Claude Shyaka

ID: 801326243

Classical Machine Learning Approach

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
```

```
seed = 42

# Load energy data
df_dataset = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/data/power_dataset.csv", del
                         index_col="time")
df_dataset.head()
```

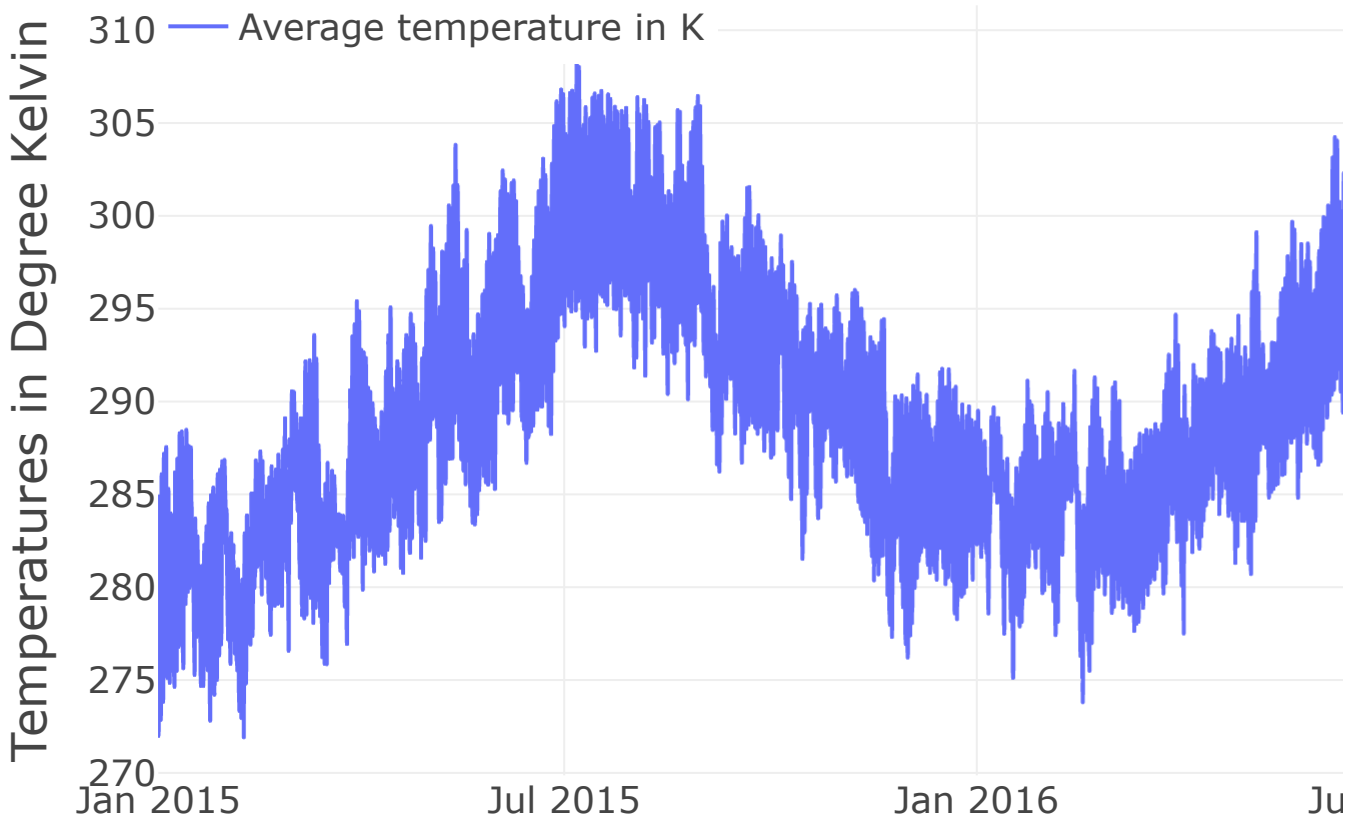
time	Average temperature in K	Average pressure in hPa	Average humidity in %	Average rain in