```
import pandas as pd

```

```
dataset = pd.read_csv('household_power_consumption.txt', sep=';', header=0, low_
```

```
dataset.shape
```

```
(2075259, 7)
```

```
dataset.head()
```

	Global_active_power	Global_reactive_power	Voltage	Global_intensit
datetime				
2006-12-16 17:24:00	4.216	0.418	234.840	18.40
2006-12-16 17:25:00	5.360	0.436	233.630	23.00
2006-12-16 17:26:00	5.374	0.498	233.290	23.00
2006-12-16 17:27:00	5.388	0.502	233.740	23.00

```
import numpy as np
dataset['Global_active_power']=dataset['Global_active_power'].apply(pd.to_numer
df2=dataset['Global_active_power'][0:9999]
df2.dropna(inplace=True)

window=60
#Making  X, y
X=[]
y=[]
for i in range(window,len(df2)):
    y.append(df2[i])
    X.append(df2[i-window:i].T)
#Convert to numpy array
X=np.array(X)
y=np.array(y)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, rand
df2.head()
```

```
datetime
2006-12-16    1209.176
2006-12-17    3390.460
2006-12-18    2203.826
2006-12-19    1666.194
2006-12-20    2225.748
Name: Global_active_power, dtype: float64
```

```
from numpy import nan
from numpy import isnan
from pandas import read_csv
from pandas import to_numeric

# fill missing values with a value at the same time one day ago
def fill_missing(values):
  one_day = 60 * 24
  for row in range(values.shape[0]):
    for col in range(values.shape[1]):
      if isnan(values[row, col]):
        values[row, col] = values[row - one_day, col]

# load all data
dataset = read_csv('household_power_consumption.txt', sep=';', header=0, low_mem
# mark all missing values
dataset.replace('?', nan, inplace=True)
# make dataset numeric
dataset = dataset.astype('float32')
# fill missing
fill_missing(dataset.values)
# add a column for for the remainder of sub metering
values = dataset.values
dataset['sub_metering_4'] = (values[:,0] * 1000 / 60) - (values[:,4] + values[:
# save updated dataset
dataset.to_csv('household_power_consumption.csv')
```

```
from pandas import read_csv
# load the new file
dataset = read_csv('household_power_consumption.csv', header=0, infer_datetime_
# resample data to daily
daily_groups = dataset.resample('D')
daily_data = daily_groups.sum()
# summarize
print(daily_data.shape)
print(daily_data.head())
# save
daily_data.to_csv('household_power_consumption_days.csv')
```

```
(1442, 8)
            Global_active_power  Global_reactive_power    Voltage  \
datetime
2006-12-16             1209.176                 34.922   93552.53
2006-12-17             3390.460                226.006  345725.32
2006-12-18             2203.826                161.792  347373.64
2006-12-19             1666.194                150.942  348479.01
2006-12-20             2225.748                160.998  348923.61

            Global_intensity  Sub_metering_1  Sub_metering_2  Sub_metering_3
datetime
2006-12-16            5180.8             0.0           546.0          4926.0
2006-12-17           14398.6          2033.0          4187.0         13341.0
2006-12-18            9247.2          1063.0          2621.0         14018.0
2006-12-19            7094.0           839.0          7602.0          6197.0
2006-12-20            9313.0             0.0          2648.0         14063.0

            sub_metering_4
datetime
2006-12-16    14680.933319
2006-12-17    36946.666732
2006-12-18    19028.433281
2006-12-19    13131.900043
2006-12-20    20384.800011
```

```python
def evaluate_forecasts(actual, predicted):
  scores = list()
  # calculate an RMSE score for each day
  for i in range(actual.shape[1]):
    # calculate mse
    mse = mean_squared_error(actual[:, i], predicted[:, i])
    # calculate rmse
    rmse = sqrt(mse)
    # store
    scores.append(rmse)
  # calculate overall RMSE
  s = 0
  for row in range(actual.shape[0]):
    for col in range(actual.shape[1]):
      s += (actual[row, col] - predicted[row, col])**2
  score = sqrt(s / (actual.shape[0] * actual.shape[1]))
  return score, scores
```

```python
from numpy import split
from numpy import array
from pandas import read_csv

# split a univariate dataset into train/test sets
def split_dataset(data):
  # split into standard weeks
  train, test = data[1:-328], data[-328:-6]
  # restructure into windows of weekly data
  train = array(split(train, len(train)/7))
  test = array(split(test, len(test)/7))
  return train, test
```

```python

# load the new file
dataset = read_csv('household_power_consumption_days.csv', header=0, infer_date
train, test = split_dataset(dataset.values)
# validate train data
print(train.shape)
print(train[0, 0, 0], train[-1, -1, 0])
# validate test
print(test.shape)
print(test[0, 0, 0], test[-1, -1, 0])
```

```
(159, 7, 8)
3390.46 1309.2679999999998
(46, 7, 8)
2083.4539999999984 2197.006000000004
```

```python
dataset.head()
```

