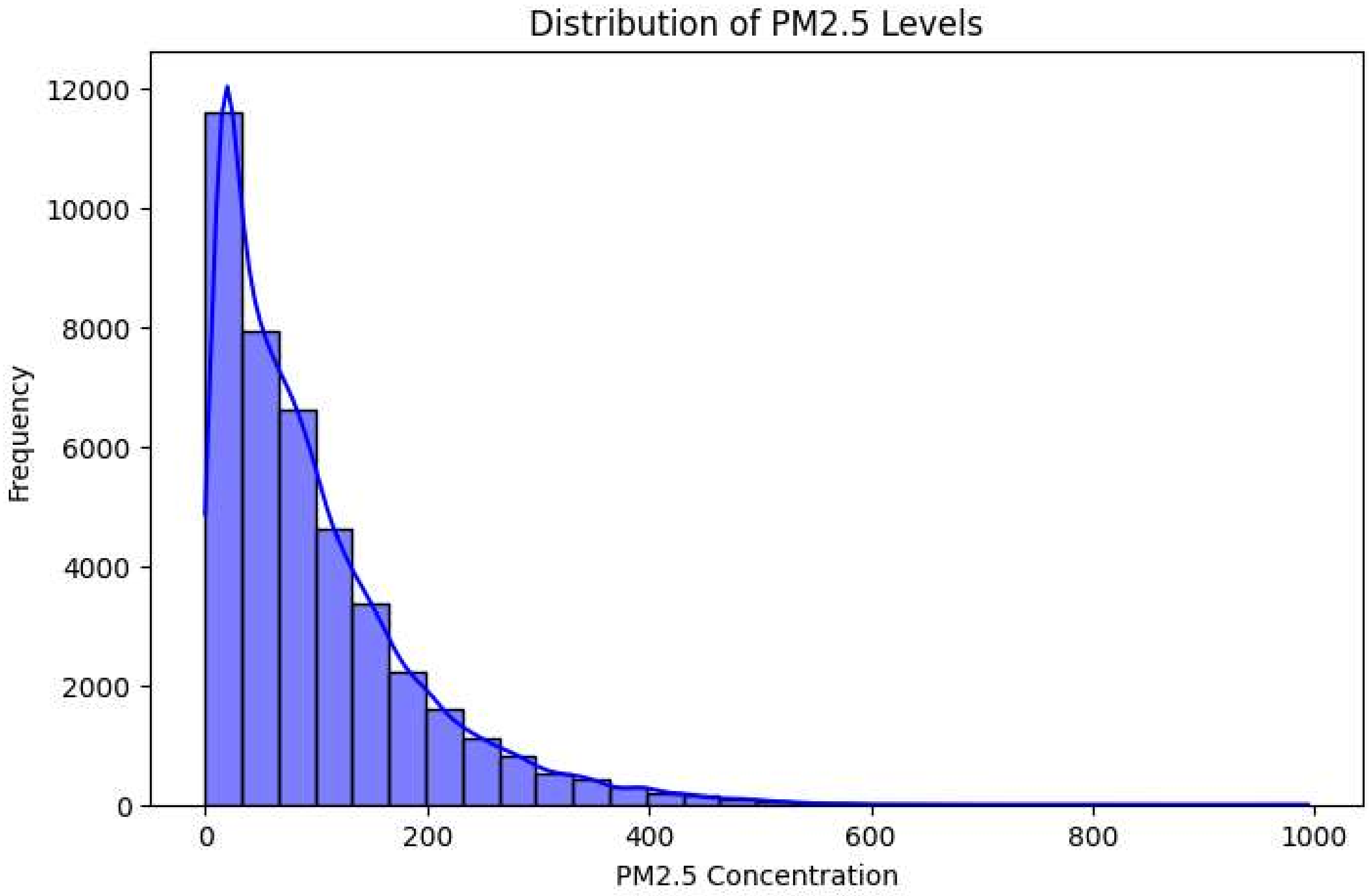

```
# Encoding categorical variable
X = pd.get_dummies(X, columns=['cbwd'], drop_first=True)
```

```
X
```

	month	hour	DEWP	TEMP	PRES	Iws	Is	Ir	cbwd_NW	cbwd_SE	cbwd_cv
24	1	0	-16	-4.0	1020.0	1.79	0	0	False	True	False
25	1	1	-15	-4.0	1020.0	2.68	0	0	False	True	False
26	1	2	-11	-5.0	1021.0	3.57	0	0	False	True	False
27	1	3	-7	-5.0	1022.0	5.36	1	0	False	True	False
28	1	4	-7	-5.0	1022.0	6.25	2	0	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...
43819	12	19	-23	-2.0	1034.0	231.97	0	0	True	False	False
43820	12	20	-22	-3.0	1034.0	237.78	0	0	True	False	False
43821	12	21	-22	-3.0	1034.0	242.70	0	0	True	False	False
43822	12	22	-22	-4.0	1034.0	246.72	0	0	True	False	False
43823	12	23	-21	-3.0	1034.0	249.85	0	0	True	False	False

