3.5. Air Quality in Dar es Salaam TZ

In [ ]:
```
!python -m playwright install chromium
```

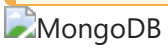
In [1]:
```
import warnings

warnings.simplefilter(action="ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

In [53]:
```
# Import libraries here
import inspect
import time
import warnings

import matplotlib.pyplot as plt
import pandas as pd
import plotly.express as px
import seaborn as sns
from IPython.display import VimeoVideo
from pymongo import MongoClient
from sklearn.metrics import mean_absolute_error
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.ar_model import AutoReg

warnings.filterwarnings("ignore")
```

Prepare Data

Connect

> 🛠️ **Instruction:** Locate the IP address of the machine running MongoDB and assign it to the variable `host`. Make sure to use a **string** (i.e., wrap the IP in quotes).
>
> ⚠️ **Note:** The IP address is **dynamic** — it may change every time you start the lab. Always check the current IP before proceeding.



In [4]:
```
host = "192.73.189.2"
```

Task 3.5.1

In [5]:
```
client = MongoClient(host=host, port=27017)
db = client["air-quality"]
dar = db["dar-es-salaam"]
```

Explore

Task 3.5.2

In [7]:
```
sites = dar.distinct("metadata.site")
sites
```

Out[7]:
```
[23, 11]
```

Task 3.5.3

In [11]:
```
result = dar.aggregate(
    [
        {"$group": {"_id": "$metadata.site", "count":{"$count": {}}}}
    ]
)
readings_per_site = list(result)
readings_per_site
```

Out[11]:
```
[{'_id': 23, 'count': 60020}, {'_id': 11, 'count': 173242}]
```

Import

Task 3.5.4

In [21]:
```
def wrangle(collection):
    results = collection.find(
        {"metadata.site": 11, "metadata.measurement": "P2"},
        projection={"P2": 1, "timestamp": 1, "_id": 0},
    )

    y = pd.DataFrame(results).set_index("timestamp")

    # Localize timezone
    y.index = y.index.tz_localize("UTC").tz_convert("Africa/Dar_es_Salaam")

    # Remove outliers
    y = y[y["P2"] < 100]

    # Resample to 1H window, ffill
    y = y["P2"].resample("1H").mean().fillna(method="ffill")

    # Drop null values
    y.dropna(inplace=True)

    return y
```

Use your `wrangle` function to query the `dar` collection and return your cleaned results.

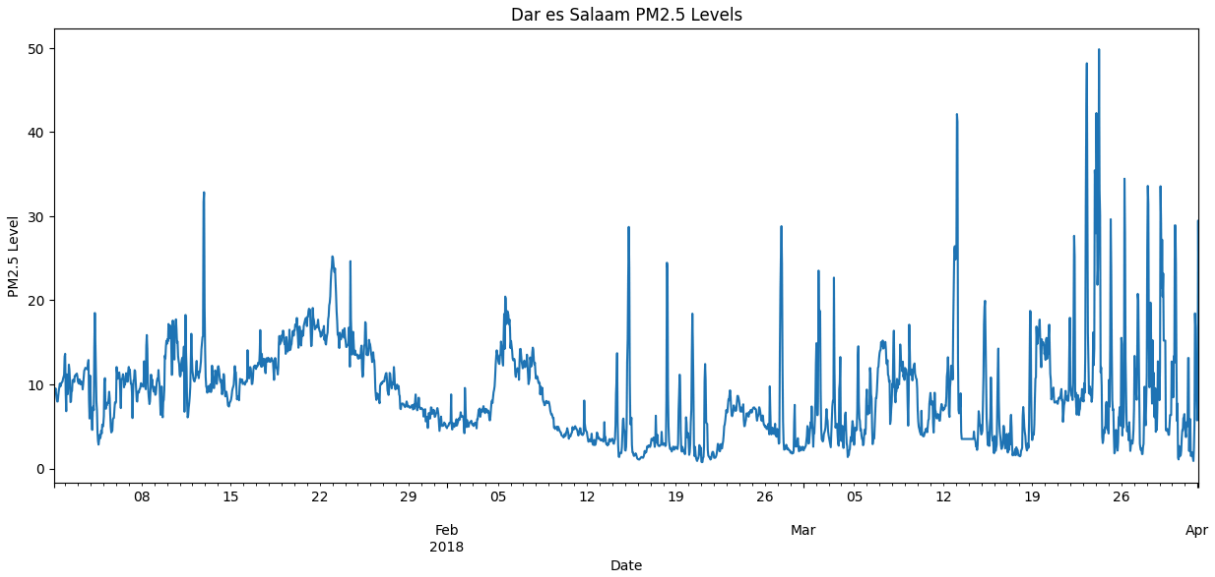
In [22]:
```
y = wrangle(dar)
print(type(y))
```

