This is an project related to time series forecasting for predictive maintence. *This project uses dataset that are available on Kaggle.

About Dataset Context This an example data source which can be used for Predictive Maintenance Model Building. It consists of the following data:

Machine conditions and usage: The operating conditions of a machine e.g. data collected from sensors. Failure history: The failure history of a machine or component within the machine. Maintenance history: The repair history of a machine, e.g. error codes, previous maintenance activities or component replacements. Machine features: The features of a machine, e.g. engine size, make and model, location. Details Telemetry Time Series Data (PdM_telemetry.csv): It consists of hourly average of voltage, rotation, pressure, vibration collected from 100 machines for the year 2015.

Error (PdM_errors.csv): These are errors encountered by the machines while in operating condition. Since, these errors don't shut down the machines, these are not considered as failures. The error date and times are rounded to the closest hour since the telemetry data is collected at an hourly rate.

Maintenance (PdM_maint.csv): If a component of a machine is replaced, that is captured as a record in this table. Components are replaced under two situations: 1. During the regular scheduled visit, the technician replaced it (Proactive Maintenance) 2. A component breaks down and then the technician does an unscheduled maintenance to replace the component (Reactive Maintenance). This is considered as a failure and corresponding data is captured under Failures. Maintenance data has both 2014 and 2015 records. This data is rounded to the closest hour since the telemetry data is collected at an hourly rate.

Failures (PdM_failures.csv): Each record represents replacement of a component due to failure. This data is a subset of Maintenance data. This data is rounded to the closest hour since the telemetry data is collected at an hourly rate.

Metadata of Machines (PdM_Machines.csv): Model type & age of the Machines.

download dataset

```
##load data using pandas
import os
import pandas as pd
working_path = "/content/drive/MyDrive/data/"
df_tele = pd.read_csv(working_path+"PdM_telemetry.csv")
df_fail = pd.read_csv(working_path+"PdM_failures.csv")
df_err = pd.read_csv(working_path+"PdM_errors.csv")
df_maint = pd.read_csv(working_path+"PdM_maint.csv")
```

```
#printing the top 5 row
df_fail.head(n=5)
```

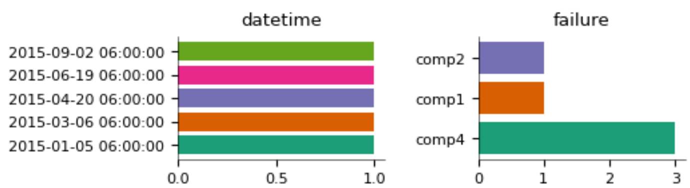
1 to 5 of 5 entries Filter

index	datetime	machineID	failure
0	2015-01-05 06:00:00	1	comp4
1	2015-03-06 06:00:00	1	comp1
2	2015-04-20 06:00:00	1	comp2
3	2015-06-19 06:00:00	1	comp4
4	2015-09-02 06:00:00	1	comp4

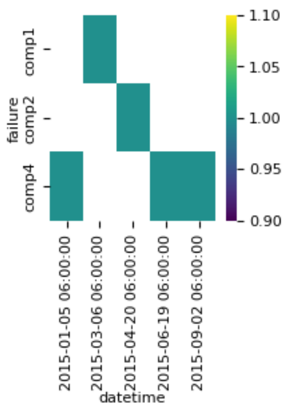
Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