41757 rows × 11 columns

```
plt.figure(figsize=(8,5))
sns.histplot(df["pm2.5"].dropna(), bins=30, kde=True, color="blue")
plt.xlabel("PM2.5 Concentration")
plt.ylabel("Frequency")
plt.title("Distribution of PM2.5 Levels")
plt.show()
```




```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)


# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_test_scaled
```

```
array([[ 0.14484032, -1.36923608,  0.98698145, ..., -0.69106751,
          1.36742289, -0.52199088],
        [-0.43422628, -1.22487237,  0.50191281, ..., -0.69106751,
         -0.73130266, -0.52199088],
        [ 0.72390693, -0.21432641,  0.29402625, ...,  1.44703662,
         -0.73130266, -0.52199088],
        ...,
        [ 1.01344023, -0.21432641,  0.22473073, ...,  1.44703662,
         -0.73130266, -0.52199088],
        [-1.30282619,  0.36312842, -0.67611102, ..., -0.69106751,
          1.36742289, -0.52199088],
        [ 1.59250684,  1.6624018 , -0.81470206, ..., -0.69106751,
         -0.73130266,  1.91574229]])
```

```
# Models dictionary
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "SVM": SVC(),
    "Random Forest": RandomForestClassifier(n_estimators=100),
    "KNN": KNeighborsClassifier()  # Default k value
}
```

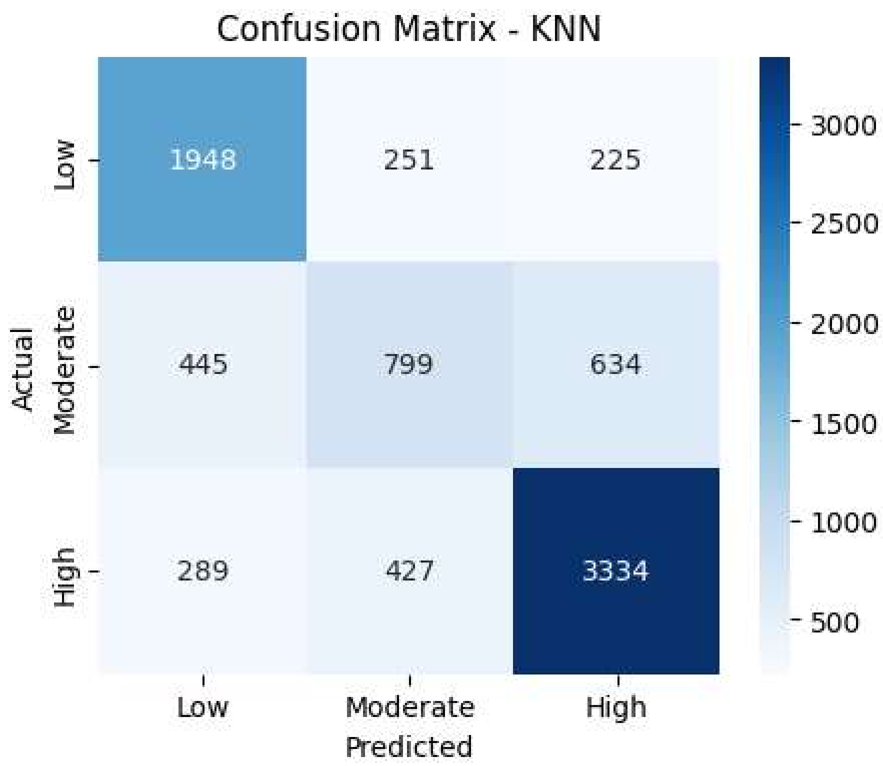
```
# Training models and evaluating metrics
print("Model Performance Before PCA:\n")
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')

    print(f"{name}: Accuracy={accuracy:.4f}, Precision={precision:.4f}, Recall={recall:.4f}")
    print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Model Performance Before PCA:
```

```
Logistic Regression: Accuracy=0.6577, Precision=0.6142, Recall=0.6577

Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.70      0.71      2424
           1       0.40      0.10      0.16      1878
           2       0.66      0.89      0.76      4050

    accuracy                           0.66      8352
   macro avg       0.59      0.56      0.54      8352
weighted avg       0.61      0.66      0.61      8352

SVM: Accuracy=0.6976, Precision=0.6673, Recall=0.6976

Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.76      0.76      2424
           1       0.51      0.12      0.20      1878
           2       0.69      0.93      0.79      4050

    accuracy                           0.70      8352
   macro avg       0.65      0.60      0.58      8352
weighted avg       0.67      0.70      0.65      8352

Random Forest: Accuracy=0.7551, Precision=0.7413, Recall=0.7551

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.81      0.80      2424
           1       0.60      0.39      0.48      1878
           2       0.78      0.89      0.83      4050

    accuracy                           0.76      8352
   macro avg       0.72      0.70      0.70      8352
weighted avg       0.74      0.76      0.74      8352

KNN: Accuracy=0.7281, Precision=0.7180, Recall=0.7281
```