```
<class 'pandas.core.series.Series'>
```

Explore Some More

Task 3.5.5

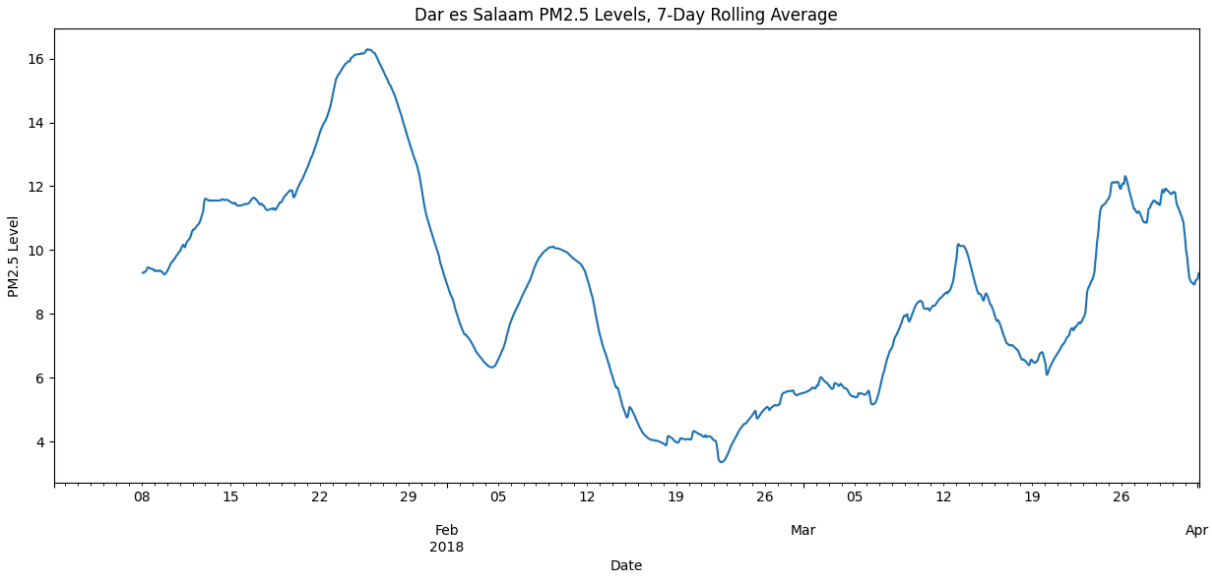
In [25]:
```
fig, ax = plt.subplots(figsize=(15, 6))

# use ax=ax in your plot
y.plot(xlabel="Date", ylabel="PM2.5 Level", title="Dar es Salaam PM2.5 Levels", ax=
```



Task 3.5.6

In [27]:
```
fig, ax = plt.subplots(figsize=(15, 6))

# use ax=ax in your plot
y.rolling(168).mean().plot(ax=ax, xlabel="Date", ylabel="PM2.5 Level", title="Dar e
```

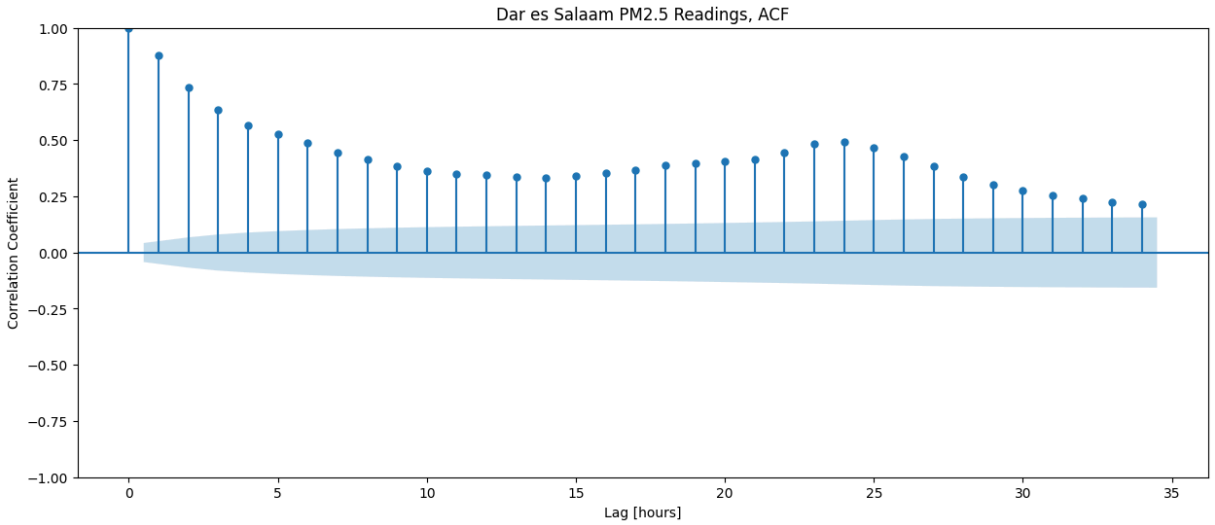
Out[27]:
```
<Axes: title={'center': 'Dar es Salaam PM2.5 Levels, 7-Day Rolling Average'}, xlab
el='Date', ylabel='PM2.5 Level'>
```



Task 3.5.7

In [34]:
```
fig, ax = plt.subplots(figsize=(15, 6))