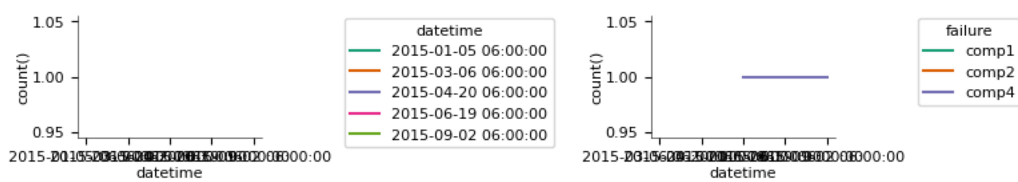
Categorical distributions



2-d categorical distributions



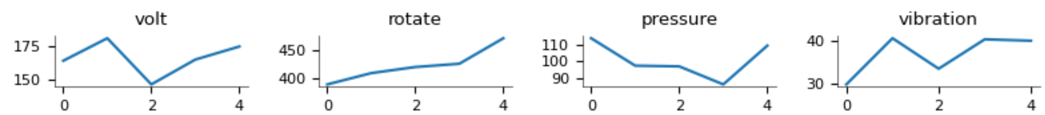
Time series



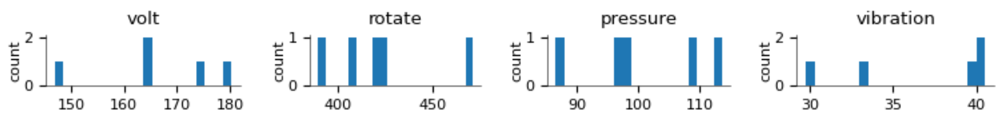

```
df_sel = df_tele.loc[df_tele['machineID']==11].reset_index(drop=True)
df_sel.head(n=5)
```

	datetime	machineID	volt	rotate	pressure	vibration
0	2015-01-01 06:00:00	11	164.039259	389.699577	113.619975	29.775109
1	2015-01-01 07:00:00	11	180.325510	409.788550	97.506203	40.512160
2	2015-01-01 08:00:00	11	146.917119	420.626012	97.087205	33.420937
3	2015-01-01 09:00:00	11	164.895416	426.409611	86.494413	40.263365
4	2015-01-01 10:00:00	11	174.330072	471.774231	109.277816	39.931749