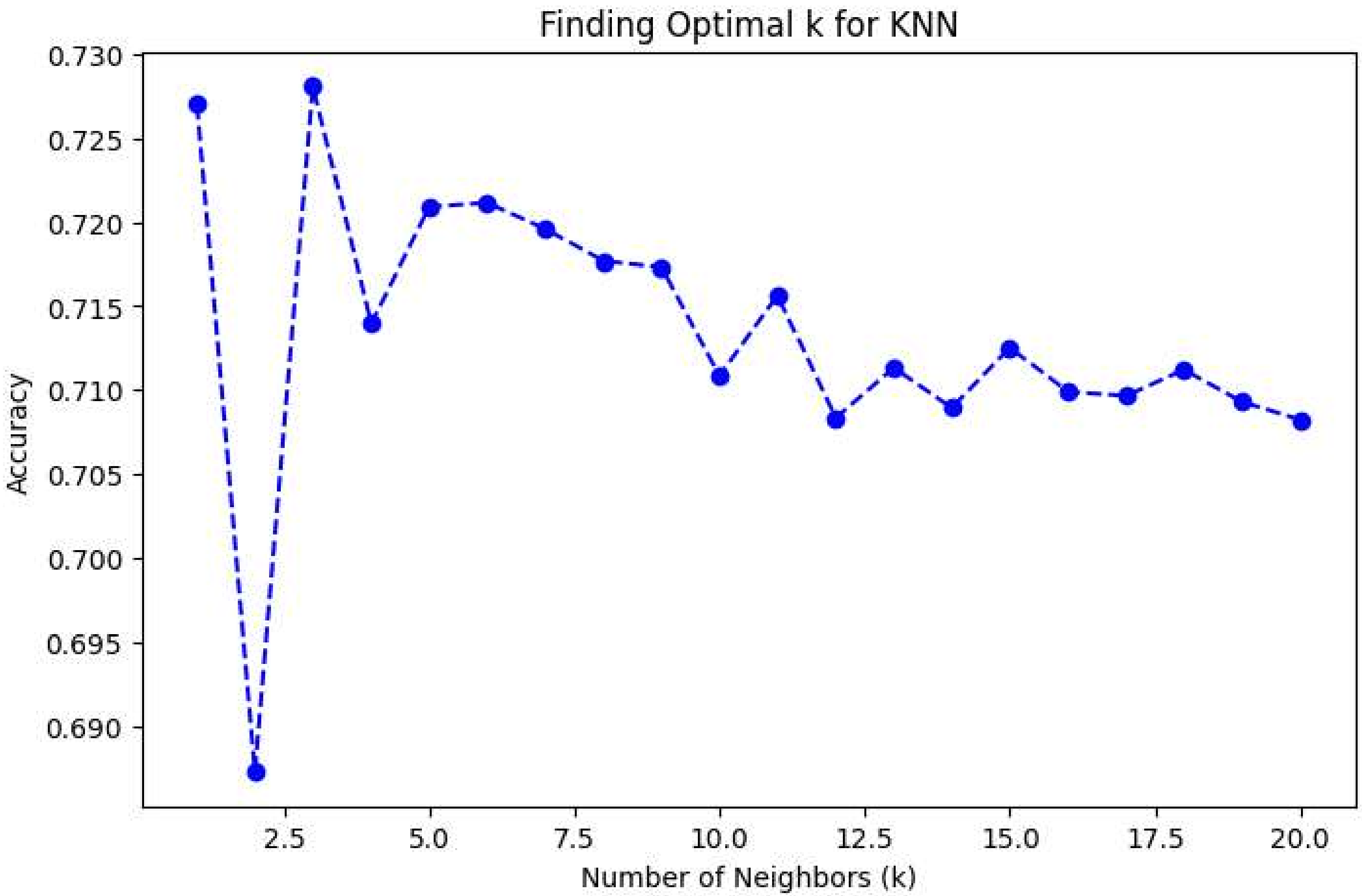
```
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.80      0.76      2424
           1       0.54      0.43      0.48      1878
           2       0.80      0.82      0.81      4050

    accuracy                           0.73      8352
   macro avg       0.69      0.68      0.68      8352
weighted avg       0.72      0.73      0.72      8352
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Low", "Moderate", "High"], yticklabels=["Low", "Moderate", "High"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix - {name}')
plt.show()
```




```
# Finding the best k-value for KNN using cross-validation
k_values = list(range(1, 21))
knn_scores = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    knn_scores.append(accuracy_score(y_test, y_pred))


# Plotting k-values vs accuracy
plt.figure(figsize=(8, 5))
plt.plot(k_values, knn_scores, marker='o', linestyle='dashed', color='b')
plt.xlabel("Number of Neighbors (k)")
```

```
plt.ylabel("Accuracy")
plt.title("Finding Optimal k for KNN")
plt.show()

best_k = k_values[np.argmax(knn_scores)]
print(f"Best k-value for KNN: {best_k}")
```




```
Best k-value for KNN: 3
```

```
# Retraining KNN with best k
models["KNN"] = KNeighborsClassifier(n_neighbors=best_k)
models["KNN"]
```