last 1 hour in mm	Average rain in last 3 hours in mm	Average snow in last 3 hours in mm	total load actual
2015-01-01 00:00:00+01:00	272.491463	1016.4	82.4	82.4	82.4	82.4	25385.0
2015-01-01 01:00:00+01:00	272.512700	1016.2	82.4	82.4	82.4	82.4	24382.0

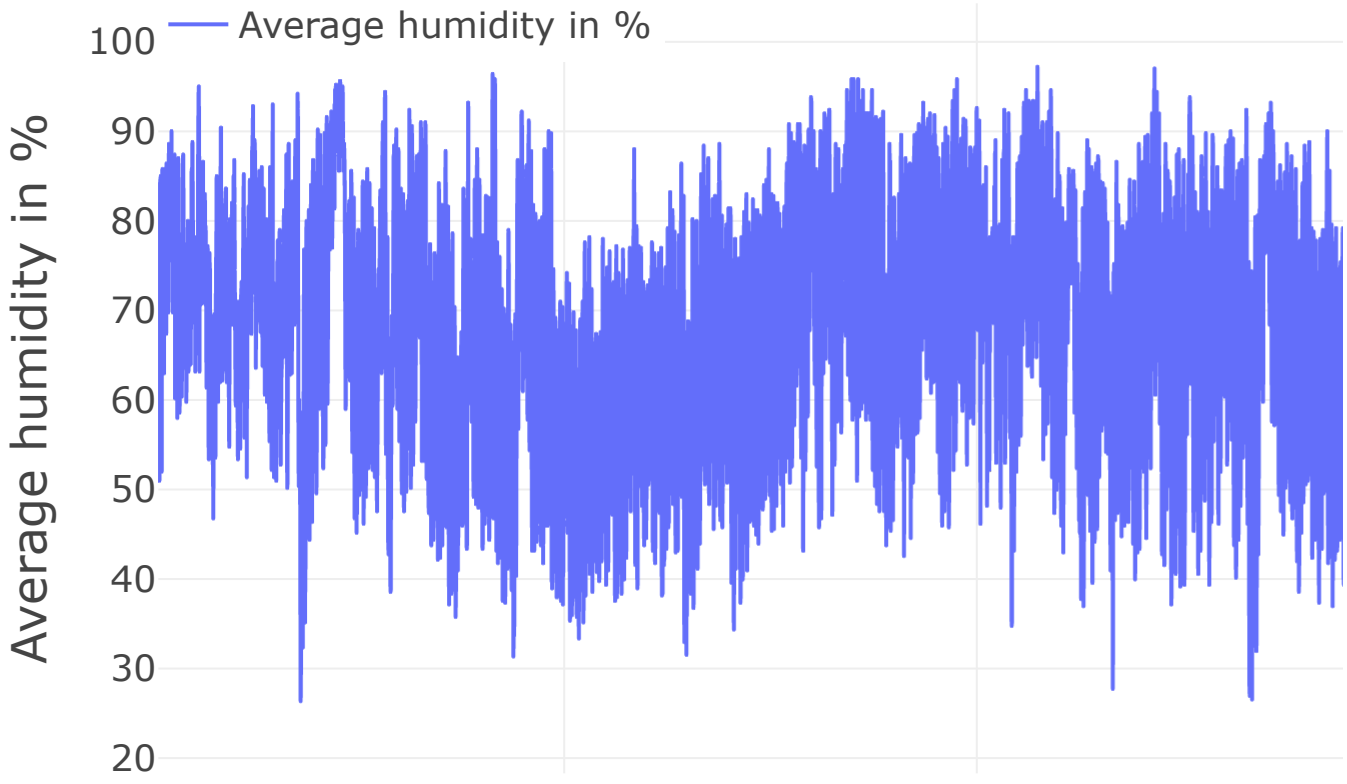
```
# Obtain values from pd frame
data = df_dataset.values
batch_svr = data[:5000, :]
data.shape
```

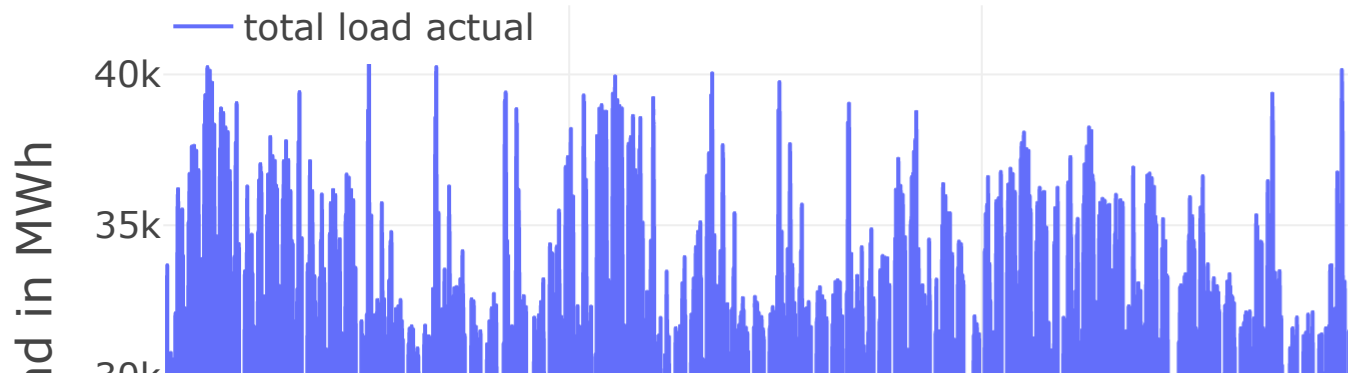
```
(38568, 7)
```

