# American International University- Bangladesh

CSC 4162: Programming in Python
Final Project Report
Fall 23-24

Supervised By:
Dr. Abdus Salam
Assistant Professor, Computer Science-AIUB

Project Title: Air Quality Prediction
Section: B
Group: 2

Submitted By:

| Student Name | Student Id |
|---|---|
| MD. SHAFIUR RAHMAN | 20-44191-2 |
| MD. ABDULLA AL MAMUN | 20-44192-2 |

## PROJECT: AIR QUALITY PREDICTION

### DATASET DESCRIPTION:

The dataset, titled "AQI and Lat Long of Countries," comprises 16,695 entries spanning various cities across different countries. It contains detailed information on air quality, specifically the Air Quality Index (AQI) and related pollutants, along with geographical coordinates. Key columns in the dataset include:

- **Country**: The country where the city is located.
- **City**: Name of the city.
- **AQI Value**: Numeric value representing the Air Quality Index.
- **AQI Category**: Categorical representation of the AQI (e.g., Good, Moderate).
- **CO AQI Value**: Carbon Monoxide AQI value.
- **CO AQI Category**: Categorical representation of the CO AQI.
- **Ozone AQI Value**: Ozone AQI value.
- **Ozone AQI Category**: Categorical representation of the Ozone AQI.
- **NO2 AQI Value**: Nitrogen Dioxide AQI value.
- **NO2 AQI Category**: Categorical representation of the NO2 AQI.
- **PM2.5 AQI Value**: Particulate Matter 2.5 AQI value.
- **PM2.5 AQI Category**: Categorical representation of the PM2.5 AQI.
- **lat**: Latitude of the city.
- **lng**: Longitude of the city.

## FEATURES:

**Task 1:** Read/Load the dataset file in your program. Use Pandas library to complete this task.

```
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/AQI and Lat Long of Countries.csv')
df.head()
```

The code uses Pandas library to read a CSV file containing AQI (Air Quality Index) and latitude-longitude data of countries. It utilizes pd.read_csv() to load the dataset into a DataFrame named df and then displays the first few rows using df.head().

**Task 2:** Apply appropriate data cleaning techniques to the dataset. In this step, replace bad data using proper methods and do not delete any record except duplicate records. Use Pandas library to complete this task.

