Part 2 Time Series Feature Engineering

By Josh Houlding

Feature engineering is the process of adjusting the representation of the data to improve the efficacy of the model. In time series, data scientists construct the output of their model by identifying the variable that they need to predict at a future time (ex: future energy demand or load next month) and then leverage historical data and feature engineering to create input variables that will be used to make predictions for that future date. For this activity, in 500-750 words, answer the following:

1. Discuss the main goals/benefits of performing feature engineering on Time-Series data.
2. Perform the following types of Features on your selected time-series dataset and report the results of each:
    - Date Time Features: from the Date column, "Feature Extract" three additional columns to your data frame: one for the Year, one for the Month, and one for the Day. Show the results.
    - Lag Features: use the shift function to "Feature Extract" three additional columns: same day last week, same day last month, same day last year. Show the results.
    - Window Features: Use the rolling method to "Feature Extract" an additional column that shows a 2-month rolling average. Show the results.
    - Expanding Feature: here, we're not considering window size. We want to consider all the values in the data frame. Use the expanding method to "Feature Extract" an additional column that shows the maximum value till date. Show the results of the data frame.
3. Discuss some additional insights you gained from leveraging the additional knowledge you performed in the previous step. How can this help you build a better time series forecasting solution as a data scientist?
4. "Feature Extract" an additional column called "Q" to show the quarterly data of your data frame by using the resample function. Show the results. Hint: call the mean of the resample function.
5. Perform the same step you did in step 4, but show the Yearly data in this step.

APA style is not required, but solid academic writing is expected. This assignment uses a rubric. Please review the rubric prior to beginning the assignment to become familiar with the expectations for successful completion. You are not required to submit this assignment to LopesWrite.

Task 1

Discuss the main goals/benefits of performing feature engineering on Time-Series data.

Feature engineering is essential for time-series analysis because it can be used to convert dates into three separate features (year, month and day), which can then be used as inputs in a model to predict outputs. In this case, we are trying to predict pollution levels over time, so being able to use the year, month and day as inputs is essential.

Task 2

Perform the following types of Features on your selected time-series dataset and report the results of each:

- Date Time Features: from the Date column, "Feature Extract" three additional columns to your data frame: one for the Year, one for the Month, and one for the Day. Show the results.
- Lag Features: use the shift function to "Feature Extract" three additional columns: same day last week, same day last month, same day last year. Show the results.
- Window Features: Use the rolling method to "Feature Extract" an additional column that shows a 2-month rolling average. Show the results.
- Expanding Feature: here, we're not considering window size. We want to consider all the values in the data frame. Use the expanding method to "Feature Extract" an additional column that shows the maximum value till date. Show the results of the data frame.

In [26]:
```
# Import libraries
import pandas as pd
```

In [27]:
```
# Import dataset
df = pd.read_excel("AirQualityUCI.xlsx")

# View data
df.head(3)
```

Out[27]:

	date	time	true_co	sensor_co	true_nmhc	true_benzene	sensor_nmhc	true_nox	sensor_nox	true_no2	sensor_no2	true_o3	temp	r
0	2004-03-10	18:00:00	2.6	1360.00	150.0	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.6	
1	2004-03-10	19:00:00	2.0	1292.25	112.0	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.3	
2	2004-03-10	20:00:00	2.2	1402.00	88.0	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.9	

Date Time Features: from the Date column, "Feature Extract" three additional columns to your data frame: one for the Year, one for the Month, and one for the Day. Show the results.

In [28]:
```
# Extract year, month and day from "date"
df.insert(0, "year", df["date"].dt.year)
df.insert(1, "month", df["date"].dt.month)
df.insert(2, "day", df["date"].dt.day)

# Show results
df.head(3)
```

Out[28]:

	year	month	day	date	time	true_co	sensor_co	true_nmhc	true_benzene	sensor_nmhc	true_nox	sensor_nox	true_no2	sensor_no
0	2004	3	10	2004-03-10	18:00:00	2.6	1360.00	150.0	11.881723	1045.50	166.0	1056.25	113.0	1692.0
1	2004	3	10	2004-03-10	19:00:00	2.0	1292.25	112.0	9.397165	954.75	103.0	1173.75	92.0	1558.7
2	2004	3	10	2004-03-10	20:00:00	2.2	1402.00	88.0	8.997817	939.25	131.0	1140.00	114.0	1554.5

Lag Features: use the shift function to "Feature Extract" three additional columns: same day last week, same day last month, same day last year. Show the results.