# use ax=ax in your plot
plot_acf(y, ax=ax)
plt.xlabel("Lag [hours]")
plt.ylabel("Correlation Coefficient")
plt.title("Dar es Salaam PM2.5 Readings, ACF")
```

Out[34]:
```
Text(0.5, 1.0, 'Dar es Salaam PM2.5 Readings, ACF')
```

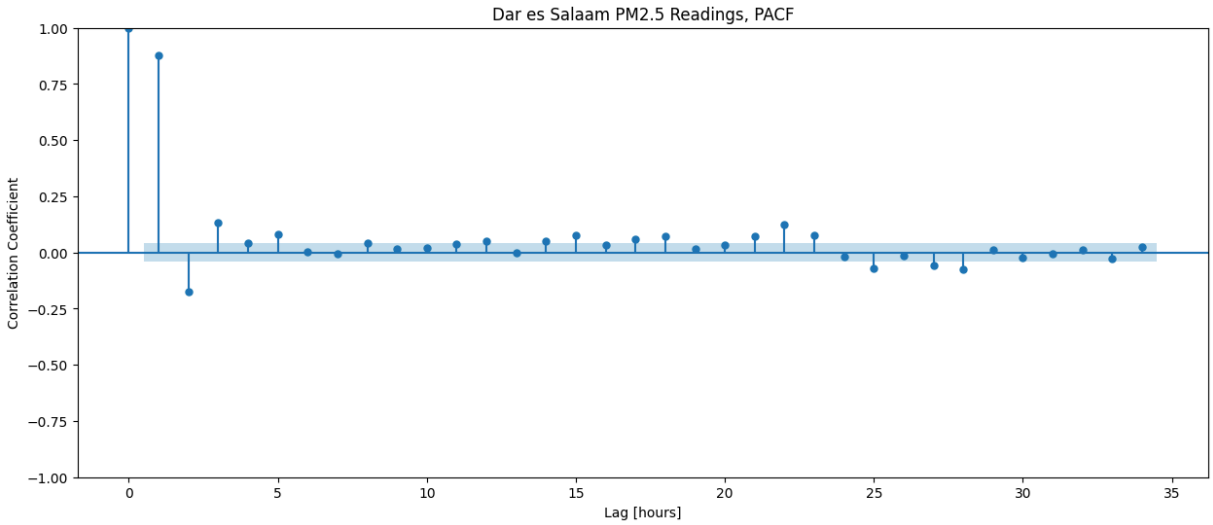


Task 3.5.8

In [36]:
```
fig, ax = plt.subplots(figsize=(15, 6))

