3.5. Air Quality in Dar es Salaam TZ

```
import warnings

import wqet_grader

warnings.simplefilter(action="ignore", category=FutureWarning)
wqet_grader.init("Project 3 Assessment")
```

```
# 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

Task 3.5.1: Connect to MongoDB server running at host `"localhost"` on port `27017`. Then connect to the `"air-quality"` database and assign the collection for Dar es Salaam to the variable name `dar`.

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

Task 3.5.1: Connect to MongoDB server running at host `"localhost"` on port `27017`. Then connect to the `"air-quality"` database and assign the collection for Dar es Salaam to the variable name `dar`.

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

```
wqet_grader.grade("Project 3 Assessment", "Task 3.5.1", [dar.name])
```

Yup. You got it.

Score: 1

Explore

Task 3.5.2: Determine the numbers assigned to all the sensor sites in the Dar es Salaam collection. Your submission should be a list of integers.

```
sites = dar.distinct("metadata.site")
sites
```

```
[11, 23]
```

```
wqet_grader.grade("Project 3 Assessment", "Task 3.5.2", sites)
```

Excellent! Keep going.

Score: 1

Task 3.5.3: Determine which site in the Dar es Salaam collection has the most sensor readings (of any type, not just PM2.5 readings). You submission `readings_per_site` should be a list of dictionaries that follows this format:

```
[{'_id': 6, 'count': 70360}, {'_id': 29, 'count': 131852}]
```

Note that the values here ☝ are from the Nairobi collection, so your values will look different.

```
result = dar.aggregate(
    [
        {"$group": {"_id": "$metadata.site", "count": {"$count": {}}}},
```

Note that the values here ☝ are from the Nairobi collection, so your values will look different.

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

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

```
wqet_grader.grade("Project 3 Assessment", "Task 3.5.3", readings_per_site)
```

Way to go!

Score: 1

Import

Task 3.5.4: Create a `wrangle` function that will extract the PM2.5 readings from the site that has the most total readings in the Dar es Salaam collection. Your function should do the following steps:

1. Localize reading time stamps to the timezone for `"Africa/Dar_es_Salaam"`.
2. Remove all outlier PM2.5 readings that are above 100.
3. Resample the data to provide the mean PM2.5 reading for each hour.
4. Impute any missing values using the forward-fill method.
5. Return a Series `y`.

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

    # Read
    df = pd.DataFrame(results).set_index("timestamp")

    #1.Localize
    df.index = df.index.tz_localize("UTC").tz_convert("Africa/Dar_es_Salaam")
```

```
    #1.Localize
    df.index = df.index.tz_localize("UTC").tz_convert("Africa/Dar_es_Salaam")

    #2.Remove
    df = df[df["P2"] <= 100]

    #3.Resample & 4.Impute
    y = df["P2"].resample("1H").mean().fillna(method='ffill')

    return y
```

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

```
y = wrangle(dar)
y.head()
```

```
timestamp
2018-01-01 03:00:00+03:00    9.456327
2018-01-01 04:00:00+03:00    9.400833
2018-01-01 05:00:00+03:00    9.331458
2018-01-01 06:00:00+03:00    9.528776
2018-01-01 07:00:00+03:00    8.861250
Freq: H, Name: P2, dtype: float64
```

```
wqet_grader.grade("Project 3 Assessment", "Task 3.5.4", wrangle(dar))
```