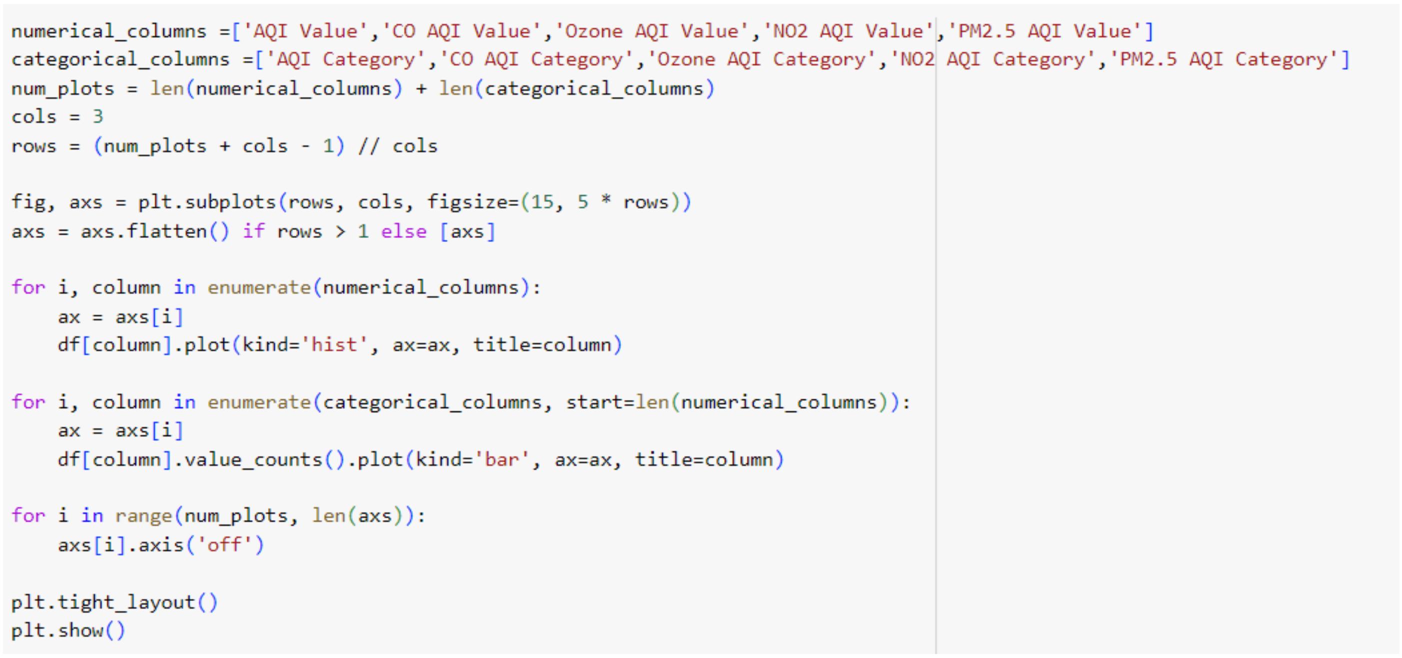
```
df.info()
df.isna().sum()
```

```
x = df["Country"].mode()[0]
df["Country"].fillna(x, inplace = True)
df.info()
df.isna().sum()
```

```
df.duplicated().sum()
```

This task involves handling missing values in the dataset without removing any records except for duplicates. The code assesses missing values using df.info() and df.isna().sum(). It replaces missing values in the "Country" column with the mode (most frequent value) of that column using fillna(). Duplicate records are identified and removed using df.duplicated().sum().

**Task 3:** Draw graphs to analyze the frequency distributions of the features. Use Matplotlib library to complete this task.



```python
numerical_columns =['AQI Value','CO AQI Value','Ozone AQI Value','NO2 AQI Value','PM2.5 AQI Value']
categorical_columns =['AQI Category','CO AQI Category','Ozone AQI Category','NO2 AQI Category','PM2.5 AQI Category']
num_plots = len(numerical_columns) + len(categorical_columns)
cols = 3
rows = (num_plots + cols - 1) // cols

fig, axs = plt.subplots(rows, cols, figsize=(15, 5 * rows))
axs = axs.flatten() if rows > 1 else [axs]

for i, column in enumerate(numerical_columns):
    ax = axs[i]
    df[column].plot(kind='hist', ax=ax, title=column)

for i, column in enumerate(categorical_columns, start=len(numerical_columns)):
    ax = axs[i]
    df[column].value_counts().plot(kind='bar', ax=ax, title=column)

for i in range(num_plots, len(axs)):
    axs[i].axis('off')

plt.tight_layout()
plt.show()
```

For this task, the code generates histograms for numerical columns and bar plots for categorical columns using Matplotlib. It iterates through numerical and categorical columns, plotting their respective distributions using plot(kind='hist') for numerical data and value_counts().plot(kind='bar') for categorical data.