In [29]:
```
# Add column for same day last week
df.insert(3, "same_day_last_year", df["date"] - pd.DateOffset(years=1))
```

```
df.insert(4, "same_day_last_month", df["date"] - pd.DateOffset(months=1))
df.insert(5, "same_day_last_week", df["date"] - pd.DateOffset(weeks=1))

# Move "date" to before same day columns
date = df.pop("date")
df.insert(3, "date", date)

# Show results
df.head(3)
```

Out[29]:

	year	month	day	date	same_day_last_year	same_day_last_month	same_day_last_week	time	true_co	sensor_co	...	true_benzene	s
0	2004	3	10	2004-03-10	2003-03-10	2004-02-10	2004-03-03	18:00:00	2.6	1360.00	...	11.881723	
1	2004	3	10	2004-03-10	2003-03-10	2004-02-10	2004-03-03	19:00:00	2.0	1292.25	...	9.397165	
2	2004	3	10	2004-03-10	2003-03-10	2004-02-10	2004-03-03	20:00:00	2.2	1402.00	...	8.997817	

3 rows × 21 columns

Window Features: Use the rolling method to "Feature Extract" an additional column that shows a 2-month rolling average. Show the results.

It is not specified which column a 2-month rolling average column must be created for, so I created one for every numeric variable in the dataset. The new rolling average columns all follow this naming convention: `<originalcolumnname_2mra>`, where "2mra" stands for 2-month rolling average.

In [30]:
```
# Set "date" as index
df.set_index("date", inplace=True)
```

In [31]:
```
# Calculate 2-month rolling averages for numeric columns
numeric_columns = df.select_dtypes(include="number").columns
for col in numeric_columns:
    df[col + "_2mra"] = df[col].rolling(window=60).mean()

# Show results
```

```
new_columns = df.iloc[:, -16:]
new_columns.tail(3)
```

Out[31]:

date	year_2mra	month_2mra	day_2mra	true_co_2mra	sensor_co_2mra	true_nmhc_2mra	true_benzene_2mra	sensor_nmhc_2mra	true_nox_
2005-04-04	2005.0	4.0	2.833333	1.222546	970.777778	218.811816	4.392192	700.994444	176.82
2005-04-04	2005.0	4.0	2.866667	1.239212	973.527778	218.811816	4.518891	707.336111	177.96
2005-04-04	2005.0	4.0	2.900000	1.265879	977.452778	218.811816	4.692441	715.686111	180.68

Expanding Feature: here, we're not considering window size. We want to consider all the values in the data frame. Use the expanding method to "Feature Extract" an additional column that shows the maximum value till date. Show the results of the data frame.

Once again, we will create a new expanding maximum for each numeric column. They follow the syntax `<originalcolumnname>_exp_max`.

In [32]:

```
# Calculate expanding maximum for numeric columns
for col in numeric_columns:
    df[col + "_exp_max"] = df[col].expanding().max()

# Show results
new_columns = df.iloc[:, -16:]
new_columns.head(3)
```

Out[32]:

date	year_exp_max	month_exp_max	day_exp_max	true_co_exp_max	sensor_co_exp_max	true_nmhc_exp_max	true_benzene_exp_max	senso
2004-03-10	2004.0	3.0	10.0	2.6	1360.0	150.0	11.881723	
2004-03-10	2004.0	3.0	10.0	2.6	1360.0	150.0	11.881723	
2004-03-10	2004.0	3.0	10.0	2.6	1402.0	150.0	11.881723	

Task 3

Discuss some additional insights you gained from leveraging the additional knowledge you performed in the previous step. How can this help you build a better time series forecasting solution as a data scientist?

