Week7_DataCheck

November 17, 2020

1 Check The data

Check if the Numpy data is the same as the excel data

[2]:
```
#Import modules:
import numpy as np
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
import glob
import seaborn as sns

from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from statsmodels.tsa.arima_model import ARMA
```

[3]:
```
#define all variables:
    #Paths
loadpath_np = "/home/18005152/notebooks/zero/DATA/"
paths_np = glob.glob(loadpath_np+"*.npy")
```

[4]:
```
#functions
def nParse3(n):
    """output a string that is 3 decimals long. Levy en Jefry kunnen uitleg␣
 ↪geven"""
    number = str(n)
    if len(number) == 1:
        number = "00" + number
    elif len(number) == 2:
        number = "0" + number
    elif len(number) == 3:
        number = number
    return str(number)

def check_eq_w_range(Numpy_Sum,Excel_Sum,abs_diff):
    """Check if a value is equal within a certain range."""
    if abs(Numpy_Sum-Excel_Sum) < abs_diff:
```

```
        return True
    else:
        return False

def open_file_append_sentence(msg):
    """open the file, appand the sentence, and then close the file."""
    with open("DataCheck_info_NEW.txt",'a') as f:
        f.write(str(msg))
        f.close()
```

[5]:
```
sheet_dict = {'solar': [3, 'Out'],
              'smartMeter': [7, 'Out'],
              'energyWtwReg': [2, 'In'],
              'energyHeatpump': [2, 'In']}

houses = list()
def Jef_Func():
    for i in range(1,2):
        df_delta = pd.DataFrame()
        for sheetname, j in sheet_dict.items():
            df = pd.DataFrame(np.load(loadpath_np + sheetname + '_' +␣
 ↪nParse3(i) + '.npy'))
            df = df.set_index(pd.DatetimeIndex(pd.to_datetime(df[0],unit='s').
 ↪values))
            df = df.resample('5min').sum()

            if sheetname == 'smartMeter':
                col = df[6].shift(-1) - df[6]
                col = col.shift(1)
                df_delta[str(sheetname)+'In'+'_delta'] = col

            col = df[j[0]].shift(-1) - df[j[0]]
            col = col.shift(1)
            df_delta[str(sheetname)+j[1]+'_delta'] = col

        houses.append(df_delta)
    return houses
```

[6]:
```
houses = Jef_Func()
df = houses[0]['solarOut_delta']
df = df.resample('D').sum()
df = df.dropna()
df = pd.DataFrame(df)

#aanname: alle negatieve waarden eruit.
df = df[(df["solarOut_delta"]>=-20) & (df["solarOut_delta"]<=100)]
```

```
%matplotlib notebook
ax = df.plot(kind='density',xlim=[-20,80],grid=True)
ax.set_xlabel("Daily solar Production [kWh]")
plt.show()
print(df.describe())
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
       solarOut_delta
count      356.000000
mean        22.722079
std         16.450727
min          0.000000
25%          8.200000
50%         19.180000
75%         37.082500
max         58.640000
```

this is where the program will be ran.

[7]:
```
#prepare the data:
houses = Jef_Func()
df = houses[0]['solarOut_delta']
df = df.resample('60min').sum()

#set variables:
sdate = "2019-10-19"
edate = "2019-10-25"
start = 0
stop = 1000
factor = 10
dick = {}

#reshape the data:
y = (df[sdate:edate]-(df[sdate:edate].values)[0]).values.reshape(-1,1).tolist()
y = [i[0] for i in y]
x = ((df[sdate:edate].index).values.reshape(-1,1)).tolist()
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=2/7,␣
 ↪random_state=0,shuffle=False)

for qC in tqdm(range(start+1,stop*factor)):
    svr = SVR(C=qC/factor)
    svr.fit(X_train,y_train)
    r2 = svr.score(X_test,y_test)

    dick[str(qC)] = r2
```

```
 12%|          | 1204/9999 [00:04<00:29, 293.25it/s]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-7-026be2c9b0c7> in <module>
     20 for qC in tqdm(range(start+1,stop*factor)):
     21     svr = SVR(C=qC/factor)
---> 22     svr.fit(X_train,y_train)
     23     r2 = svr.score(X_test,y_test)
     24

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/sklearn/svm/_base.py in␣
 ↪fit(self, X, y, sample_weight)
    215
    216         seed = rnd.randint(np.iinfo('i').max)
--> 217         fit(X, y, sample_weight, solver_type, kernel, random_seed=seed)
    218         # see comment on the other call to np.iinfo in this file
    219

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/sklearn/svm/_base.py in␣
 ↪_dense_fit(self, X, y, sample_weight, solver_type, kernel, random_seed)
    274                 cache_size=self.cache_size, coef0=self.coef0,
    275                 gamma=self._gamma, epsilon=self.epsilon,
--> 276                 max_iter=self.max_iter, random_seed=random_seed)
    277
    278         self._warn_from_fit_status()

KeyboardInterrupt:
```