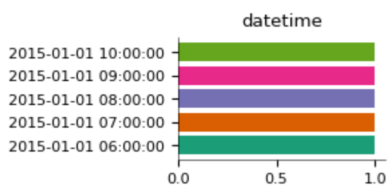
Values



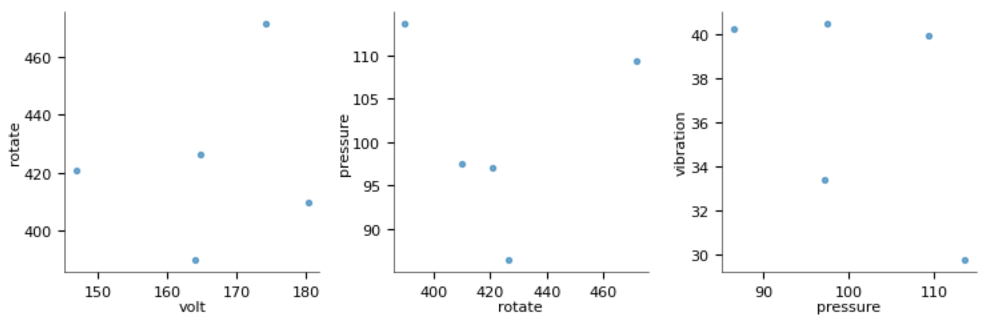
Distributions



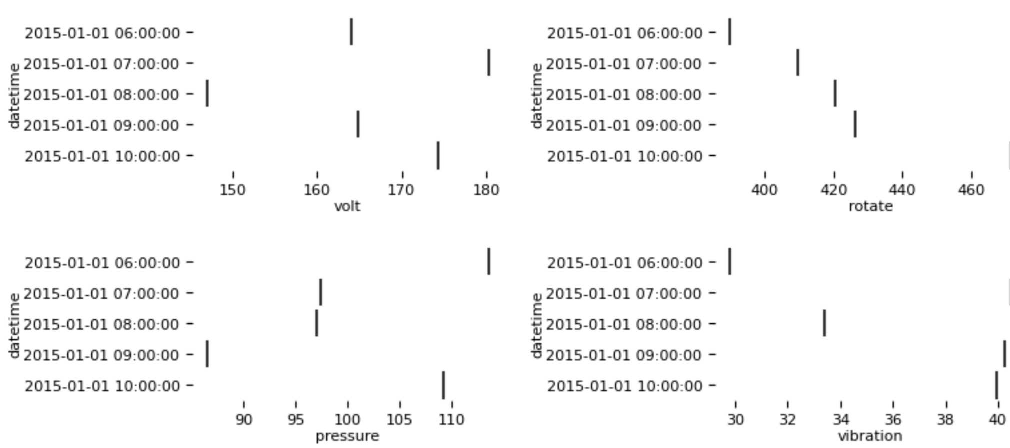
Categorical distributions



2-d distributions



Faceted distributions

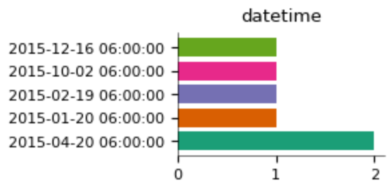


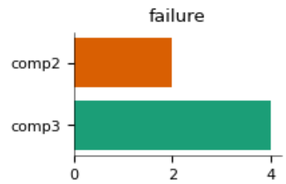
Time series

```
#check failure record of same machine whose ID is 11
sel_fail = df_fail.loc[df_fail['machineID']==11]
pd.DataFrame(sel_fail)
```

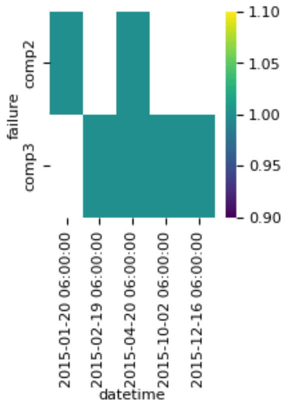
	datetime	machineID	failure
58	2015-01-20 06:00:00	11	comp2
59	2015-02-19 06:00:00	11	comp3
60	2015-04-20 06:00:00	11	comp2
61	2015-04-20 06:00:00	11	comp3
62	2015-10-02 06:00:00	11	comp3
63	2015-12-16 06:00:00	11	comp3

Categorical distributions

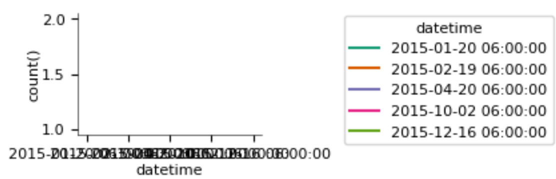


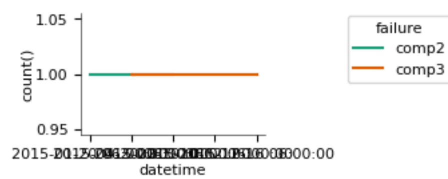


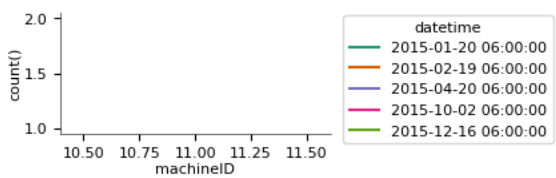
2-d categorical distributions

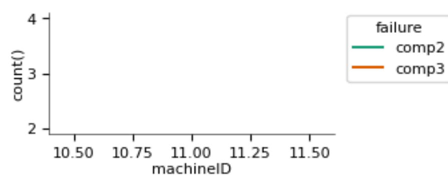


Time series





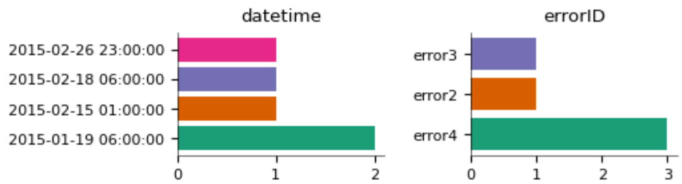




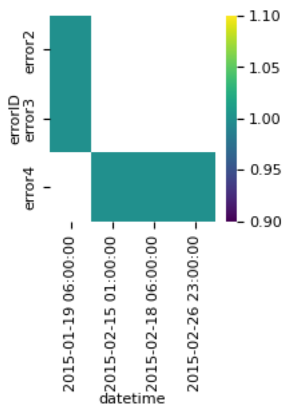
```
#check error record of machine with id is 11
sel_err = df_err.loc[df_err['machineID']==11]
pd.DataFrame(sel_err).head()
```

	datetime	machineID	errorID
360	2015-01-19 06:00:00	11	error2
361	2015-01-19 06:00:00	11	error3
362	2015-02-15 01:00:00	11	error4
363	2015-02-18 06:00:00	11	error4
364	2015-02-26 23:00:00	11	error4