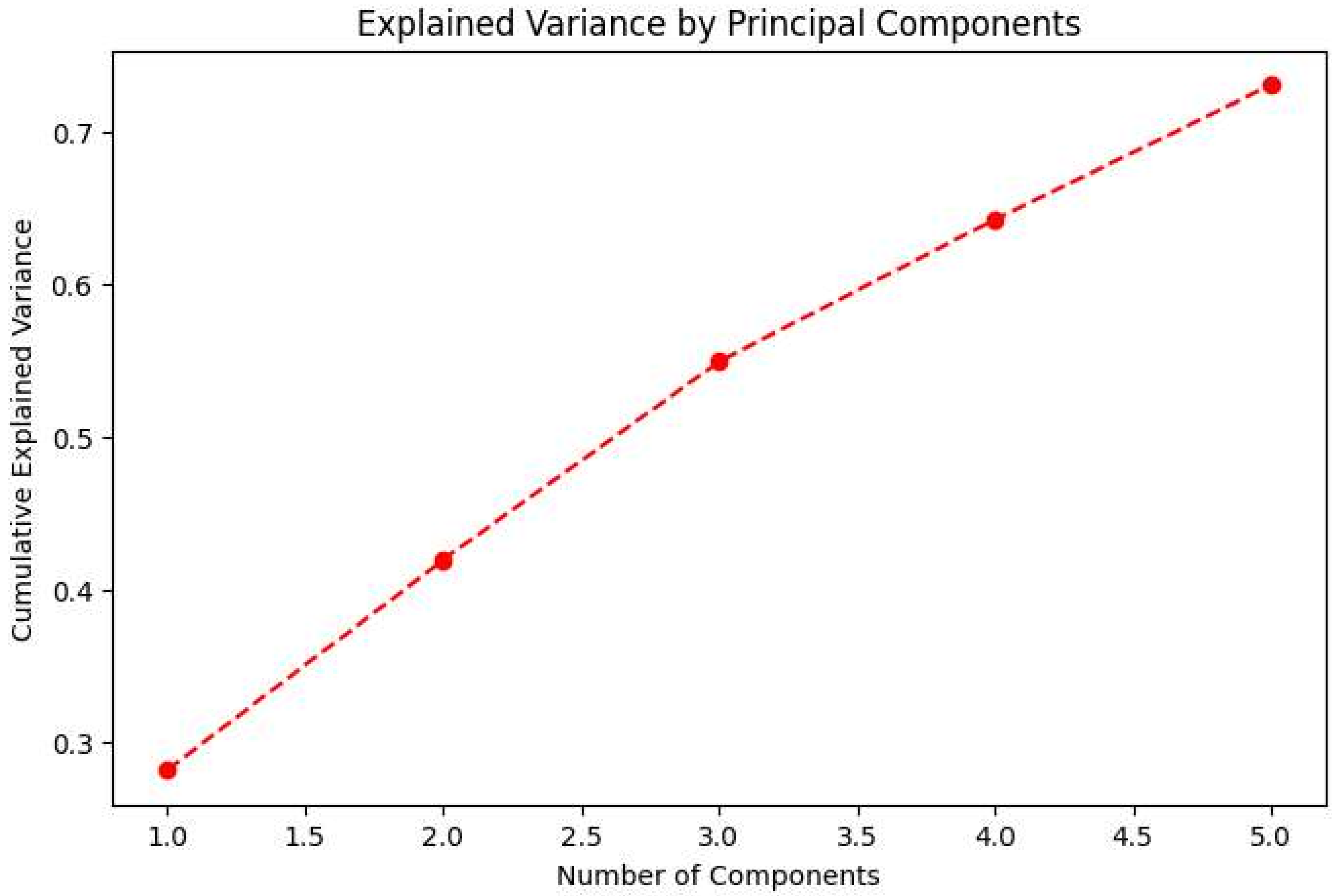
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
# PCA for dimensionality reduction
pca = PCA(n_components=5)  # Keeping 5 principal components
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
X_test_pca
```