[8]:
```
x = [float(i)/1000 for i in dick.keys()]
y = dick.values()

%matplotlib notebook
plt.scatter(x,y,)
plt.xlabel("C-value")
plt.ylabel("r^2-score")
plt.grid()
plt.legend()
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
No handles with labels found to put in legend.
```

[10]:
```
#Main loop:
%matplotlib notebook
for path in ["/home/18005152/notebooks/zero/DATA/solar_001.npy"]:
 ↪#sorted(paths_np):
    houses = Jef_Func()
    df = houses[0]['solarOut_delta']
    df = df.resample('60min').sum()
    #what is the house number?:
    house_number = path[-7:-4]
    sheet = path[33:path.index(house_number)-1]

    sdate = "2019-08-23"
    edate = "2019-08-25"

    y = (df[sdate:edate]-(df[sdate:edate].values)[0]).values.reshape(-1,1).
 ↪tolist()
    y = [i[0] for i in y]
    x = ((df[sdate:edate].index).values.reshape(-1,1)).tolist()

    X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=1/3,␣
 ↪random_state=0,shuffle=False)

    svr = SVR(C=500)
    svr.fit(X_train,y_train)
    plt.subplots(figsize=(10,5))
    plt.plot(X_train,svr.predict(X_train),color='black',label="SVR prediction")
    plt.plot(X_test,svr.predict(X_test),color='g',label="SVR prediction")
    plt.scatter(X_train,y_train,color="r",label="Training Data")
    plt.scatter(X_test,y_test,color="b",label="Testing Data")
    plt.xlabel("Unix Time [s]")
    plt.ylabel("Solar production hourly [kWh]")
    plt.grid()
    plt.title("R^2= "+str(svr.score(X_test,y_test)))
    plt.legend(loc="upper right")
    plt.show()
    plt.savefig("SVG_500.png",dpi=1000)

    print("R_score model:")
    print(svr.score(X_test,y_test))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
R_score model:
-1.9265086259127324
```

[15]:
```
#Main loop:
%matplotlib notebook
for path in ["/home/18005152/notebooks/zero/DATA/solar_001.npy"]:
 ↪#sorted(paths_np):
    #what is the house number?:
    house_number = path[-7:-4]
    sheet = path[33:path.index(house_number)-1]

    try:
        df = pd.DataFrame(np.load(path))
        df = df.set_index(pd.DatetimeIndex(pd.to_datetime(df[0],unit='s').
 ↪values))
        df = df[1].resample('10min').sum()*1/12000
    except ValueError:
        continue

    date = "2019-06-06"

    y = (df[date:"2019-06-08"]-(df[date:"2019-06-08"].values)[0]).values.
 ↪reshape(-1,1).tolist()
    y = [i[0] for i in y]
    x = ((df[date:"2019-06-08"].index).values.reshape(-1,1)).tolist()

    X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=0.33333,␣
 ↪random_state=0,shuffle=False)

    svr = SVR()
    svr.fit(X_train,y_train)
    plt.scatter(X_train,svr.predict(X_train),color='black',label="SVR␣
 ↪prediction")
    plt.scatter(X_test,svr.predict(X_test),color='g',label="SVR prediction")
    plt.scatter(X_train,y_train,color="r",label="Training Data")
    plt.scatter(X_test,y_test,color="b",label="Testing Data")
    plt.legend()
    plt.show()

    print("R_score model:")
    print(svr.score(X_test,y_test))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
R_score model:
0.21376313316996376
```

2 Is the data an random walk?

[12]:
```
pathN="/home/18005152/notebooks/zero/DATA/solar_001.npy"
df = pd.DataFrame(np.load(pathN))
df = df.set_index(pd.DatetimeIndex(pd.to_datetime(df[0],unit='s').values))
mod = ARMA(df[1]-df[1][0],order=(1,0))
res = mod.fit()
print(res.summary())
print(res.params)
```

```
/opt/jupyterhub/anaconda/lib/python3.6/site-
packages/statsmodels/tsa/base/tsa_model.py:218: ValueWarning: A date index has
been provided, but it has no associated frequency information and so will be
ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
                              ARMA Model Results
==============================================================================
Dep. Variable:                      1   No. Observations:                104834
Model:                     ARMA(1, 0)   Log Likelihood             -791880.266
Method:                       css-mle   S.D. of innovations            461.605
Date:                Wed, 14 Oct 2020   AIC                        1583766.532
Time:                        17:03:33   BIC                        1583795.213
Sample:                             0   HQIC                       1583775.218

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       -944.0133     31.040    -30.413      0.000   -1004.851    -883.176
ar.L1.1        0.9541      0.001   1031.402      0.000       0.952       0.956
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0481           +0.0000j            1.0481            0.0000
-----------------------------------------------------------------------------
const     -944.013281
ar.L1.1      0.954079
dtype: float64
```