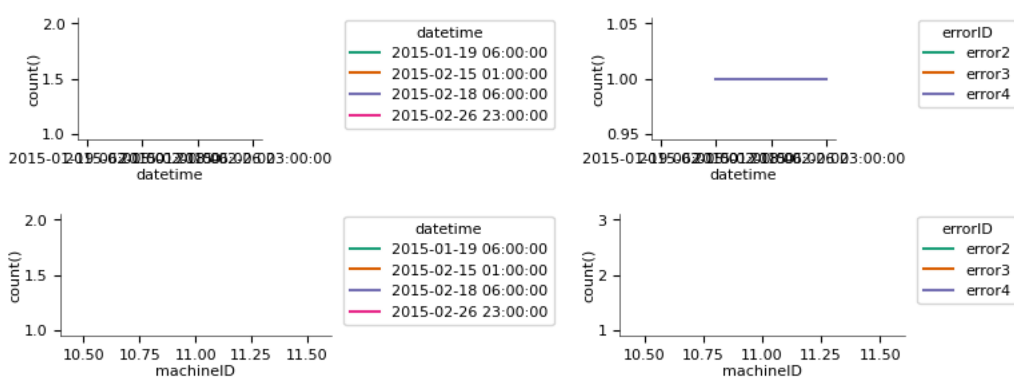
Categorical distributions



2-d categorical distributions



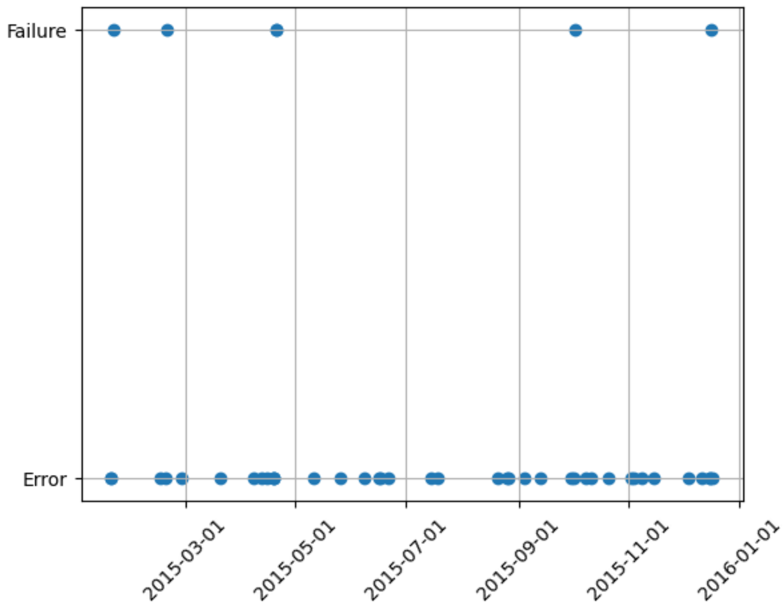
Time series




```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
fig, ax = plt.subplots()
y_category = list()
for iter in range(0,len(sel_fail)):
  y_category.append("Failure")
```

```
for iter in range(0,len(sel_err)):
  y_category.append("Error")

#time stamp
df_timestamp = pd.concat([sel_fail['datetime'],sel_err['datetime']],ignore_index=True,axis=0)
df_plot = pd.DataFrame({"timestamp": df_timestamp,"category": y_category})
df_plot.loc[:,"timestamp"] = pd.to_datetime(df_plot.loc[:,"timestamp"])
df_plot.sort_values(by=['timestamp'],inplace= True,ignore_index= True)
#plotting data with timestamp as x-axis
ax.scatter('timestamp','category',data= df_plot)
yearfmt = mdates.DateFormatter('%Y-%m-%d')
ax.xaxis.set_major_formatter(yearfmt)
ax.tick_params(axis='x',rotation=45)
ax.grid()
```