**Task 4:** Draw graphs to illustrate if there is any relationship between target column to any other columns of the dataset. Use Matplotlib library to complete this task.

```
LE = LabelEncoder()
df['Country'] = LE.fit_transform(df['Country'])
df['City'] = LE.fit_transform(df['City'])
df['AQI Category'] = LE.fit_transform(df['AQI Category'])
df['NO2 AQI Category'] = LE.fit_transform(df['NO2 AQI Category'])
df['CO AQI Category'] = LE.fit_transform(df['CO AQI Category'])
df['Ozone AQI Category'] = LE.fit_transform(df['Ozone AQI Category'])
df['PM2.5 AQI Category'] = LE.fit_transform(df['PM2.5 AQI Category'])
df.head()
```

```
df.info()
```

```
df = df.drop(['lat','lng'],axis=1)
corr_matrix = df.corr()
plt.figure(figsize=(10, 8))
heatmap = plt.imshow(corr_matrix, cmap='rainbow', interpolation='nearest')
for i in range(len(corr_matrix)):
    for j in range(len(corr_matrix)):
        plt.text(j, i, f'{corr_matrix.iloc[i, j]:.2f}', ha='center', va='center', color='black')

plt.colorbar(heatmap)
plt.title('Correlation Matrix')
plt.xticks(np.arange(len(corr_matrix)), corr_matrix.columns, rotation=90)
plt.yticks(np.arange(len(corr_matrix)), corr_matrix.index)
plt.tight_layout()
plt.show()
```

Here, the code encodes categorical columns using LabelEncoder and creates a correlation matrix for the dataset. It uses Matplotlib to generate a heatmap visualization of the correlation matrix, displaying the correlation coefficients between different columns.

**Task 5:** Perform scaling to the features of the dataset. Remember that you will need to apply data conversion before performing scaling if it is needed.

```
LE = LabelEncoder()
for col in categorical_columns:
    df[col] = LE.fit_transform(df[col])
scaler = StandardScaler()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
df.head()
```

Feature scaling is performed in this task. Categorical columns are encoded using LabelEncoder, and numerical columns are scaled using StandardScaler from the scikit-learn library. This step ensures that all features are on a similar scale.

**Task 6:** Split your data into two parts: Training dataset and Testing dataset. You must use the function train_test_split() to complete this task and use value 123 as the value of the random_state parameter of this function.

```
X = df.drop(['AQI Category'], axis=1)
y = df['AQI Category']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 123)
X_train.shape, X_test.shape, y_train, y_test
```

The code splits the dataset into training and testing sets using train_test_split() from scikit-learn. It separates the features (X) and the target variable (y) and allocates 80% of the data for training and 20% for testing, setting a random state for reproducibility.

**Task 7:** Apply Naïve Bayes Classifier to the dataset. Build (train) your prediction model in this step.

```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
y_pred_train = gnb.predict(X_train)
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))

print('Training set score: {:.4f}'.format(gnb.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(gnb.score(X_test, y_test)))
```

This task involves training a Naïve Bayes classifier on the training dataset. The code uses GaussianNB() from scikit-learn to instantiate the classifier, fits it to the training data using fit(), and predicts the target variable for the test set. Model accuracy scores for both training and test sets are calculated using accuracy_score()

**Taks 8:** Calculate the confusion matrix for your model. Interpret it in detail in the report.

```
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix)
cm_display.plot()
plt.show()
```

The code computes the confusion matrix using confusion_matrix() from scikit-learn's metrics module. It then displays the confusion matrix using ConfusionMatrixDisplay from the same library to visualize the model's performance.

**Task 9:** Calculate the accuracy, precision, recall and f-1 score of your model.

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')
print(f"Accuracy: {accuracy}\nPrecision: {precision}\nRecall: {recall}\nF1 Score: {f1}")
```

This task calculates various evaluation metrics - accuracy, precision, recall, and F1-score for the model. These metrics are computed using functions like accuracy_score(), precision_score(), recall_score(), and f1_score() from scikit-learn's metrics module.

**Task 10:** Show how 10-fold cross validation can be used to build a naïve bayes classifier and report the accuracy of this model.

```
k_folds = KFold(n_splits = 10)
scores = cross_val_score(gnb, X, y, cv = k_folds)
print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

Here, the code demonstrates the use of KFold cross-validation with 10 splits to assess the Naïve Bayes classifier's performance. It uses cross_val_score() from scikit-learn to obtain cross-validation scores and computes the average score.