```
array([[ 1.99479745, -0.77706637,  0.33248258,  0.70556583, -1.09132009],
       [ 0.86122984,  0.5691142 ,  0.31834262,  0.9746783 , -1.12703565],
       [ 0.17724983,  0.00720149,  1.64182335,  0.03028731, -0.3956051 ],
       ...,
       [-0.62365595,  0.00349314,  1.50716775, -0.22351607, -0.13556023],
       [-0.56218826, -1.04784366, -1.94541602,  0.19288826, -0.23468626],
       [-0.53052817,  1.70535313, -0.68288109, -1.94729849,  1.75619295]])
```

```
# Plot PCA explained variance
plt.figure(figsize=(8,5))
plt.plot(range(1, 6), pca.explained_variance_ratio_.cumsum(), marker='o', linestyle='--', color='r')
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Explained Variance by Principal Components")
plt.show()

print("\nModel Performance After PCA:\n")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred_pca = model.predict(X_test_pca)

    accuracy_pca = accuracy_score(y_test, y_pred_pca)
    precision_pca = precision_score(y_test, y_pred_pca, average='weighted')
    recall_pca = recall_score(y_test, y_pred_pca, average='weighted')
```

```
print(f"{name}: Accuracy={accuracy_pca:.4f}, Precision={precision_pca:.4f}, Recall={recall_pca:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred_pca))
```




```
Model Performance After PCA:

Logistic Regression: Accuracy=0.5803, Precision=0.5191, Recall=0.5803

Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.57      0.58      2424
           1       0.30      0.00      0.00      1878
           2       0.58      0.86      0.69      4050

    accuracy                           0.58      8352
   macro avg       0.49      0.47      0.42      8352
```

```
weighted avg       0.52      0.58      0.50      8352

SVM: Accuracy=0.6318, Precision=0.6065, Recall=0.6318

Classification Report:
               precision    recall  f1-score   support

           0       0.73      0.60      0.66      2424
           1       0.46      0.01      0.02      1878
           2       0.60      0.94      0.73      4050

    accuracy                           0.63      8352
   macro avg       0.60      0.52      0.47      8352
weighted avg       0.61      0.63      0.55      8352

Random Forest: Accuracy=0.6584, Precision=0.6345, Recall=0.6584

Classification Report:
               precision    recall  f1-score   support

           0       0.72      0.69      0.71      2424
           1       0.41      0.25      0.31      1878
           2       0.69      0.83      0.75      4050

    accuracy                           0.66      8352
   macro avg       0.61      0.59      0.59      8352
weighted avg       0.63      0.66      0.64      8352

KNN: Accuracy=0.6136, Precision=0.5979, Recall=0.6136

Classification Report:
               precision    recall  f1-score   support

           0       0.60      0.70      0.65      2424
           1       0.39      0.27      0.32      1878
           2       0.69      0.72      0.71      4050

    accuracy                           0.61      8352
   macro avg       0.56      0.56      0.56      8352
weighted avg       0.60      0.61      0.60      8352
```

```
# Confusion Matrix after PCA
cm_pca = confusion_matrix(y_test, y_pred_pca)
plt.figure(figsize=(5,4))
sns.heatmap(cm_pca, annot=True, fmt='d', cmap='Greens', xticklabels=["Low", "Moderate", "High"], yticklabels=["Low", "Moderate", "Hi
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix - {name} (After PCA)')
plt.show()
```