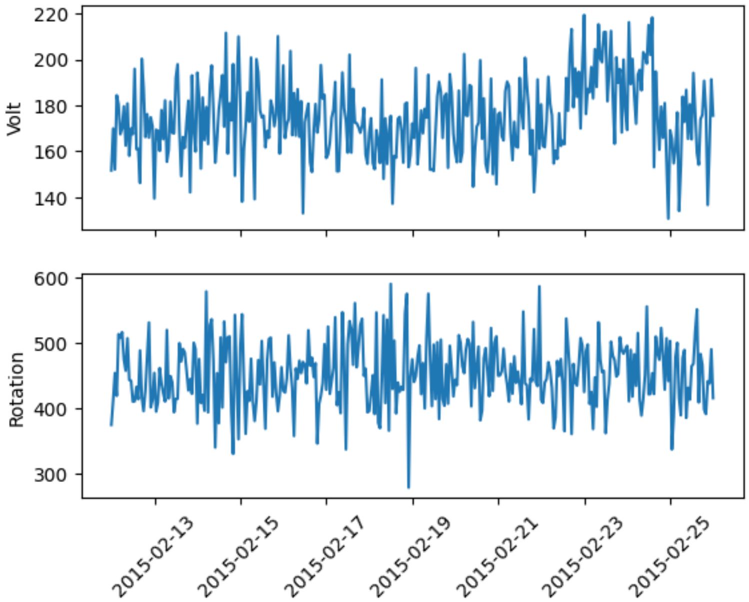
```
<ipython-input-7-bb339a4d0e56>:14: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` w
  df_plot.loc[:,"timestamp"] = pd.to_datetime(df_plot.loc[:,"timestamp"])
```



Feature check

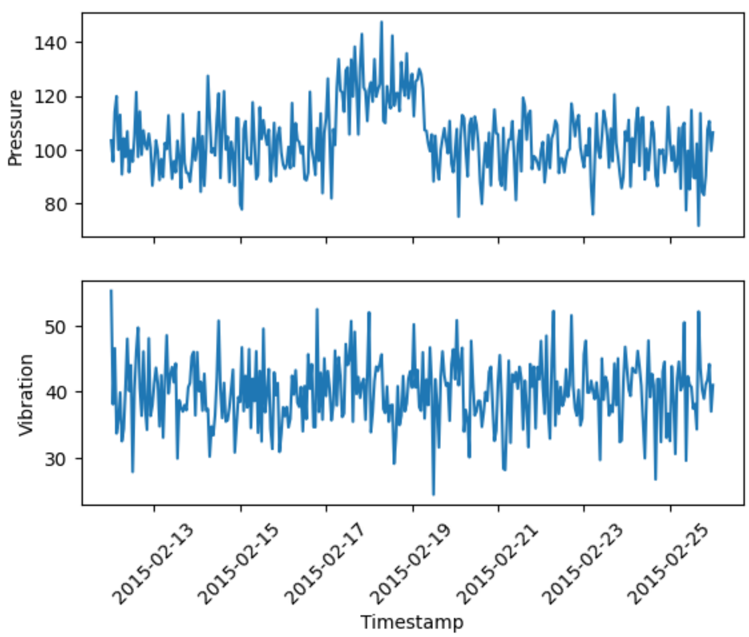
```
df_sel.loc[:,'datetime'] = pd.to_datetime(df_sel.loc[:,"datetime"])
#select the date to check from failure records
st = df_sel.loc[df_sel['datetime']=='2015-02-19'].index.values[0]
#Then filtre the data
#the error occurs
select = df_sel.loc[st-7*24:st + 7*24,:]
#plot volt antd rotation feature
fig, ax = plt.subplots(nrows=2, sharex=True)
ax[0].plot('datetime','volt',data=select)
ax[0].set_ylabel("Volt")

ax[1].plot('datetime','rotate',data=select)
ax[1].tick_params(axis='x',rotation=45)
ax[1].set_xlabel("Timestamp")
ax[1].set_ylabel("Rotation")
```

```
<ipython-input-8-b4c6265e093d>:1: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` wi
  df_sel.loc[:,'datetime'] = pd.to_datetime(df_sel.loc[:,"datetime"])
Text(0, 0.5, 'Rotation')
```