3 Check Statistics

- checks the program statistics

[16]:
```
st_df = pd.read_csv("DataCheck_info_NEW.txt",header=None)
values = st_df[[2]].value_counts()
```

```
%matplotlib notebook
plt.subplots(figsize=(30,7))
st_df = st_df.pivot(0,1,3)#.fillna(0)
ax = sns.heatmap(st_df, vmin=0, vmax=0.25, xticklabels=np.arange(121))
ax.set_yticks(np.arange(28))
ax.set_yticklabels(st_df.index)
ax.set_xticks(np.arange(121))
ax.set_xlim(0,120)
plt.title('HEATMAP SHOWING HOW THE SUM OF THE EXCEL & NUMPY DATASETS DIFFER PER␣
 ↪SHEET FOR EVERY HOUSE')
ax.set_ylabel(None)
ax.set_xlabel(None)
plt.savefig('heetmapje_week5_v1.0.png', dpi=1200)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-16-32329f886c97> in <module>
----> 1 st_df = pd.read_csv("DataCheck_info_NEW.txt",header=None)
      2 values = st_df[[2]].value_counts()
      3
      4 get_ipython().run_line_magic('matplotlib', 'notebook')
      5 plt.subplots(figsize=(30,7))

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/pandas/io/parsers.py in␣
 ↪read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col,␣
 ↪usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters,␣
 ↪true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows,␣
 ↪na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates␣
 ↪infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates,␣
 ↪iterator, chunksize, compression, thousands, decimal, lineterminator,␣
 ↪quotechar, quoting, doublequote, escapechar, comment, encoding, dialect,␣
 ↪error_bad_lines, warn_bad_lines, delim_whitespace, low_memory, memory_map,␣
 ↪float_precision)
    684     )
    685
--> 686     return _read(filepath_or_buffer, kwds)
    687
    688

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/pandas/io/parsers.py in␣
 ↪_read(filepath_or_buffer, kwds)
    450
    451     # Create the parser.
--> 452     parser = TextFileReader(fp_or_buf, **kwds)
    453
    454     if chunksize or iterator:

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/pandas/io/parsers.py in␣
 ↪__init__(self, f, engine, **kwds)
```

```
    934             self.options["has_index_names"] = kwds["has_index_names"]
    935
--> 936         self._make_engine(self.engine)
    937
    938     def close(self):

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/pandas/io/parsers.py in␣
 ↪_make_engine(self, engine)
   1166     def _make_engine(self, engine="c"):
   1167         if engine == "c":
-> 1168             self._engine = CParserWrapper(self.f, **self.options)
   1169         else:
   1170             if engine == "python":

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/pandas/io/parsers.py in␣
 ↪__init__(self, src, **kwds)
   1996         kwds["usecols"] = self.usecols
   1997
-> 1998         self._reader = parsers.TextReader(src, **kwds)
   1999         self.unnamed_cols = self._reader.unnamed_cols
   2000

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.
 ↪_setup_parser_source()

FileNotFoundError: [Errno 2] No such file or directory: 'DataCheck_info_NEW.txt
```

4 Heatmap without Empty or Pickled tables

[5]:
```
st_df = pd.read_csv("Z.DataCheck_info_NEW.txt",header=None)
st_df = st_df[st_df[2]!='Pickle problem']
st_df = st_df[st_df[2]!='Empty Table']

%matplotlib notebook
plt.subplots(figsize=(30,7))
st_df = st_df.pivot(0,1,3)
ax = sns.heatmap(st_df, vmin=0, vmax=0.25, xticklabels=np.arange(121))
ax.set_yticks(np.arange(len(st_df.index)))
ax.set_yticklabels(st_df.index)
ax.set_xticks(np.arange(121))
ax.set_xlim(0,120)
plt.title('HEATMAP SHOWING HOW THE SUM OF THE EXCEL & NUMPY DATASETS DIFFER PER␣
 ↪SHEET FOR EVERY HOUSE')
```

```
ax.set_ylabel(None)
ax.set_xlabel(None)
plt.savefig('heetmapje_week5_v1.0.png', dpi=1200)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

5 Check single file

[36]:
```
ex_df = pd.read_excel(loadpath_ex+'058'+'.xlsx', engine="openpyxl",␣
↪sheet_name='thermostat')
np_data = np.nan_to_num(np.load(loadpath_np+'thermostat_058.npy'),np.nan)

ex_sum = ex_df.sum().sum()
np_sum = np_data.sum()

print(abs(np_sum-ex_sum))
```

```
0.0
```