Party time! 🎉 🎉 🎉

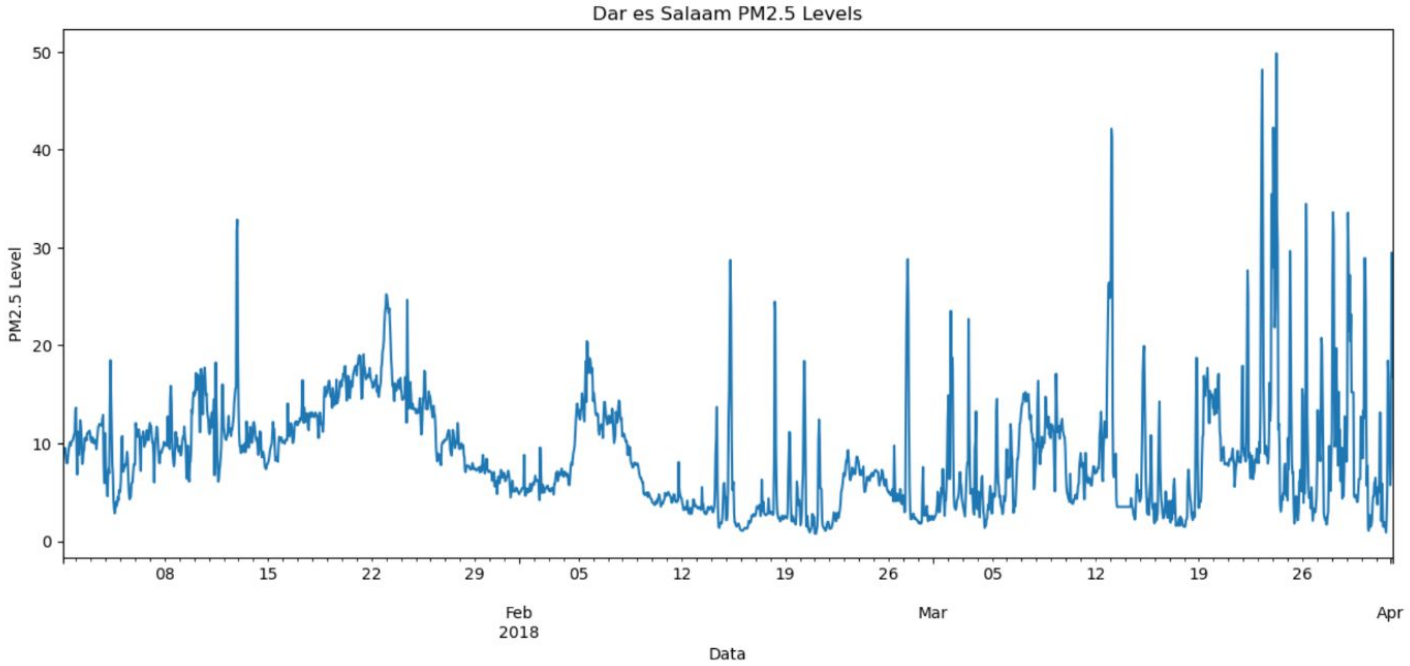
Score: 1

Explore Some More

Task 3.5.5: Create a time series plot of the readings in `y`. Label your x-axis `"Date"` and your y-axis `"PM2.5 Level"`. Use the title `"Dar es Salaam PM2.5 Levels"`.

```
fig, ax = plt.subplots(figsize=(15, 6))
y.plot(xlabel="Data", ylabel="PM2.5 Level", title="Dar es Salaam PM2.5 Levels", ax=ax)
# Don't delete the code below 👇
plt.savefig("images/3-5-5.png", dpi=150)
```

```
fig, ax = plt.subplots(figsize=(15, 6))
y.plot(xlabel="Data", ylabel="PM2.5 Level", title="Dar es Salaam PM2.5 Levels", ax=ax)
# Don't delete the code below 👇
plt.savefig("images/3-5-5.png", dpi=150)
```



