assignment_1

Oskar Våle

2024-09-06

```
knitr::opts_chunk$set(echo = TRUE)
library(reticulate)
use_python("/usr/bin/python3")
```

Power consumption dataset

The dataset I chose is the "Individual Household Electric Power Consumption" dataset from the UCI machine learning repository. The dataset contains measurements of electric power consumption in a single household over a period of almost 4 years. The data includes variables such as global active power, global reactive power, voltage, and global intensity, as well as sub-metering values corresponding to different household areas like the kitchen, laundry room, and other appliances. Additionally, the data includes timestamps that have been used to extract the time of day and the month in which the data was recorded.

This is the variable information from UCI's page: 1.date: Date in format dd/mm/yyyy 2.time: time in format hh:mm:ss 3.global_active_power: household global minute-averaged active power (in kilowatt) 4.global_reactive_power: household global minute-averaged reactive power (in kilowatt) 5.voltage: minute-averaged voltage (in volt) 6.global_intensity: household global minute-averaged current intensity (in ampere) 7.sub_metering_1: energy sub-metering No. 1 (in watt-hour of active energy). It corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not electric but gas powered). 8.sub_metering_2: energy sub-metering No. 2 (in watt-hour of active energy). It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light. 9.sub_metering_3: energy sub-metering No. 3 (in watt-hour of active energy). It corresponds to an electric water-heater and an air-conditioner.

I decided to make one-hot encoded columns for whether a given datapoint is recorded in the morning, afternoon, evening or night instead of training on the 'Time' column. My hunch is that the model will perform better this way. I also made a new column 'Month', and let this one just be integers 1 - 12. Due to temperature, holidays and other factors, energy consumption is likely to vary from month to month. There is no point in training on days of the month though, as, except for weekends and holidays, there is no reason one day should have more consumption than the next.

```
import pandas as pd
from ucimlrepo import fetch_ucirepo

# Fetch dataset
individual_household_electric_power_consumption = fetch_ucirepo(id=235)
```

```
## /home/oskarva/.local/lib/python3.10/site-packages/ucimlrepo/fetch.py:97: DtypeWarning: Columns (2,3,4
##   df = pd.read_csv(data_url)
```

```
# Data (as pandas dataframes)
X = individual_household_electric_power_consumption.data.features
y = individual_household_electric_power_consumption.data.targets
```

```
# Variable information
print(individual_household_electric_power_consumption.variables)
```

```
##                     name     role         type  ... description units missing_values
## 0                   Date  Feature         Date  ...        None  None             no
## 1                   Time  Feature  Categorical  ...        None  None             no
## 2    Global_active_power  Feature   Continuous  ...        None  None             no
## 3  Global_reactive_power  Feature   Continuous  ...        None  None             no
## 4                Voltage  Feature   Continuous  ...        None  None             no
## 5       Global_intensity  Feature   Continuous  ...        None  None             no
## 6         Sub_metering_1  Feature   Continuous  ...        None  None             no
## 7         Sub_metering_2  Feature   Continuous  ...        None  None             no
## 8         Sub_metering_3  Feature   Continuous  ...        None  None             no
##
## [9 rows x 7 columns]
```

```
# Combine features and targets for easier manipulation
data = pd.concat([X, y], axis=1)
#NOTE: The above code is taken from UCI's "import in python" function. This

# Convert 'Date' and 'Time' into a single datetime column
data['Datetime'] = pd.to_datetime(data['Date'] + ' ' + data['Time'], format='%d/%m/%Y %H:%M:%S')

# Function to categorize time of day
def categorize_time_of_day(hour):
    if 6 <= hour < 12:
        return 'Morning'
    elif 12 <= hour < 18:
        return 'Afternoon'
    elif 18 <= hour < 24:
        return 'Evening'
    else:
        return 'Night'

# Apply the function to create a new column 'Time_of_Day'
data['Time_of_Day'] = data['Datetime'].dt.hour.apply(categorize_time_of_day)

# Extract the month and create a new column 'Month'
data['Month'] = data['Datetime'].dt.month

# Drop the original Date, Time, and Datetime columns if not needed
data = data.drop(columns=['Date', 'Time', 'Datetime'])

# As the output shows, some of the data is still objects. We therefore need to
# convert it to numerical values.
print(data.dtypes)
```

```
## Global_active_power       object
## Global_reactive_power     object
## Voltage                   object
## Global_intensity          object
## Sub_metering_1            object
## Sub_metering_2            object
## Sub_metering_3           float64
## Time_of_Day               object
```

```
## Month                          int64
## dtype: object
```

```
cols_to_convert = ['Global_active_power', 'Global_reactive_power', 'Voltage',
                   'Global_intensity', 'Sub_metering_1', 'Sub_metering_2']

for col in cols_to_convert:
    data[col] = pd.to_numeric(data[col], errors='coerce')

#Check for NaN values
print(data.isna().sum())
```

```
## Global_active_power      25979
## Global_reactive_power    25979
## Voltage                  25979
## Global_intensity         25979
## Sub_metering_1           25979
## Sub_metering_2           25979
## Sub_metering_3           25979
## Time_of_Day                  0
## Month                        0
## dtype: int64
```

```
# As the output shows, some of the data is still objects. We therefore need to
# convert it to numerical values.
print(data.dtypes)
```

```
## Global_active_power      float64
## Global_reactive_power    float64
## Voltage                  float64
## Global_intensity         float64
## Sub_metering_1           float64
## Sub_metering_2           float64
## Sub_metering_3           float64
## Time_of_Day               object
## Month                      int64
## dtype: object
```

```
# Drop rows that contain NaN values
data.dropna(axis=0, inplace=True)


# Perform one-hot encoding on the 'Time_of_Day' column, as mentionet above.. This
# is due to the fact that linear regression can only handle numerical values.
# I could also have had a single Time_of_Day column with morning=1,
# afternoon = 2, evening = 3, night = 4, but this implies that morning is far from
# night (4-1=3), which could negatively impact the model.
data = pd.get_dummies(data, columns=['Time_of_Day'], drop_first=False)
```

