AIR QUALITY IN NAIROBI AROUND DANDORA DUMPSITE

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
```

In this project, you will be prepare, clean and plot time series data. The goal of this project is to:

- Prepare Nairobi **time series data**.
- Plot time series data for **PM10 and PM2.5 readings**.

PREPARE DATA

IMPORT

You will prepare the data by making sure its in the right place and format for analysis. Th first step of any data preparation is importing the data and cleaning it.

The data for this project is from one of Africa's largest open data platforms: openAfrica and it comes in a CSV file: **e-waste-story-walk_about.csv.

Read in the file into a dataframe named df.

In [2]:
```
df = pd.read_csv('/Users/Cheryl/Downloads/e-waste-story-walk_about.csv')
```

Clean df

Now that you have your dataframe, its time to inspect them to see if they need any cleaning.

You will inspect **df** by looking at its **shape** attribute. Then use the **info** method to see the data types and number of missing values for each column. Finally, use the **head** method to determine to look at the first five rows of your dataset.

In [3]:
```
df.shape
```

Out[3]:
```
(64, 6)
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64 entries, 0 to 63
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    64 non-null     object 
 1   Time    64 non-null     object 
 2   PM 10   64 non-null     float64
 3   PM 2.5  64 non-null     float64
 4   Lat     0 non-null      float64
 5   Lon     0 non-null      float64
dtypes: float64(4), object(2)
memory usage: 3.1+ KB
```

In [5]:
```
df.head()
```

Out[5]:

	Date	Time	PM 10	PM 2.5	Lat	Lon
0	2019-03-13	10:50:18	28.02	7.47	NaN	NaN
1	2019-03-13	10:50:55	37.00	9.00	NaN	NaN
2	2019-03-13	10:51:32	43.53	8.88	NaN	NaN
3	2019-03-13	10:52:09	27.92	8.57	NaN	NaN
4	2019-03-13	10:52:46	40.95	9.75	NaN	NaN

It looks like there are a couple of problems in this DataFrame that you need to solve. First, there are many rows with NaN values in the **"lat"** and **"lon"** columns. Second, since the date is for a single day in **13/03/2019**, create a new column **DateTime** and drop the **Date** and **Time** columns.

Clean **df** by dropping rows with NaN values. Then remove the "$" and "," characters from "price_usd" and recast the values in the column as floats.

In [6]:
```
# Remove NaN Values
df.drop(columns=["Lat", "Lon"], inplace = True)
```

In [7]:
```
#Create "DateTime" by combining "Date" and "Time columns"
df["DateTime"] = (df["Date"] + df["Time"])
df.drop(columns=["Date", "Time"], inplace = True)
```

In [8]:
```
df.head()
```

Out[8]:

	PM 10	PM 2.5	DateTime
0	28.02	7.47	2019-03-1310:50:18
1	37.00	9.00	2019-03-1310:50:55
2	43.53	8.88	2019-03-1310:51:32
3	27.92	8.57	2019-03-1310:52:09
4	40.95	9.75	2019-03-1310:52:46

You can now set the index to "DateTime".

In [9]:
```
df = pd.DataFrame(df).set_index("DateTime")
print(df.info())
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 64 entries, 2019-03-1310:50:18 to 2019-03-1311:45:40
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   PM 10   64 non-null     float64
 1   PM 2.5  64 non-null     float64
dtypes: float64(2)
memory usage: 1.5+ KB
None
```

Out[9]:

DateTime	PM 10	PM 2.5
2019-03-1310:50:18	28.02	7.47
2019-03-1310:50:55	37.00	9.00
2019-03-1310:51:32	43.53	8.88
2019-03-1310:52:09	27.92	8.57
2019-03-1310:52:46	40.95	9.75

Save df

The data is clean and now you need to save it as a CSV file so that you can examine it in your Explanatory Data Analysis (EDA).

In [10]:
```
df.to_csv('/Users/Cheryl/Downloads/e-waste-story-walk_about-clean.csv', index = False)
```

EXPLORE

The next step is Explanatory Data Analysis (EDA) where you get a feel for your data by summarizing its main characteristics using descriptive statistics and data visualizaton. You can start by looking at each column and asking yourself questions about what it says abut your dataset.

In [11]:
```
df.shape
```

Out[11]:
```
(64, 2)
```

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 64 entries, 2019-03-1310:50:18 to 2019-03-1311:45:40
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   PM 10   64 non-null     float64
 1   PM 2.5  64 non-null     float64
dtypes: float64(2)
memory usage: 1.5+ KB
```