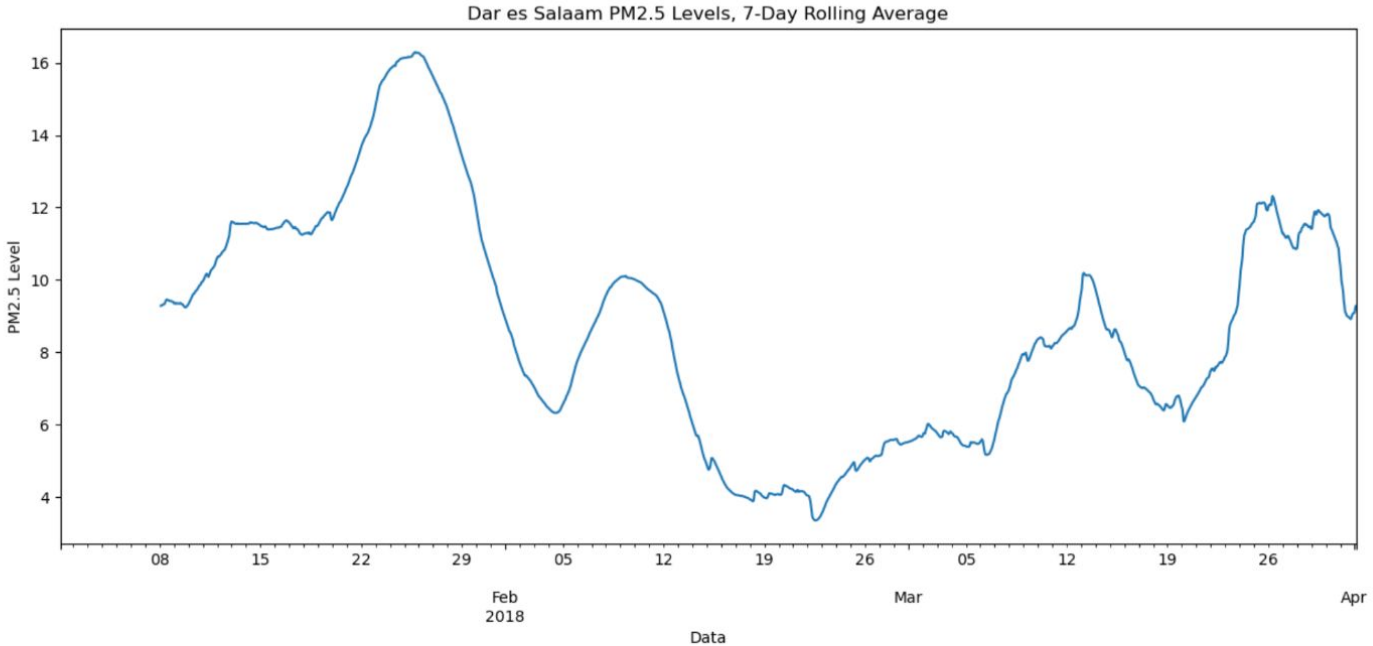

```
with open("images/3-5-5.png", "rb") as file:
    wqet_grader.grade("Project 3 Assessment", "Task 3.5.5", file)
```

Python master 😁

Score: 1

Task 3.5.6: Plot the rolling average of the readings in `y`. Use a window size of `168` (the number of hours in a week). Label your x-axis `"Date"` and your y-axis `"PM2.5 Level"`. Use the title `"Dar es Salaam PM2.5 Levels, 7-Day Rolling Average"`.

```
fig, ax = plt.subplots(figsize=(15, 6))
y.rolling(168).mean().plot(ax=ax,xlabel="Data", ylabel="PM2.5 Level", title="Dar es Salaam PM2.5 Levels, 7-Day Rolling Average")
# Don't delete the code below 👇
plt.savefig("images/3-5-6.png", dpi=150)
```



