Report

Sections required in your report:

- Main objective of the analysis that also specifies whether your model will be focused on a specific type of Time Series, Survival Anlaysis, or Deep Learning and the benefits that your analysis brings to the business or stakeholders of this data.

- Brief description of the data set you chose, a summary of its attributes, and an outline of what you are trying to accomplish with this analysis.

- Brief summary of data exploration and actions taken for data cleaning or feature engineering.

- Summary of training at least three variations of the Time Series, Survival Analysis, or Deep Learning model you selected. For example, you can use different models or different hyperparameters.

- A paragraph explaining which of your models you recommend as a final model that best fits your needs in terms of accuracy or explainability.

- Summary Key Findings and Insights, which walks your reader through the main findings of your modeling exercise.

- Suggestions for next steps in analyzing this data, which may include suggesting revisiting this model or adding specific data features to achieve a better model.

1. Main objective of the analysis that also specifies whether your model will be focused on a specific type of Time Series, Survival Anlaysis, or Deep Learning and the benefits that your analysis brings to the business or stakeholders of this data.

Main objective of the project is to build best posible prediction of PM 2.5 daily levels for Ursynow, a district at Warsaw.
https://en.wikipedia.org/wiki/Particulate_pollution
The knowledge provided from predictions can ba used as a tool by authorities to impose actions aimed for a reduction of daily PM 2.5 levels.
One simple example having probably quite quick impact is a rule to use cars of either odd or even plate numbers depening on what calendar day it is and what predicted PM 2.5 level would be.

Due to characteristics of specific deep learning models - namely Recurrent Neural Networks (RNN) and Long-Short Term Memory (LSTM) - I decided to try both for time series forcasting.

2. Brief description of the data set you chose, a summary of its attributes, and an outline of what you are trying to accomplish with this analysis.

The data set used comes from https://aqicn.org/

It shows daily averages of:

- PM 2.5
- PM 10
- O3
- NO2
- SO2
- CO

measured at Ursynów, Warszawa, Mazowieckie for last 85 months.
Daily measures are based on the 24 hours average of hourly readings.
The source for data is Regional Inspectorate for Environmental Protection at Warsaw (Wojewódzki Inspektorat Ochrony Środowiska w Warszawie).

As a result of my analysis I hope to show that forcasting levels of PM 2.5 is possible.
I hope to encourage authorities to provide forcasts to general public so citizens - with some degree of uncertenity - would know what to expect in comming day or days.
Citizens suffering from respiratory system diseases might be a group of highest interest of such forcasts.

From an inital summary of data set attributes we can tell that data set suffers from some NULL values:

In [56]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2475 entries, 2013-12-31 to 2021-01-01
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   pm25    2432 non-null   float64
 1   pm10    2433 non-null   float64
 2   o3      2441 non-null   float64
 3   no2     2417 non-null   float64
 4   so2     1628 non-null   float64
 5   co      588 non-null    float64
dtypes: float64(6)
memory usage: 135.4 KB
```

More details on GitHub

3. Brief summary of data exploration and actions taken for data cleaning or feature engineering.

Since my goal is to predic daily levels of [pm25] any preprocessing will focus mainly on that feature.

NULL values

The data set suffers from NULL values what have to be dealt with:

In [57]:
```
data.isna().sum()
```

Out[57]:
```
pm25      43
pm10      42
```

```
o3         34
no2        58
so2       847
co       1887
dtype: int64
```

Interpolation

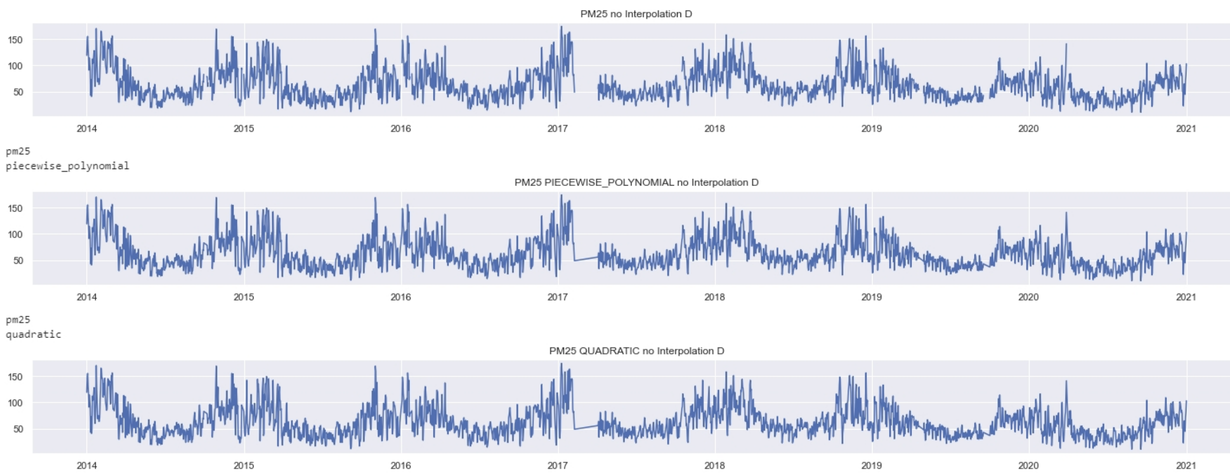
Since I am working with time series for a method of data imputation I have chosen Interpolation.
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.interpolate.html

The effects of two exemplary interpolations methods out of many available algorithms are shown below.
At the end I used quadratic method.

In [58]:

```python
Image(filename='pm25_interpolation.jpg')
```

Out[58]:



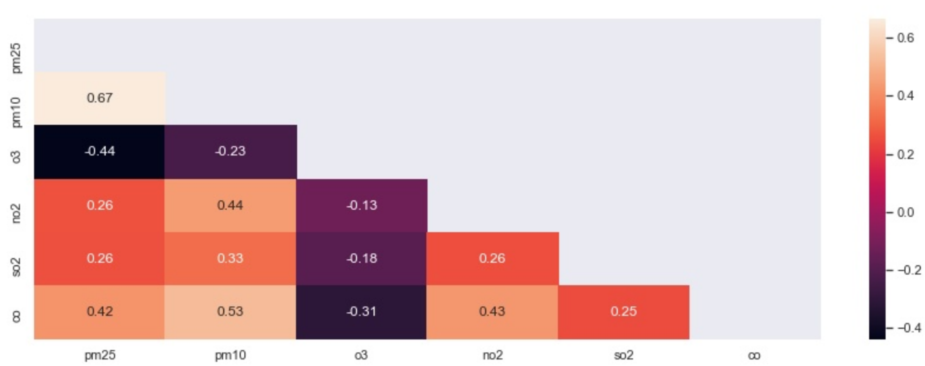
Correlations

Out of simple curiosity I had a brief look into correlations between all available features:

In [59]:

```python
Image(filename='ini_corr_matrix.jpg')
```

Out[59]:



Log normalization

To normalize distribution of series I have transformed them with a use of np.log1p function.

In [60]:

```python
df_transformed.head()
```

Out[60]:

date	pm25	pm10	o3	no2	so2	co
2013-12-31	nan	3.738	2.708	1.946	1.609	1.609
2014-01-01	4.787	3.989	2.197	2.398	1.609	2.197
2014-01-02	4.984	4.060	1.099	2.773	1.609	2.485
2014-01-03	5.050	3.871	1.946	2.639	1.609	2.079
2014-01-04	4.890	3.714	2.773	2.485	1.792	nan

MinMax scaling

Although models I developed are trying to predict feature values basing on historical data of very same variable, for the sake of training I have scaled [pm25] with a use of MinMaxScaler()

In [61]:

```
df_transformed_scaled.tail()
```

Out[61]:

date	pm25
2020-12-28	0.540
2020-12-29	0.501
2020-12-30	0.674
2020-12-31	0.722
2021-01-01	0.806

More details on GitHub

4. Summary of training at least three variations of the Time Series, Survival Analysis, or Deep Learning model you selected. For example, you can use different models or different hyperparameters.

I have train three deep learning models.
One Recurent Neural Network (RNN) and two long-short term memory neural networsk (LSTM).
The data set was split into train_X and train_y series to mimic suppervised learning.

The shape of whole [pm25] series:

In [62]:

```
df_transformed_scaled.shape
```

Out[62]:

```
(2475, 1)
```

The length of test size series for [pm25] was set to 1%:

In [63]:

```
train_X, train_y = train_test_split(df_transformed_scaled["pm25"], test_size=0.01, shuf
```

The length of [pm25] series kept for testing:

In [64]:
```
train_y.shape
```

Out[64]:
```
(25,)
```

RNN

First model - RNN - was trained using only 100 epochs:

```
model = fit_SimpleRNN(train_X, train_y, cell_units=100, epochs=100)
```

Architecture of model was not very deep nor complicated:

```
model = Sequential()
model.add(SimpleRNN(cell_units, input_shape=(train_X.shape[1],1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

The summary of RNN architecture:

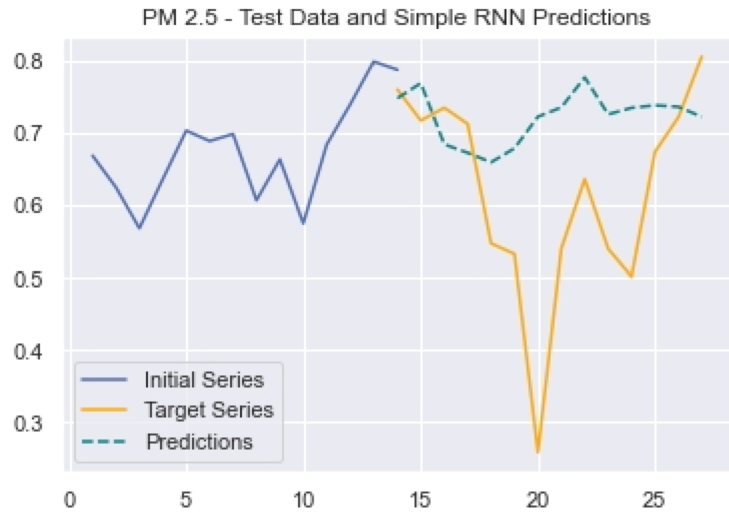
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_1 (SimpleRNN)     (None, 100)               10200
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 10,301
Trainable params: 10,301
Non-trainable params: 0
_________________________________________________________________
```

- The loss based on mean_squared_error metrics got supprisingly low: 0.0044

The visualisation of RNN predictions (scaled data):

In [65]:
```
Image(filename='pm25_rnn_prediction.jpg')
```

Out[65]:



- Surprisingly the model seems to quite successfully predict future events basing on past patterns - that is really promising sign

LSTM - shallow deep learning

Second model - LSTM - was trained using larger number of diffrent weights and 1000 epochs:

```
model = fit_LSTM(train_X, train_y, cell_units=70, epochs=1000)
```

Architecture of the model was not very deep nor complicated either:

```
model = Sequential()
model.add(LSTM(cell_units, input_shape=(train_X.shape[1],1)))
#,return_sequences= True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

The summary of LSTM architecture:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 70)                20160
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 71
=================================================================
Total params: 20,231
Trainable params: 20,231
Non-trainable params: 0
_________________________________________________________________
```

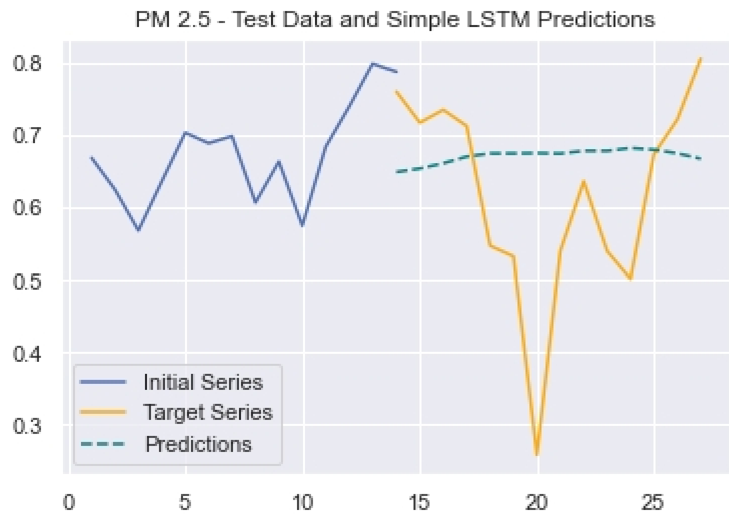
- The loss based on mean_squared_error metrics scored: 0.0161
- Simple RNN models scored better

The visualisation of LSTM predictions (scaled data):

In [66]:

```
Image(filename='pm25_shallow_lstm_prediction.jpg')
```

Out[66]:



- Unfortunatelly the model seems to express too low variance and too high bias

LSTM - deeper model

Third model - LSTM - a deeper model than the previous example was trained using more layers hence using larger amount of weights:

```
model = fit_LSTM(train_X, train_y, cell_units=100, epochs=1000)
```

Architecture of the model:

```
model = Sequential()
model.add(LSTM(cell_units, return_sequences=True, input_shape=
(train_X.shape[1],1))) #,return_sequences= True))
model.add(Dropout(0.2))
model.add(LSTM(140, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(280, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

The summary of LSTM architecture:

Model: "sequential_2"

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 14, 100)	40800
dropout (Dropout)	(None, 14, 100)	0
lstm_2 (LSTM)	(None, 14, 140)	134960