In [13]:
```
df.head()
```

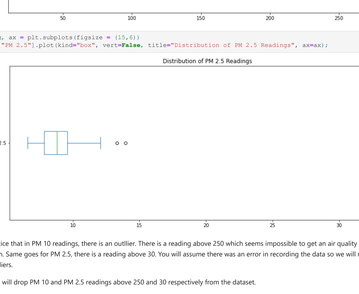
Out[13]:

DateTime	PM 10	PM 2.5
2019-03-1310:50:18	28.02	7.47
2019-03-1310:50:55	37.00	9.00
2019-03-1310:51:32	43.53	8.88
2019-03-1310:52:09	27.92	8.57
2019-03-1310:52:46	40.95	9.75

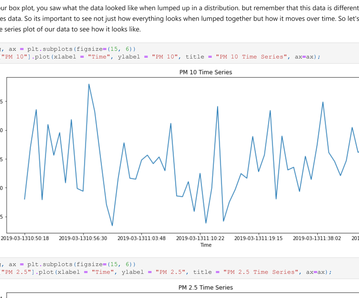
While there is only one Dtype on our DataFrame (float64), there is categorical data: **PM10** and **PM2.5** readings. They require a different exploration in our analysis. **PM10** refers to particles with an aerodynamic diameter smaller than 10 µm, and **PM2.5** refers to particles with an aerodynamic diameter smaller than 2.5 µm. You may also hear **PM10** called coarse dust and **PM2.5** called fine dust.

We will create a boxplot of the **PM10** and **PM2.5** readings in **df** to visual any outliers.

In [14]:
```
fig, ax = plt.subplots(figsize = (15,6))
ax = df["PM 10"].plot(kind="box", vert=False, title="Distribution of PM 10 Readings", ax=ax);
```



In [15]:
```
fig, ax = plt.subplots(figsize = (15,6))
df["PM 2.5"].plot(kind="box", vert=False, title="Distribution of PM 2.5 Readings", ax=ax);
```



Notice that in PM 10 readings, there is an outlier. There is a reading above 250 which seems impossible to get an air quality reading that high. Same goes for PM 2.5, there is a reading above 30. You will assume there was an error in recording the data so we will remove the outliers.

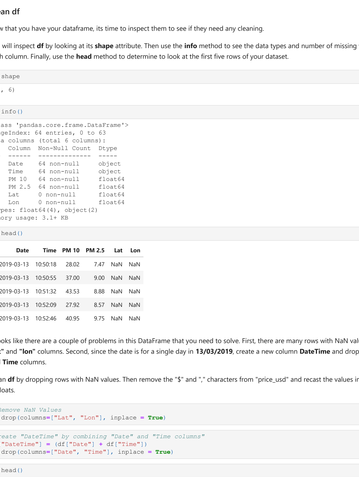
You will drop PM 10 and PM 2.5 readings above 250 and 30 respectively from the dataset.

In [16]:
```
# Remove Outliers
df = df[df["PM 10"] < 250]
df = df[df["PM 2.5"] < 30]
df.shape
```

Out[16]:
```
(63, 2)
```

In our box plot, you saw what the data looked like when lumped up in a distribution. but remember that this data is different, it is time series data. So its important to see not just how everything looks when lumped together but how it moves over time. So let's create a time series plot of our data to see how it looks like.

In [17]:
```
fig, ax = plt.subplots(figsize=(15, 6))
df["PM 10"].plot(xlabel = "Time", ylabel = "PM 10", title = "PM 10 Time Series", ax=ax);
```



In [18]:
```
fig, ax = plt.subplots(figsize=(15, 6))
df["PM 2.5"].plot(xlabel = "Time", ylabel = "PM 2.5", title = "PM 2.5 Time Series", ax=ax);
```

From our plot, the time series data is kinda all over the place. As you move from left to right, you are moving progressively into the future.

Since our data is limited to a short period, you will not resample df to provide the mean PM10 and PM2.5 readings for each hour or create a lag feature that contains the mean PM10 and PM2.5 readings for the previous hour. Hence create a correlation matrix to see if there is a relationship between what happened in the previous hour to what's happening now. This is also known as an autocorrelation.

In [ ]:
```
```