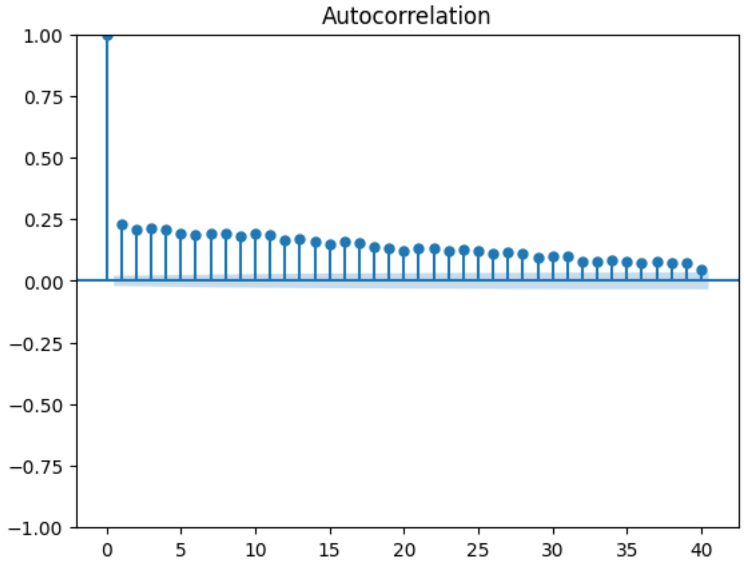

```
#plot pressure and vibration feature
fig, ax = plt.subplots(nrows=2, sharex=True)
ax[0].plot('datetime','pressure',data=select)
ax[0].set_ylabel("Pressure")

ax[1].plot('datetime','vibration',data=select)
ax[1].tick_params(axis="x",rotation=45)
ax[1].set_xlabel("Timestamp")
ax[1].set_ylabel("Vibration")
```

```
Text(0, 0.5, 'Vibration')
```



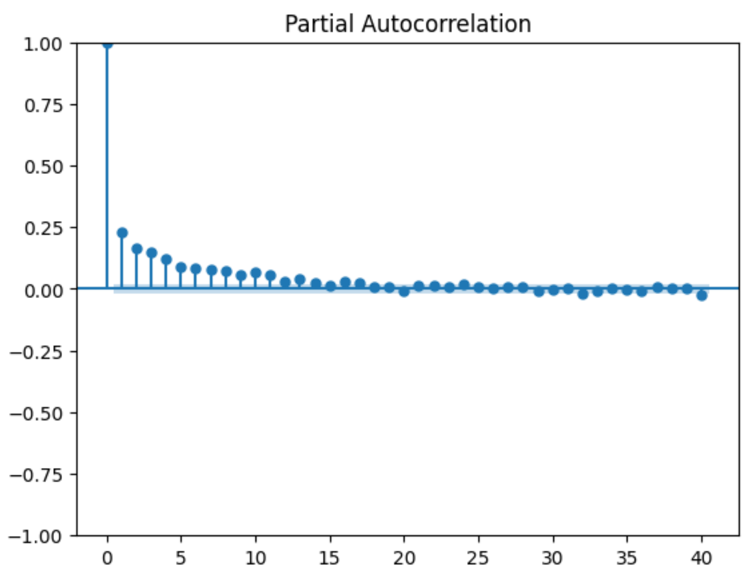
Check autocorrelation and partial autocorrelation

```
#Import plotting function
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
#Autocorrelation plot
plot_acf(df_sel['pressure'], lags=40)
plt.show()
```




```
#Partial autocorrelation plot
plot_pacf(df_sel['pressure'],lags=40)
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/graphics/tsaplots.py:348: FutureWarning: The defaul
  warnings.warn(
```



Model Selection

prepare data input and output

we will use LSTM model, one of the famous prediction model in time-series forecasting task. To use it, first we need to provide input and output data in the correct format.

