Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Data Visualization and Analysis

Overview:

The main goal of this project is to begin thinking about and analyzing data and manipulating it via a program.

Background:

To complete the Data Visualization and Analysis project, we were given a dataset named "project1-hits.txt". This datafile consists simply of a list of #hits per hour for an entire month (31 days). It comes as a comma-separated list of values: each line contains the hour of the month and the number of visits that occurred during that hour (e.g., 1,2272). There are 24×31=744 total lines. Afterwards, we analyze this dataset, visualize it, and try to produce a trend line in order to predict future performance. To do these, we use several techniques such as data pre-processing, data visualization, linear regression etc. We use Python to execute this programming project using PyCharm. All computations were performed by our program, not by a built-in library routine. The code has been attached at the end of this report.

Data Pre-processing: [read in and clean the data]

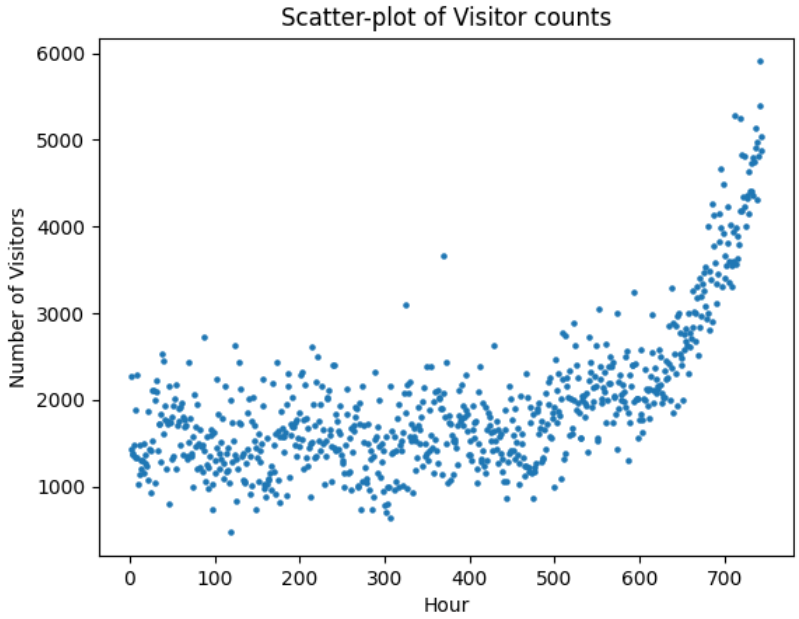
After reading the dataset, we observe that there are some types of error has prevented the data from being measured and/or recorded at certain times, represented as a 'nan' ("not a number") value in the datafile. We replace those 'nan' ("not a number) values by the average number of hits of that particular day. For instance, if there is 'nan' value in day 1, we calculate the average number of hits of the day 1 and replace that average value with the 'nan' value. A part of our code has been attached below which was used for data pre-processing.