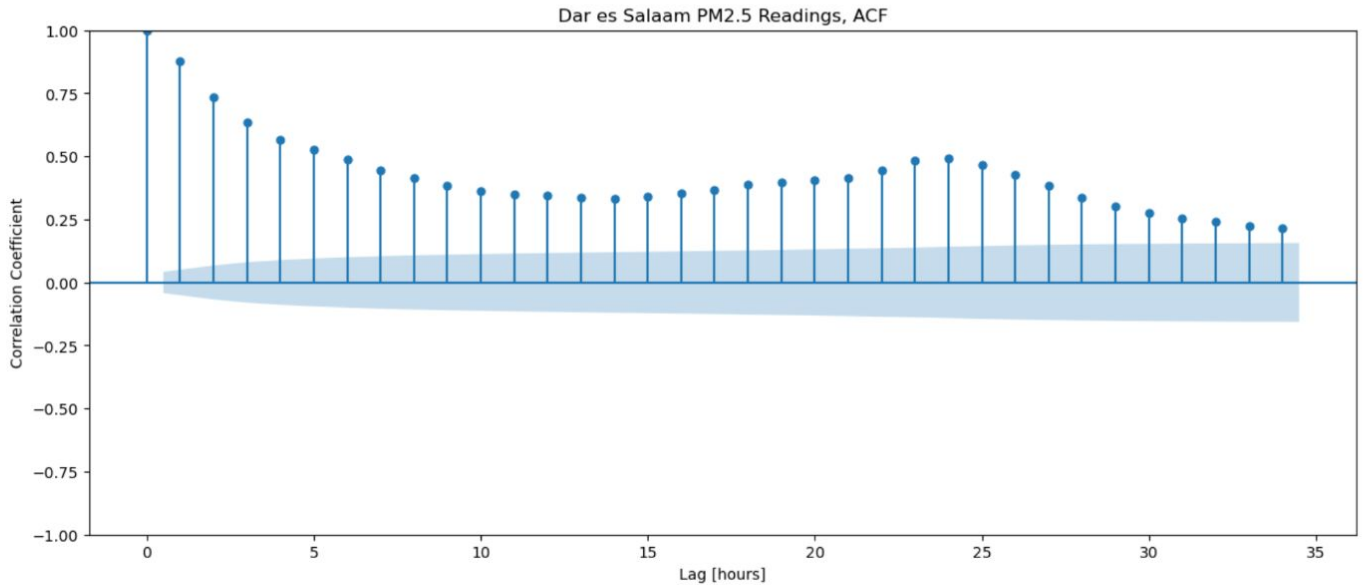

```
with open("images/3-5-6.png", "rb") as file:
    wqet_grader.grade("Project 3 Assessment", "Task 3.5.6", file)
```

Excellent work.

Score: 1

Task 3.5.7: Create an ACF plot for the data in `y`. Be sure to label the x-axis as `"Lag [hours]"` and the y-axis as `"Correlation Coefficient"`. Use the title `"Dar es Salaam PM2.5 Readings, ACF"`.

```
fig, ax = plt.subplots(figsize=(15, 6))
plot_acf(y, ax=ax)
plt.xlabel("Lag [hours]")
plt.ylabel("Correlation Coefficient")
plt.title("Dar es Salaam PM2.5 Readings, ACF")
# Don't delete the code below 👇
plt.savefig("images/3-5-7.png", dpi=150)
```



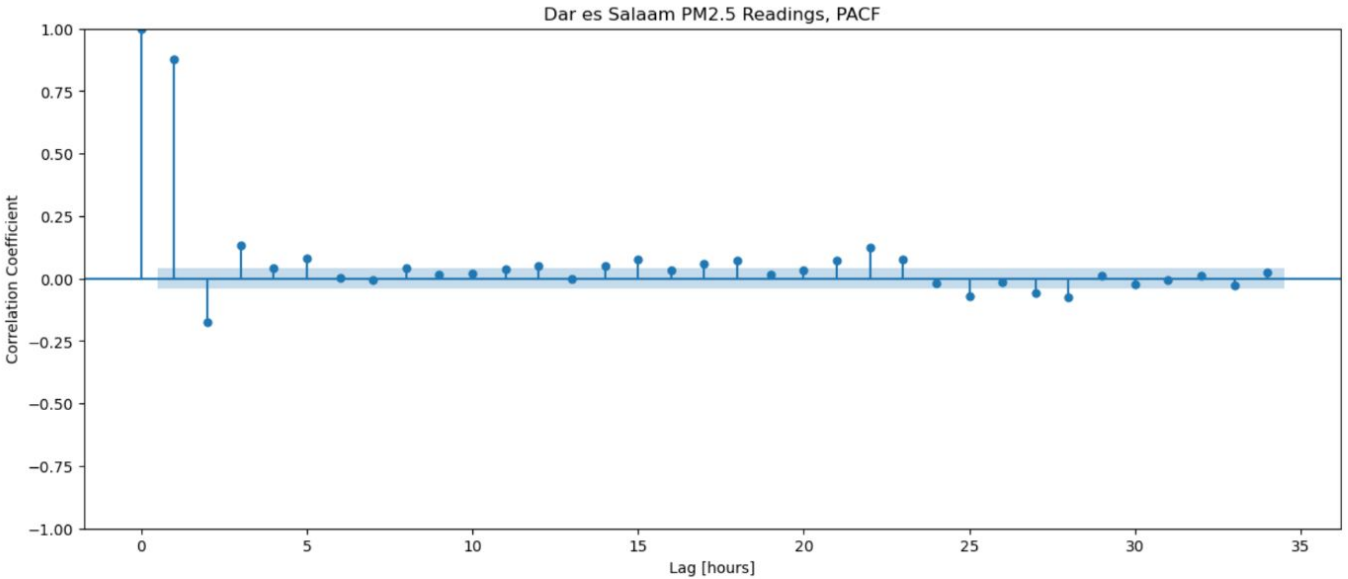

```
with open("images/3-5-7.png", "rb") as file:
    wqet_grader.grade("Project 3 Assessment", "Task 3.5.7", file)
```