# Use ax=ax in your plot
plot_pacf(y, ax=ax)
plt.xlabel("Lag [hours]")
```

```
plt.ylabel("Correlation Coefficient")
plt.title("Dar es Salaam PM2.5 Readings, PACF")
```

Out[36]: Text(0.5, 1.0, 'Dar es Salaam PM2.5 Readings, PACF')



Split

Task 3.5.9

In [38]:
```
cutoff_test = int(len(y) * 0.90)
y_train = y[:cutoff_test]
y_test = y[cutoff_test:]
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
y_train shape: (1944,)
y_test shape: (216,)
```

Build Model

Baseline

Task 3.5.10

In [40]:
```
y_train_mean = y_train.mean()
y_pred_baseline = [y_train_mean] * len(y_train)
mae_baseline = mean_absolute_error(y_train, y_pred_baseline)

print("Mean P2 Reading:", y_train_mean)
print("Baseline MAE:", mae_baseline)
```

```
Mean P2 Reading: 8.57142319061077
Baseline MAE: 4.053101181299159
```

Iterate

Task 3.5.11

> **Tip:** In this task, you'll need to combine the model you learned about in **Task 3.3.8** with the hyperparameter tuning technique you learned in **Task 3.4.9**.

In [54]:
```
# Create range to test different lags
p_params = range(1, 31)

# Create empty list to hold mean absolute error scores
maes = []

# Iterate through all values of p in `p_params`
for p in p_params:
    # Build model
    model = AutoReg(y_train, lags=p).fit()

    # Make predictions on training data, dropping null values caused by lag
    y_pred = model.predict().dropna()

    # Calculate mean absolute error for training data vs predictions
    mae = mean_absolute_error(y_train.iloc[p:], y_pred)

    # Append `mae` to list `maes`
    maes.append(mae)

# Put list `maes` into Series with index `p_params`
mae_series = pd.Series(maes, name="mae", index=p_params)

# Inspect head of Series
mae_series.head()
```

Out[54]:
```
1    1.059376
2    1.045182
3    1.032489
4    1.032147
5    1.031022
Name: mae, dtype: float64
```

Task 3.5.12

In [56]:
```
best_p = 26
best_model = AutoReg(y_train, lags=best_p).fit()
```

Task 3.5.13

In [60]:
```
y_train_resid = best_model.resid
y_train_resid.name = "residuals"
y_train_resid.head()
```

Out[60]:
```
timestamp
2018-01-02 05:00:00+03:00   -0.412913
2018-01-02 06:00:00+03:00    1.484934
2018-01-02 07:00:00+03:00    1.672359
2018-01-02 08:00:00+03:00   -0.368030
2018-01-02 09:00:00+03:00   -0.536868
Freq: H, Name: residuals, dtype: float64
```

Slight Code Change

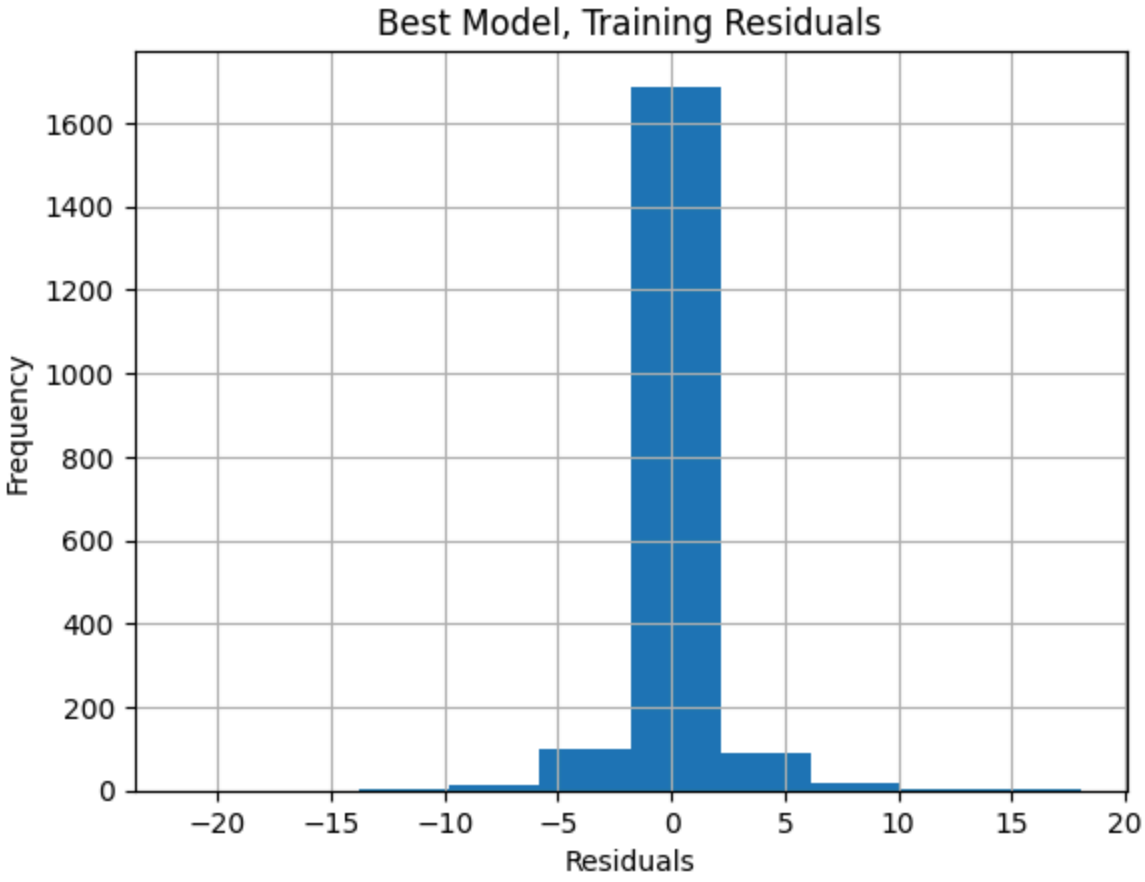
In the following task, you'll notice a small change in how plots are created compared to what you saw in the lessons. While the lessons use the global matplotlib method like `plt.plot(...)`, in this task, you are expected to use the object-oriented (OOP) API instead. This means creating your plots using `fig, ax = plt.subplots()` and then calling plotting methods on the `ax` object, such as `ax.plot(...)`, `ax.hist(...)`, or `ax.scatter(...)`.

If you're using pandas' or seaborn's built-in plotting methods (like `df.plot()` or `sns.lineplot()`), make sure to pass the `ax=ax` argument so that the plot is rendered on the correct axes.

This approach is considered best practice and will be used consistently across all graded tasks that involve matplotlib.

Task 3.5.14

In [62]:
```
# Plot histogram of residuals
fig, ax = plt.subplots()