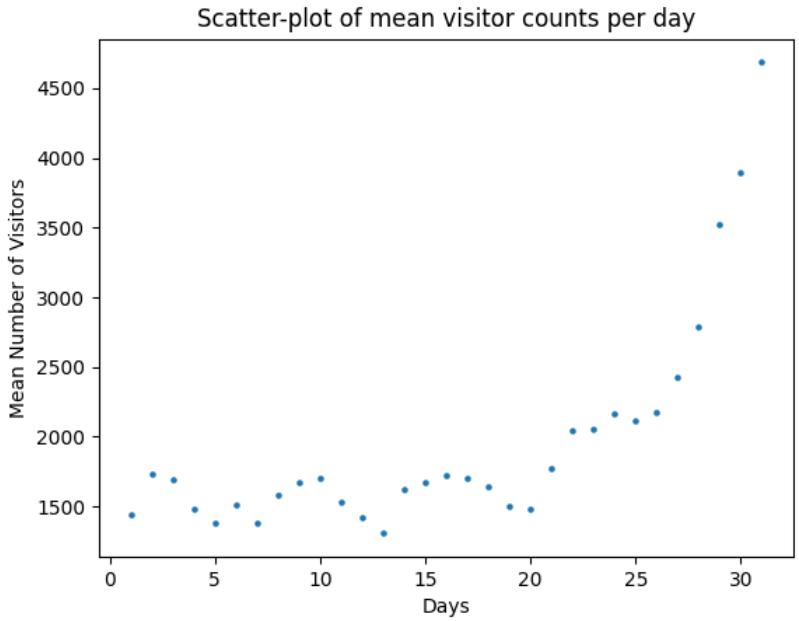
```
def load_clean_data():
    '''
    Loading data from file and filling in the missing values
    :return: data[]
    '''

    # Read data from file into 2D array
    data = []
    with open("project1-hits.txt", mode="r") as fin:
        for line in fin:
            parts = line.strip().split(",")
            data.append(parts)

    # Convert to int and fill in missing data
    sum = 0
    count = 0
    nan_found = False
    for i in range(0, len(data), 1):
        hour = int(data[i][0])
        visits = data[i][1]
        if visits == 'nan':
            nan_found = True
            visits = -1
        else:
            visits = int(visits)
            sum += visits
            count += 1
        data[i] = [hour, visits]

        # For every 24 hours i.e. day, replace missing data (-1) with
        # mean visitor count for that day.
        if nan_found and data[i][0] % 24 == 0:
            mean = sum * 1.0 / count
            for j in range((i - 24 + 1), i + 1, 1):
                if data[j][1] == -1:
                    data[j][1] = mean
            sum = 0
            count = 0
            nan_found = False

    return data
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Visualization: [display the data]

We use a Python visualization library named "matplotlib" to visualize the data by creating the scatterplot (a Cartesian display of two-variable data) within our program. Two plots have been attached below. From the first plot, we can see that, it follows an increasing trend though the number of hits increases exponentially within the last 10 days of this month. At the same time, the standard deviation has narrowed down towards the end of this month. And from the second plot, we observe that more people are reading it the longer it's been active.





Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Analysis and Discussion: [perform simple linear regression on the data]:

We decide to begin with the simplest analysis called linear regression. The linear regression is a method for fitting a curve, in this case a straight line, to a set of points. The *slope* of the line represents the correlation between the *x (Hours)* and *y (Number of visitors)* values; the *intercept* gives the center of mass of the data points. There are several different ways of performing a linear regression, typically based on the *least-squares* method that attempts to minimize the sum of squared residuals (i.e., the error). A simple method follows.

Obtain/calculate:

- Σ X: the sum of all X values
- Σ Y: the sum of all Y values
- Σ XY: the sum of the products of each X, Y pair
- Σ X^2: the sum of the squares of every X value
- Σ Y^2: the sum of the squares of every Y value

Suppose N is the number of data points. Then the relevant calculations are:

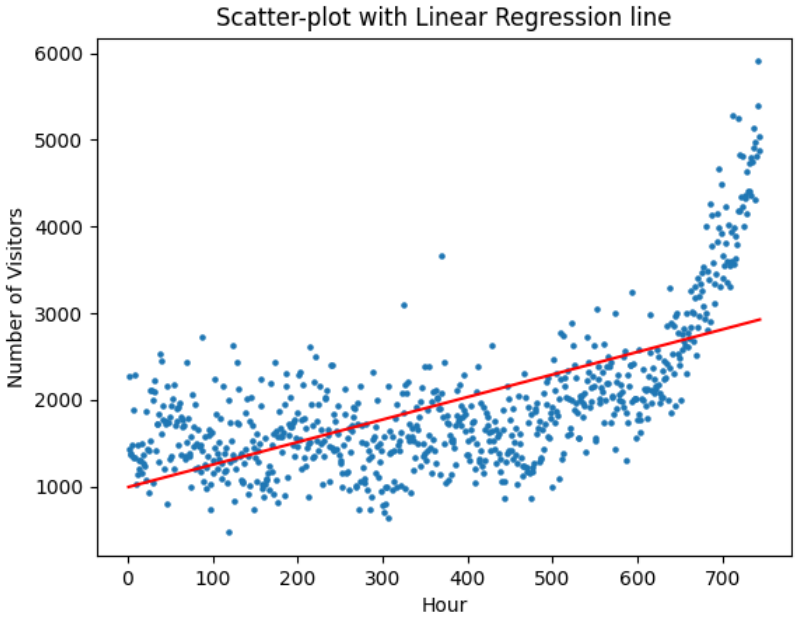
$$\text{slope} = \frac{(N\sum XY) - (\sum X \sum Y)}{(N\sum X^2) - (\sum X)^2}$$

$$\text{intercept} = \frac{\sum Y - (\text{slope} \sum X)}{N}$$

With these values we create the regression equation:

$$Y = \text{intercept} + \text{slope} * X$$

The visualization of the regression analysis of our data has been attached below:



The regression equation that accurately captures current and expected visitor behavior is -

$$Y = 993.72 + 2.60 * X$$

and we use it to make predictions about the future.

Prediction:

- How many visits would you expect at Noon on the fifth day of the next month?

$$Y= 993.72 + 2.60 \times \{(4x24)+12+744\} = 3.208.92 \text{ visits}$$

Future Work:

We can see from our data visualization that an exponential increasing trend exists in our dataset. As a result, if we use simple linear regression for future prediction, it will not predict the number of visits accurately. So, to get the expected prediction, we can use polynomial regression instead of simple linear regression model.

On the other hand, we analyze the number of visits based on the hour only whereas there can be few more facts other than the hours to analyze the accurate number of visits.

[Please turn the next page for the source-code]