Good work!

Score: 1

Task 3.5.8: Create an PACF plot for the data in `y`. Be sure to label the x-axis as `"Lag [hours]"` and the y-axis as `"Correlation Coefficient"`. Use the title `"Dar es Salaam PM2.5 Readings, PACF"`.

```
fig, ax = plt.subplots(figsize=(15, 6))
plot_pacf(y, ax=ax)
plt.xlabel("Lag [hours]")
plt.ylabel("Correlation Coefficient")
plt.title("Dar es Salaam PM2.5 Readings, PACF")

# Don't delete the code below 👇
plt.savefig("images/3-5-8.png", dpi=150)
```




```
with open("images/3-5-8.png", "rb") as file:
    wqet_grader.grade("Project 3 Assessment", "Task 3.5.8", file)
```

Python master 😁

Score: 1

Split

Task 3.5.9: Split `y` into training and test sets. The first 90% of the data should be in your training set. The remaining 10% should be in the test set.

```
cutoff_test = int(len(y) * 0.90)

y_train = y.iloc[:cutoff_test]
y_test = y.iloc[cutoff_test:]

print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

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

```
wqet_grader.grade("Project 3 Assessment", "Task 3.5.9a", y_train)
```

Excellent! Keep going.

Score: 1

```
wqet_grader.grade("Project 3 Assessment", "Task 3.5.9b", y_test)
```

Correct.

Score: 1

Build Model

Baseline

Task 3.5.10: Establish the baseline mean absolute error for your model.

```
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)
```

Baseline

Task 3.5.10: Establish the baseline mean absolute error for your model.

```
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
```

```
wqet_grader.grade("Project 3 Assessment", "Task 3.5.10", mae_baseline)
```

Yes! Keep on rockin'. 🎸 That's right.

Score: 1

Iterate

Task 3.5.11: You're going to use an `AutoReg` model to predict PM2.5 readings, but which hyperparameter settings will give you the best performance? Use a `for` loop to train your AR model on using settings for `lags` from 1 to 30. Each time you train a new model, calculate its mean absolute error and append the result to the list `maes`. Then store your results in the Series `mae_series`.

> **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**.

```
# 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
```

```
# 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()
```

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

```
wqet_grader.grade("Project 3 Assessment", "Task 3.5.11", mae_series)
```

That's the right answer. Keep it up!

Score: 1

Task 3.5.12: Look through the results in `mae_series` and determine what value for `p` provides the best performance. Then build and train `best_model` using the best hyperparameter value.

Note: Make sure that you build and train your model in one line of code, and that the data type of `best_model` is `statsmodels.tsa.ar_model.AutoRegResultsWrapper`.

```
best_p = ARIMA(y_train).fit()
best_model = AutoReg(y_train, lags=1).fit()
```

```
wqet_grader.grade(
```

```
wqet_grader.grade(
    "Project 3 Assessment", "Task 3.5.12", [isinstance(best_model.model, AutoReg)]
)
```

Awesome work.

Score: 1

Task 3.5.13: Calculate the training residuals for `best_model` and assign the result to `y_train_resid`. **Note** that your name of your Series should be `"residuals"`.