The `year`, `month`, and `day` features are essential for analyzing trends over time, as a column in datetime format cannot be directly analyzed. The lag features we introduced enable us to identify seasonal patterns in the data and infer how various aspects of each season contribute to pollution levels. The rolling averages we calculated are helpful for smoothing out short-term fluctuations and thus giving us a clearer picture of long-term trends. Finally, the expanding maximums are useful for tracking performance metrics and setting comparison benchmarks.

Task 4

"Feature Extract" an additional column called "Q" to show the quarterly data of your data frame by using the resample function. Show the results. Hint: call the mean of the resample function.

In [33]:
```
# Resample data quarterly and store in column "Q"
for col in numeric_columns:
    col_quarterly_mean = df[col].resample("Q").mean()
    df[col + "_Q"] = col_quarterly_mean

# Show results
```

```
new_columns = df.iloc[:, -16:]
new_columns.head(3)
```

Out[33]:

date	year_Q	month_Q	day_Q	true_co_Q	sensor_co_Q	true_nmhc_Q	true_benzene_Q	sensor_nmhc_Q	true_nox_Q	sensor_nox_Q	true_no2
2004-03-10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2004-03-10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2004-03-10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

We see some null values for these quarterly columns. Let's do some digging to see if they are all null, or if just some of them are.

In [34]:
```
# Show how many rows have null quarterly values
print(f"Total rows: {len(new_columns)}")
null_rows = new_columns[new_columns.notna().any(axis=1)]
print(f"Rows with no null quarterly values: {len(null_rows)}")
```

```
Total rows: 9357
Rows with no null quarterly values: 120
```

There are 120 rows with no null quarterly values, suggesting that there has not been an error in calculating these values.

In [35]:
```
# Show some of the non-null columns
new_columns[new_columns.notna().any(axis=1)].head(3)
```

Out[35]:

date	year_Q	month_Q	day_Q	true_co_Q	sensor_co_Q	true_nmhc_Q	true_benzene_Q	sensor_nmhc_Q	true_nox_Q	sensor_nox_Q	true_ı
2004-03-31	2004.0	3.0	20.870588	2.287156	1222.685784	181.138465	9.935104	935.540686	147.953701	1029.058987	103.
2004-03-31	2004.0	3.0	20.870588	2.287156	1222.685784	181.138465	9.935104	935.540686	147.953701	1029.058987	103.
2004-03-31	2004.0	3.0	20.870588	2.287156	1222.685784	181.138465	9.935104	935.540686	147.953701	1029.058987	103.

We now see some values that are not null.

Task 5

Perform the same step you did in step 4, but show the Yearly data in this step.

In [36]:
```
# Resample data yearly and store in column "Y"
for col in numeric_columns:
    col_yearly_mean = df[col].resample("Y").mean()
    df[col + "_Y"] = col_quarterly_mean

# Show results
new_columns = df.iloc[:, -16:]
new_columns.tail(3)
```

Out[36]:

date	year_Y	month_Y	day_Y	true_co_Y	sensor_co_Y	true_nmhc_Y	true_benzene_Y	sensor_nmhc_Y	true_nox_Y	sensor_nox_Y	true_no2_Y
2005-04-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2005-04-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2005-04-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [37]:
```
# Show some of the non-null columns
new_columns[new_columns.notna().any(axis=1)].head(3)
```

Out[37]:

date	year_Y	month_Y	day_Y	true_co_Y	sensor_co_Y	true_nmhc_Y	true_benzene_Y	sensor_nmhc_Y	true_nox_Y	sensor_nox_Y	true_no
2004-03-31	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789
2004-03-31	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789
2004-03-31	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789412	0.789

In [38]:
```
# Export modified dataset
df.to_excel("ModifiedAirQualityUCI.xlsx", index=True)
```

References

ChatGPT. (n.d.). https://chat.openai.com/