```python
import matplotlib.pyplot as plt


def load_clean_data():
    '''
    Loading data from file and filling in the missing values
    :return: data[]
    '''

    # Read data from file into 2D array
    data = []
    with open("project1-hits.txt", mode="r") as fin:
        for line in fin:
            parts = line.strip().split(",")
            data.append(parts)

    # Convert to int and fill in missing data
    sum = 0
    count = 0
    nan_found = False
    for i in range(0, len(data), 1):
        hour = int(data[i][0])
        visits = data[i][1]
        if visits == 'nan':
            nan_found = True
            visits = -1
        else:
            visits = int(visits)
            sum += visits
            count += 1
        data[i] = [hour, visits]

        # For every 24 hours i.e. day, replace missing data (-1) with
        # mean visitor count for that day.
        if nan_found and data[i][0] % 24 == 0:
            mean = sum * 1.0 / count
            for j in range((i - 24 + 1), i + 1, 1):
                if data[j][1] == -1:
                    data[j][1] = mean
            sum = 0
            count = 0
            nan_found = False

    return data


def scatterplot_data(data):
    '''
    Scatter Plot using the data
    :param data: visitor count data
    '''

    x_points = [row[0] for row in data]
```

```
    y_points = [row[1] for row in data]
    plt.scatter(x_points, y_points, 5)
    plt.title("Scatter-plot of Visitor counts")
    plt.xlabel("Hour")
    plt.ylabel("Number of Visitors")
    plt.show()

    x_points = [i for i in range(1, 32, 1)]
    y_points = []
    i = 0
    while i < len(data):
        sum = 0
        count = 0
        for j in range(i, i+24, 1):
            sum += data[j][1]
            count += 1
        y_points.append(sum*1.0/count)
        i += 24

    plt.scatter(x_points, y_points, 5)
    plt.title("Scatter-plot of mean visitor counts per day")
    plt.xlabel("Days")
    plt.ylabel("Mean Number of Visitors")
    plt.show()


def linear_regression(data):
    '''
    Linear Regression over Scatter plot
    using sum squared error equations for linear regression.
    :param data: visitor count data
    '''

    sum_X = sum_Y = sum_XY = sum_X2 = sum_Y2 = 0.0
    N = len(data)
    for row in data:
        sum_X += row[0]
        sum_Y += row[1]
        sum_XY += (row[0] * row[1])
        sum_X2 += (row[0] ** 2)
        sum_Y2 += (row[1] ** 2)

    slope = (N * sum_XY - sum_X * sum_Y) / (N * sum_X2 - (sum_X ** 2
))
    intercept = (sum_Y - slope * sum_X) / N

    print(f"Equation: y = {intercept:.2f} + {slope:.2f} * X")

    # Predict values for linear regression line.
    lr_x_points = [data[0][0], data[1][0], data[2][0], data[-3][0],
data[-2][0], data[-1][0]]
    lr_y_points = [
        (intercept + slope * data[0][0]),
```

```
105         (intercept + slope * data[1][0]),
106         (intercept + slope * data[2][0]),
107         (intercept + slope * data[-3][0]),
108         (intercept + slope * data[-2][0]),
109         (intercept + slope * data[-1][0])
110     ]
111
112     x_points = [row[0] for row in data]
113     y_points = [row[1] for row in data]
114
115     plt.plot(lr_x_points, lr_y_points, c="r")
116     plt.scatter(x_points, y_points, 5)
117
118     plt.title("Scatter-plot with Linear Regression line")
119     plt.xlabel("Hour")
120     plt.ylabel("Number of Visitors")
121
122     plt.show()
123
124
125 def print_data(data):
126     for i in range(0, len(data), 1):
127         print(f"{data[i][0]}, {data[i][1]}")
128
129
130 def start():
131     data = load_clean_data()
132     scatterplot_data(data)
133     linear_regression(data)
134     # print_data(data)
135
136
137 if __name__ == '__main__':
138     start()
139
```