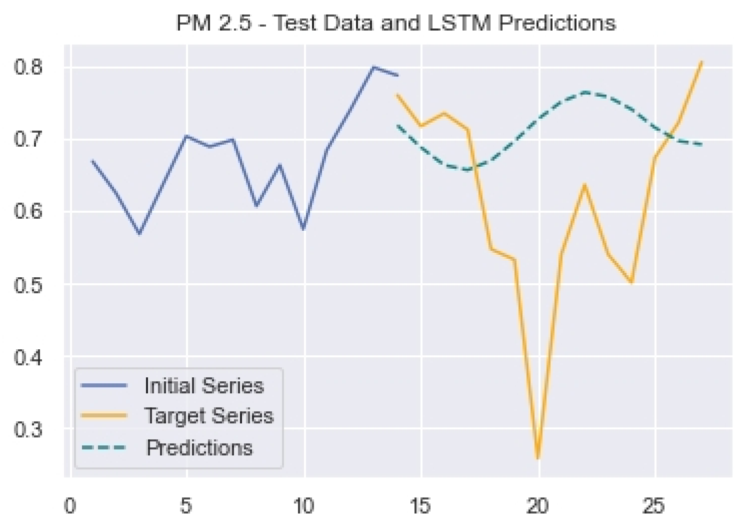
```
dropout_1 (Dropout)          (None, 14, 140)          0
_________________________________________________________________
lstm_3 (LSTM)                (None, 280)              471520
_________________________________________________________________
dropout_2 (Dropout)          (None, 280)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                281
=================================================================
Total params: 647,561
Trainable params: 647,561
Non-trainable params: 0
_________________________________________________________________
```

- After 1000 epochs the model scored mean_squared_error equal to: 0.0133

The visualisation of LSTM predictions (scaled data):

In [70]:
```
Image(filename='pm25_deep_lstm_prediction.jpg')
```

Out[70]:



- This time LSTM model seems to perform better than its previous iteration however it still seems to favour more bias over variance.
- In case of highly volatile data we would like rather to see more variance.

More details on GitHub

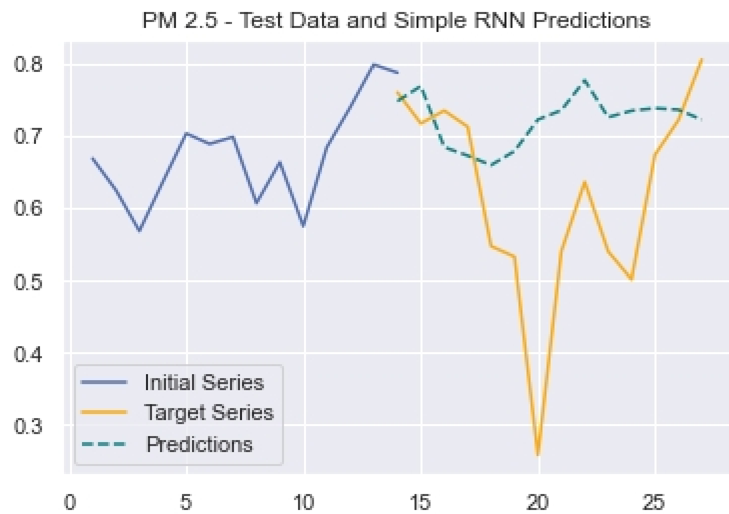
5. A paragraph explaining which of your models you recommend as a final model that best fits your needs in terms of accuracy or explainability.

Since my main goal was to build best possible predictive model I value accuracy more then explainability.
The model I recommend as a final one is the one will lowest mean squared error which is the first one.

Visualized predictions of RNN model with a loss of 0.0044:

In [71]:
```
Image(filename='pm25_rnn_prediction.jpg')
```

Out[71]:



Architecture of RNN model:

```
model = Sequential()
model.add(SimpleRNN(cell_units, input_shape=(train_X.shape[1],1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

6. Summary Key Findings and Insights, which walks your reader through the main findings of your modeling exercise.

Building (any) RNN or LSTM model with a use of Keras seems to be quite an easy task on the other hand getting right predictions of highly volatile data is extremly challenging.
A cost between variance and bias or overfitting and underfitting makes time series forcasting cumbersome task.

The magnitute or errors for predictions of [pm25] feature obviosuly depends on a complexitiy of a model's architecture.
I have build three simple deep learning models: one RNN and two LSTM.
It seems that all models have had performed well on a task of finding the trend of fitted data.
However on a task of finding more complex patterns they do act rather poorly. They express large bias and low variance.
Due to above the variance of predicted [pm25] values is definitely undervalued. What increases the size of observed error.

7. Suggestions for next steps in analyzing this data, which may include suggesting revisiting this model or adding specific data features to achieve a better model.

To be sure if chosen RNN model is any good it would wise to compare it with more commone approaches of time series foracasting, namely:

- ARIMA models

- simple Moving Averages techniques

In other words I would like to use them as another way of benchmarking RNN model.