# Use ax=ax in your plot
y_train_resid.hist(ax=ax)
ax.set_xlabel("Residuals")
ax.set_ylabel("Frequency")
ax.set_title("Best Model, Training Residuals");
```



Task 3.5.15

In [66]:
```
fig, ax = plt.subplots(figsize=(15, 6))

# Use ax=ax in your plot
plot_acf(y_train_resid, ax=ax)
ax.set_xlabel("Lag [hours]")
ax.set_ylabel("Correlation Coefficient")
ax.set_title("Dar es Salaam, Training Residuals ACF");
```

Evaluate

Task 3.5.16

In [80]:
```
# Pre-allocate Series with the same index (preserves Hourly freq)
y_pred_wfv = pd.Series(index=y_test.index, dtype=float, name="prediction")
y_pred_wfv.index.name = "timestamp"

history = y_train.copy()

for t, ts in enumerate(y_test.index):
    # Fit model
    model = AutoReg(history, lags=26).fit()

    # Forecast one step
    next_pred = model.forecast()[0]

    # Assign forecast at the correct timestamp
    y_pred_wfv.loc[ts] = next_pred

    # Update history with the true observed value
    history = pd.concat([history, y_test.iloc[[t]]])

print(y_pred_wfv.head())
```

```
timestamp
2018-03-23 03:00:00+03:00    10.414744
2018-03-23 04:00:00+03:00     8.269589
2018-03-23 05:00:00+03:00    15.178677
2018-03-23 06:00:00+03:00    33.475398
2018-03-23 07:00:00+03:00    39.571363
Freq: H, Name: prediction, dtype: float64
```

Task 3.5.17

In [82]:
```
# Enter y_pred_wfv at ... (Ellipsis) to see the test mean absolute error

test_mae = mean_absolute_error(y_test, y_pred_wfv)
print("Test MAE (walk forward validation):", round(test_mae, 2))
```

```
Test MAE (walk forward validation): 3.97
```

Communicate Results

Task 3.5.18

In [105…
```
# Build DataFrame with timestamp index
df_pred_test = pd.DataFrame({
    "y_test": y_test,
    "y_pred_wfv": y_pred_wfv
}, index=y_test.index)
```

```
# Plot with plotly express (wide mode)
fig = px.line(
    df_pred_test,
    x=df_pred_test.index,
    y=["y_test", "y_pred_wfv"]
)

# Update layout
fig.update_layout(
    title="Dar es Salaam, WFV Predictions",
    xaxis_title="Date",
    yaxis_title="PM2.5 Level"
)

fig.show()
```

Copyright 2024 WorldQuant University. This content is licensed solely for personal use. Redistribution or publication of this material is strictly prohibited.