For our experiment, we will use training data of 1 month containing 2015-02-19 period where failure happened to predict another failure which occurs at 2015-04-20 according to the failure record. The feature used will be the pressure reading and timestamp (one-hot encoded).

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Select the date to check from failure records
st_train = df_sel.loc[df_sel['datetime'] == "2015-02-19"].index.values[0]

# Then, filter the data to include approximately one month window
start_period = st_train - 14*24
end_period = st_train + 14*24

def create_feature(start, end):
  # create features from the selected machine
  pressure = df_sel.loc[start: end, 'pressure']
  timestamp = pd.to_datetime(df_sel.loc[start: end, 'datetime'])
  timestamp_hour = timestamp.map(lambda x: x.hour)
  timestamp_dow = timestamp.map(lambda x: x.dayofweek)

  # apply one-hot encode for timestamp data
  timestamp_hour_onehot = pd.get_dummies(timestamp_hour).to_numpy()

  # apply min-max scaler to numerical data
  scaler = MinMaxScaler()
  pressure = scaler.fit_transform(np.array(pressure).reshape(-1,1))

  # combine features into one
  feature = np.concatenate([pressure, timestamp_hour_onehot], axis=1)

  X = feature[:-1]
  y = np.array(feature[5:,0]).reshape(-1,1)

  return X, y, scaler

X, y, pres_scaler = create_feature(start_period, end_period)
```

```
def shape_sequence(arr, step, start):
    out = list()
    for i in range(start, arr.shape[0]):
        low_lim = i
        up_lim = low_lim + step
        out.append(arr[low_lim: up_lim])

        if up_lim == arr.shape[0]:
          # print(i)
          break

    out_seq = np.array(out)
    return out_seq

# Shape the sequence according to the length specified
X_seq = shape_sequence(X, 5, 0)
y_seq = shape_sequence(y, 1, 0)

# Separate the input and output for train and validation
X_train, X_val, y_train, y_val = train_test_split(X_seq, y_seq, test_size=0.2, shuffle=False)

print("Training data shape = ", X_train.shape)
print("Validation data shape = ", X_val.shape)
```