Including Plots

```
summary_stats = data.describe(include="all")

# Generating summary statistics for categorical variables (Time_of_Day after one-hot encoding)
categorical_summary = data[['Time_of_Day_Morning', 'Time_of_Day_Afternoon', 'Time_of_Day_Evening', 'Tim
```

```python
# Combine the summaries
pd.set_option('display.max_columns', None)
summary_stats_combined = summary_stats.append(categorical_summary)
```

```
## <string>:1: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a
```

```python
summary_stats_combined.loc['count'] = len(data)

# Display the summary statistics
print(summary_stats_combined)
```

```
##        Global_active_power  Global_reactive_power       Voltage  \
## count         2.049280e+06           2.049280e+06  2.049280e+06
## mean          1.091615e+00           1.237145e-01  2.408399e+02
## std           1.057294e+00           1.127220e-01  3.239987e+00
## min           7.600000e-02           0.000000e+00  2.232000e+02
## 25%           3.080000e-01           4.800000e-02  2.389900e+02
## 50%           6.020000e-01           1.000000e-01  2.410100e+02
## 75%           1.528000e+00           1.940000e-01  2.428900e+02
## max           1.112200e+01           1.390000e+00  2.541500e+02
## sum                    NaN                    NaN           NaN
##
##        Global_intensity  Sub_metering_1  Sub_metering_2  Sub_metering_3  \
## count      2.049280e+06    2.049280e+06    2.049280e+06    2.049280e+06
## mean       4.627759e+00    1.121923e+00    1.298520e+00    6.458447e+00
## std        4.444396e+00    6.153031e+00    5.822026e+00    8.437154e+00
## min        2.000000e-01    0.000000e+00    0.000000e+00    0.000000e+00
## 25%        1.400000e+00    0.000000e+00    0.000000e+00    0.000000e+00
## 50%        2.600000e+00    0.000000e+00    0.000000e+00    1.000000e+00
## 75%        6.400000e+00    0.000000e+00    1.000000e+00    1.700000e+01
## max        4.840000e+01    8.800000e+01    8.000000e+01    3.100000e+01
## sum                 NaN             NaN             NaN             NaN
##
##               Month  Time_of_Day_Afternoon  Time_of_Day_Evening  \
## count  2.049280e+06           2.049280e+06         2.049280e+06
## mean   6.454433e+00           2.498970e-01         2.504031e-01
## std    3.423209e+00           4.329533e-01         4.332453e-01
## min    1.000000e+00           0.000000e+00         0.000000e+00
## 25%    3.000000e+00           0.000000e+00         0.000000e+00
## 50%    6.000000e+00           0.000000e+00         0.000000e+00
## 75%    9.000000e+00           0.000000e+00         1.000000e+00
## max    1.200000e+01           1.000000e+00         1.000000e+00
## sum             NaN           5.121090e+05         5.131460e+05
##
##        Time_of_Day_Morning  Time_of_Day_Night
## count         2.049280e+06       2.049280e+06
## mean          2.496731e-01       2.500268e-01
## std           4.328239e-01       4.330283e-01
## min           0.000000e+00       0.000000e+00
## 25%           0.000000e+00       0.000000e+00
## 50%           0.000000e+00       0.000000e+00
## 75%           0.000000e+00       1.000000e+00
## max           1.000000e+00       1.000000e+00
## sum           5.116500e+05       5.123750e+05
```