```
y_train_resid = y_train - y_pred
y_train_resid.name = "residuals"
y_train_resid.head()
```

```
timestamp
2018-01-01 03:00:00+03:00   NaN
2018-01-01 04:00:00+03:00   NaN
2018-01-01 05:00:00+03:00   NaN
2018-01-01 06:00:00+03:00   NaN
2018-01-01 07:00:00+03:00   NaN
Freq: H, Name: residuals, dtype: float64
```

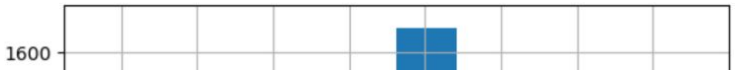
```
wqet_grader.grade("Project 3 Assessment", "Task 3.5.13", y_train_resid.tail(1500))
```

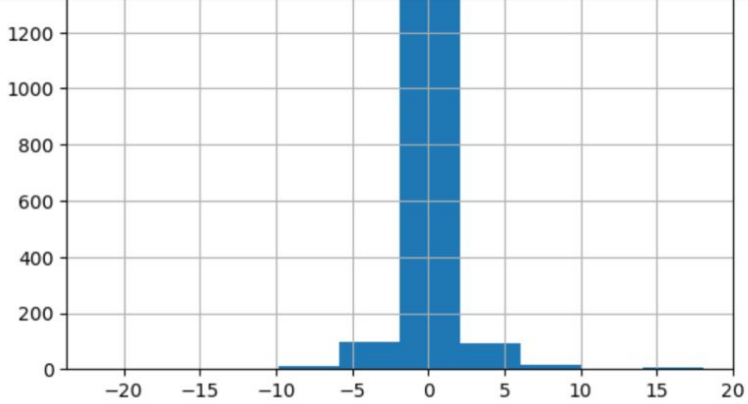
🥳

Score: 1

Task 3.5.14: Create a histogram of `y_train_resid`. Be sure to label the x-axis as `"Residuals"` and the y-axis as `"Frequency"`. Use the title `"Best Model, Training Residuals"`.

```
# Plot histogram of residuals
y_train_resid.hist()
# Don't delete the code below 👇
plt.savefig("images/3-5-14.png", dpi=150)
```





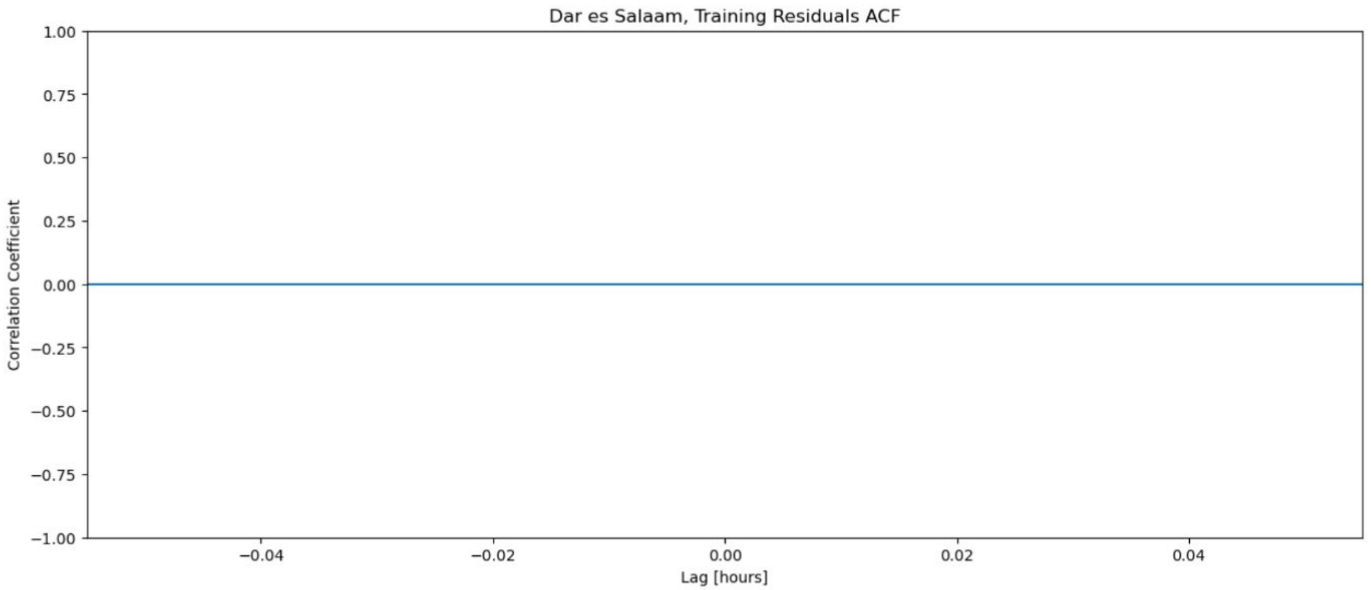