```
import plotly.express as px
import plotly.graph_objects as go
```

```
import plotly.io as pio
pio.templates.default = "plotly_white"

plot_template = dict(
    layout=go.Layout({
        "font_size": 18,
        "xaxis_title_font_size": 24,
        "yaxis_title_font_size": 24})
)

cols_to_plot = ["Average temperature in K"]
fig = px.line(df_dataset[cols_to_plot], labels=dict(
    time="Dates", value="Temperatures in Degree Kelvin", variable="Data"
))
fig.update_layout(
  template=plot_template, legend=dict(orientation='h', y=1.02, title_text="")
)
fig.show()

cols_to_plot = ["Average humidity in %"]
fig = px.line(df_dataset[cols_to_plot], labels=dict(
    time="Dates", value="Average humidity in %", variable="Data"
))
fig.update_layout(
  template=plot_template, legend=dict(orientation='h', y=1.02, title_text="")
)
fig.show()

cols_to_plot = ["total load actual"]
fig = px.line(df_dataset[cols_to_plot], labels=dict(
    time="Dates", value="Electric Load in MWh", variable="Data"
))
fig.update_layout(
  template=plot_template, legend=dict(orientation='h', y=1.02, title_text="")
)
fig.show()
```


Average temperature in K
Temperatures in Degree Kelvin
310
305
300
295
290
285
280
275
270
Jan 2015
Jul 2015
Jan 2016
Ju


Average humidity in %
Average humidity in %
100
90
80
70
60
50
40
30
20




```
import time
# Train LR to compare with the SVR results
def TrainLRModels(data):

    # Split the data in train and test set
    data_train, data_test = train_test_split(data, train_size=0.8, test_size=0.2, random_stat

    # Apply the MinMaxScaler to the Datasets
    sc = MinMaxScaler()
    data_train_scaled = sc.fit_transform(data_train)
    data_test_scaled = sc.transform(data_test)

    n_of_feature = data.shape[1]-1

    # Get the training feature and labels
    X_train, Y_train = data_train_scaled[:, :n_of_feature], data_train_scaled[:, n_of_feature

    # Get the testing features and labels
    X_test, Y_test = data_test_scaled[:, :n_of_feature], data_test_scaled[:, n_of_feature]

    # Ridge regression model
    model = GridSearchCV(
        Ridge(),
        param_grid={"alpha": [1e0, 0.1, 1e-2, 1e-3]},
    )
    t0 = time.time()
    model.fit(X_train, Y_train)
    rr_fit = time.time() - t0
    estimator = model.best_estimator_
    Y_pred = estimator.predict(X_test)
    print(f"Best RR with params: {model.best_params_} and R2 score: {model.best_score_:.3f}")
    print("Mean Squared Error from RR Model:", mean_squared_error(Y_test, Y_pred))
    print("RR complexity and bandwidth selected and model fitted in %.3f s" % rr_fit)
```

```
import time
# Train SVR models without using PCA
def TrainSVRModel(data):

    # Split the data in train and test set
    data_train, data_test = train_test_split(data, train_size=0.8, test_size=0.2, random_stat

    # Apply the MinMaxScaler to the Datasets
    sc = MinMaxScaler()
    data_train_scaled = sc.fit_transform(data_train)
    data_test_scaled = sc.transform(data_test)

    n_of_feature = data.shape[1]-1

    # Get the training feature and labels
    X_train, Y_train = data_train_scaled[:, :n_of_feature], data_train_scaled[:, n_of_feature

    # Get the testing features and labels
    X_test, Y_test = data_test_scaled[:, :n_of_feature], data_test_scaled[:, n_of_feature]

    # Support Vector Regression model
    model = GridSearchCV(
        SVR(kernel="rbf", gamma=0.1),
        param_grid={"C": [1e0, 1e1, 1e2, 1e3], "gamma": np.logspace(-2, 2, 5)},
    )
    t0 = time.time()
    model.fit(X_train, Y_train)
    svr_fit = time.time() - t0
    estimator = model.best_estimator_
    Y_pred = estimator.predict(X_test)
    print(f"Best SVR with params: {model.best_params_} and R2 score: {model.best_score_:.3f}"
    print("Mean Squared Error from SVR Model:", mean_squared_error(Y_test, Y_pred))
    print("SVR complexity and bandwidth selected and model fitted in %.3f s" % svr_fit)


# Train a linear regression model.
TrainLRModels(data)
```

```
Best RR with params: {'alpha': 1.0} and R2 score: 0.120
Mean Squared Error from RR Model: 0.035647984691320446
RR complexity and bandwidth selected and model fitted in 0.409 s
```

```
# Train a Support Vector Regression model
TrainSVRModel(data)
```

```
Best SVR with params: {'C': 1000.0, 'gamma': 10.0} and R2 score: 0.163
Mean Squared Error from SVR Model: 0.03404171727357748
SVR complexity and bandwidth selected and model fitted in 16397.253 s
```

✓ 2s completed at 3:14 PM