```
Training data shape =  (534, 5, 25)
Validation data shape =  (134, 5, 25)
```

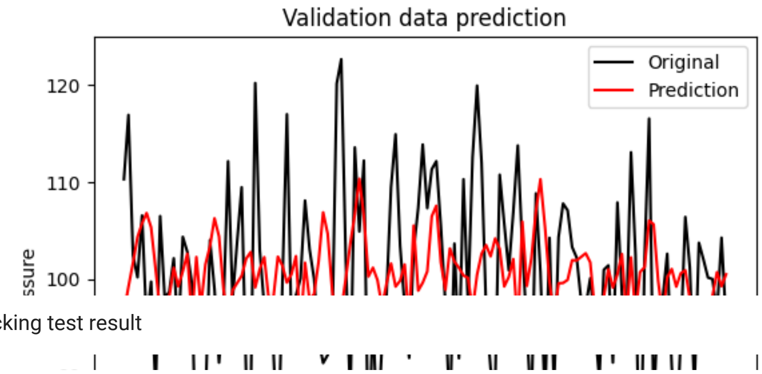
Creating a model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow.keras.losses as loss


def create_model(X_train, y_train):
  shape = X_train.shape[1]
  feat_length = X_train.shape[2]

  model = Sequential()
  model.add(LSTM(shape, activation='tanh', input_shape=(shape, feat_length), return_sequences=True))
  model.add(LSTM(shape, activation='tanh', input_shape=(shape, feat_length), return_sequences=False))
  model.add(Dense(shape, activation='relu'))
  model.add(Dense(1, activation='linear'))
  model.compile(optimizer=Adam(lr=0.035),
                loss=loss.mean_squared_error)
  model.fit(X_train, y_train, verbose=1, epochs=500)

  return model

model = create_model(X_train, y_train)
```

```
Epoch 479/500
17/17 [==============================] - 0s 9ms/step - loss: 0.0139
Epoch 480/500
17/17 [==============================] - 0s 10ms/step - loss: 0.0139
Epoch 481/500
17/17 [==============================] - 0s 9ms/step - loss: 0.0139
Epoch 482/500
17/17 [==============================] - 0s 10ms/step - loss: 0.0138
Epoch 483/500
17/17 [==============================] - 0s 10ms/step - loss: 0.0138
Epoch 484/500
17/17 [==============================] - 0s 9ms/step - loss: 0.0138
Epoch 485/500
17/17 [==============================] - 0s 9ms/step - loss: 0.0138
Epoch 486/500
17/17 [==============================] - 0s 10ms/step - loss: 0.0138
Epoch 487/500
17/17 [==============================] - 0s 10ms/step - loss: 0.0139
Epoch 488/500
17/17 [==============================] - 0s 10ms/step - loss: 0.0139
Epoch 489/500
17/17 [==============================] - 0s 9ms/step - loss: 0.0138
Epoch 490/500
17/17 [==============================] - 0s 8ms/step - loss: 0.0138
Epoch 491/500
17/17 [==============================] - 0s 9ms/step - loss: 0.0138
Epoch 492/500
17/17 [==============================] - 0s 11ms/step - loss: 0.0138
Epoch 493/500
17/17 [==============================] - 0s 10ms/step - loss: 0.0138
Epoch 494/500
17/17 [==============================] - 0s 8ms/step - loss: 0.0138
Epoch 495/500
17/17 [==============================] - 0s 13ms/step - loss: 0.0138
Epoch 496/500
17/17 [==============================] - 0s 12ms/step - loss: 0.0139
Epoch 497/500
17/17 [==============================] - 0s 13ms/step - loss: 0.0139
Epoch 498/500
17/17 [==============================] - 0s 13ms/step - loss: 0.0139
Epoch 499/500
17/17 [==============================] - 0s 14ms/step - loss: 0.0139
Epoch 500/500
17/17 [==============================] - 0s 16ms/step - loss: 0.0138
```

```
# Predict validation data using the trained model
y_pred = model.predict(X_val)
mse = MeanSquaredError()
val_err = mse(y_val.reshape(-1,1), y_pred)
print("Validation error = ", val_err.numpy())
# Return the value using inverse transform to allow better observation
plt.plot(pres_scaler.inverse_transform(y_val.reshape(-1,1)), 'k', label='Original')
plt.plot(pres_scaler.inverse_transform(y_pred.reshape(-1,1)), 'r', label='Prediction')
plt.ylabel("Pressure")
plt.xlabel("Datapoint")
plt.title("Validation data prediction")
plt.legend()
plt.show()
```

```
5/5 [==============================] - 1s 3ms/step
Validation error =  0.014821364
```



checking test result

```
# Select the date where another failure occurred
st_test = df_sel.loc[df_sel['datetime'] == "2015-04-20"].index.values[0]

# Then, filter the data to include approximately two-weeks window
start_period_test = st_test - 7*24
end_period_test = st_test + 7*24
X_test, y_test, test_scaler = create_feature(start_period_test, end_period_test)

# Shape the sequence
X_test_seq = shape_sequence(X_test, 5, 0)
y_test_seq = shape_sequence(y_test, 1, 0)

# Predict the testing data
y_pred_test = model.predict(X_test_seq)
test_err = mse(y_test_seq.reshape(-1,1), y_pred_test)
print("Testing error = ", test_err.numpy())

# Select first 200 datapoints to allow for better plotting
# Return the value using inverse transform to allow better observation
plt.plot(test_scaler.inverse_transform(y_pred_test[:200].reshape(-1, 1)), 'r', label='Prediction')
plt.plot(test_scaler.inverse_transform(y_test_seq[:200].reshape(-1, 1)), 'k', label='Original')
plt.ylabel("Pressure")
plt.xlabel("Datapoints")
plt.legend()
plt.show()
```