```
with open("images/3-5-14.png", "rb") as file:
    wqet_grader.grade("Project 3 Assessment", "Task 3.5.14", file)
```

You're making this look easy. 😊

Score: 1

Task 3.5.15: Create an ACF plot for `y_train_resid`. Be sure to label the x-axis as `"Lag [hours]"` and y-axis as `"Correlation Coefficient"`. Use the title `"Dar es Salaam, Training Residuals ACF"`.

```
fig, ax = plt.subplots(figsize=(15, 6))
plot_acf(y_train_resid, ax=ax)

plt.xlabel("Lag [hours]")
plt.ylabel("Correlation Coefficient")
plt.title("Dar es Salaam, Training Residuals ACF")

# Don't delete the code below 👇
plt.savefig("images/3-5-15.png", dpi=150)
```




```
with open("images/3-5-15.png", "rb") as file:
    wqet_grader.grade("Project 3 Assessment", "Task 3.5.15", file)
```

Boom! You got it.

Score: 1

Evaluate

Task 3.5.16: Perform walk-forward validation for your model for the entire test set `y_test`. Store your model's predictions in the Series `y_pred_wfv`. Make sure the name of your Series is `"prediction"` and the name of your Series index is `"timestamp"`.

```
y_pred_wfv = pd.Series()
history = y_train.copy()

for i in range(len(y_test)):
```

```
]: y_pred_wfv = pd.Series()
history = y_train.copy()

for i in range(len(y_test)):
    model = AutoReg(history, lags=26).fit()
    next_pred = model.forecast()
    history = history.append(y_test[next_pred.index])
    y_pred_wfv = y_pred_wfv.append(next_pred)

y_pred_wfv.name = "prediction"
y_pred_wfv.index.name = "timestamp"
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
```

```
]: wqet_grader.grade("Project 3 Assessment", "Task 3.5.16", y_pred_wfv)
```

🥳

Score: 1

Task 3.5.17: Submit your walk-forward validation predictions to the grader to see the test mean absolute error for your model.

```
]: wqet_grader.grade("Project 3 Assessment", "Task 3.5.17", y_pred_wfv)
```

Your model's mean absolute error is `3.776`. 🥳

Score: 1

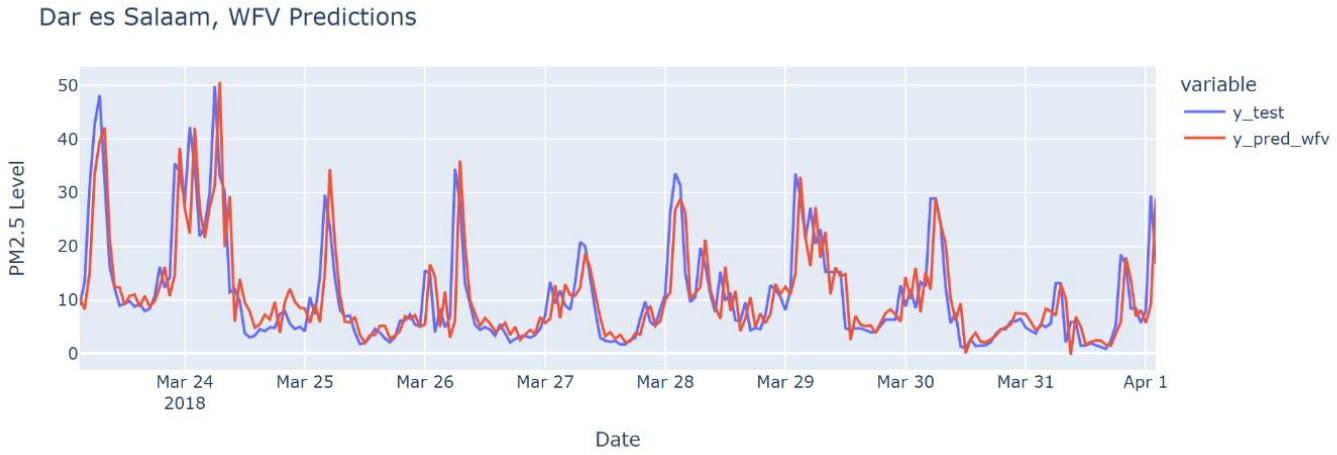
Communicate Results

Task 3.5.18: Put the values for `y_test` and `y_pred_wfv` into the DataFrame `df_pred_test` (don't forget the index). Then plot `df_pred_test` using

Communicate Results

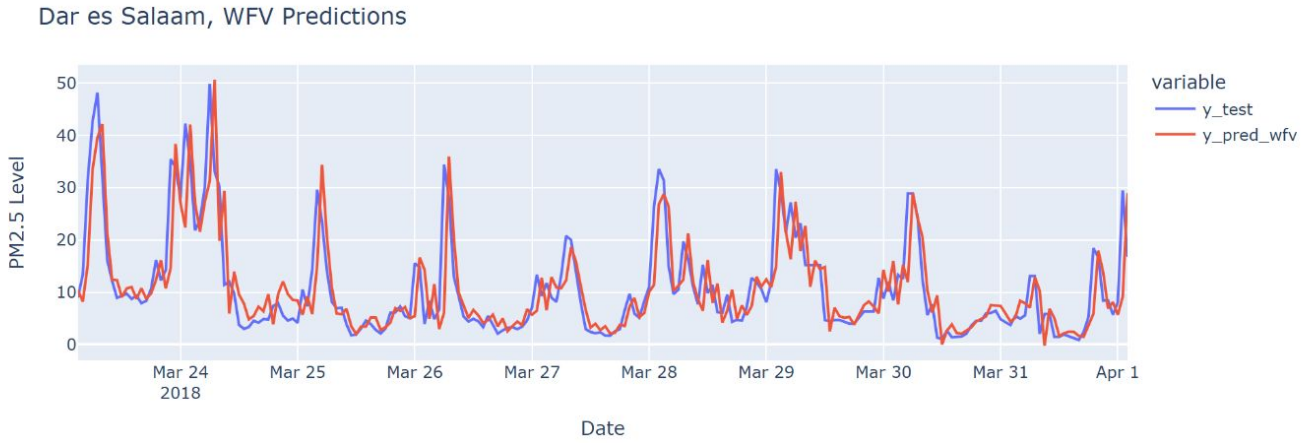
Task 3.5.18: Put the values for `y_test` and `y_pred_wfv` into the DataFrame `df_pred_test` (don't forget the index). Then plot `df_pred_test` using plotly express. In the legend, your lines should be labeled `"y_test"` and `"y_pred_wfv"`. Be sure to label the x-axis as `"Date"` and the y-axis as `"PM2.5 Level"`. Use the title `"Dar es Salaam, WFV Predictions"`.

```
df_pred_test = pd.DataFrame(
    {"y_test":y_test, "y_pred_wfv":y_pred_wfv}
)
fig = px.line(df_pred_test, labels={"value":"PM2.5"})
fig.update_layout(
    title="Dar es Salaam, WFV Predictions",
    xaxis_title="Date",
    yaxis_title="PM2.5 Level",
)
# Don't delete the code below 👇
fig.write_image("images/3-5-18.png", scale=1, height=500, width=700)

fig.show()
```




```
with open("images/3-5-18.png", "rb") as file:
```

```
    {"y_test":y_test, "y_pred_wfv":y_pred_wfv}
)
fig = px.line(df_pred_test, labels={"value":"PM2.5"})
fig.update_layout(
    title="Dar es Salaam, WFV Predictions",
    xaxis_title="Date",
    yaxis_title="PM2.5 Level",
)
# Don't delete the code below 👇
fig.write_image("images/3-5-18.png", scale=1, height=500, width=700)

fig.show()
```



2]:
```
with open("images/3-5-18.png", "rb") as file:
    wqet_grader.grade("Project 3 Assessment", "Task 3.5.18", file)
```

Way to go!

Score: 1