	Global_active_power	Global_reactive_power	Voltage	Global_intensi
datetime				
2006-12-16	1209.176	34.922	93552.53	5180
2006-12-17	3390.460	226.006	345725.32	14398
2006-12-18	2203.826	161.792	347373.64	9247
2006-12-19	1666.194	150.942	348479.01	7094
2006-12-20	2225.748	160.998	348923.61	9313

```python
dataset.shape
```

```
(1442, 8)
```

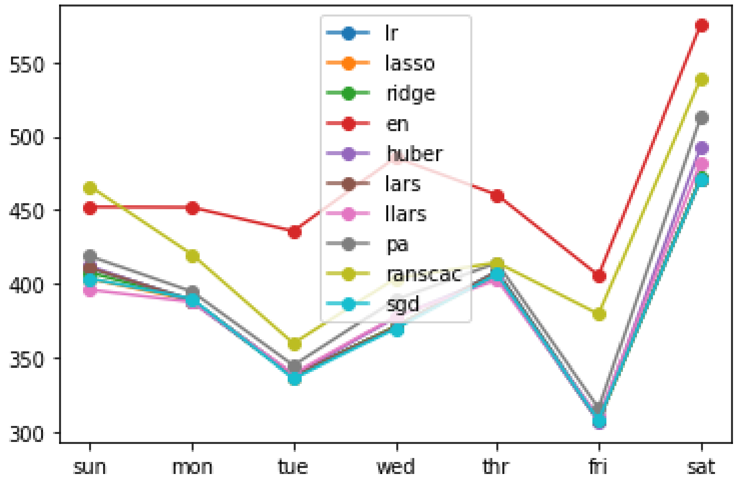
```python
# evaluate a single model
def evaluate_model(model, train, test, n_input):
  # history is a list of weekly data
  history = [x for x in train]
  # walk-forward validation over each week
  predictions = list()
  for i in range(len(test)):
    # predict the week
    yhat_sequence = ...
    # store the predictions
    predictions.append(yhat_sequence)
    # get real observation and add to history for predicting the next week
    history.append(test[i, :])
  predictions = array(predictions)
  # evaluate predictions days for each week
  score, scores = evaluate_forecasts(test[:, :, 0], predictions)
  return score, scores
```

```
from math import sqrt
from numpy import split
from numpy import array
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import HuberRegressor
from sklearn.linear_model import Lars
from sklearn.linear_model import LassoLars
from sklearn.linear_model import PassiveAggressiveRegressor
from sklearn.linear_model import RANSACRegressor
from sklearn.linear_model import SGDRegressor

# split a univariate dataset into train/test sets
def split_dataset(data):
  # split into standard weeks
  train, test = data[1:-328], data[-328:-6]
  # restructure into windows of weekly data
  train = array(split(train, len(train)/7))
  test = array(split(test, len(test)/7))
  return train, test

# evaluate one or more weekly forecasts against expected values
def evaluate_forecasts(actual, predicted):
  scores = list()
  # calculate an RMSE score for each day
  for i in range(actual.shape[1]):
    # calculate mse
    mse = mean_squared_error(actual[:, i], predicted[:, i])
    # calculate rmse
    rmse = sqrt(mse)
    # store
    scores.append(rmse)
  # calculate overall RMSE
  s = 0
  for row in range(actual.shape[0]):
    for col in range(actual.shape[1]):
      s += (actual[row, col] - predicted[row, col])**2
  score = sqrt(s / (actual.shape[0] * actual.shape[1]))
  return score, scores

# summarize scores
def summarize_scores(name, score, scores):
  s_scores = ', '.join(['%.1f' % s for s in scores])
  print('%s: [%.3f] %s' % (name, score, s_scores))

# prepare a list of ml models
def get_models(models=dict()):
  # linear models
  models['lr'] = LinearRegression()
  models['lasso'] = Lasso()
  models['ridge'] = Ridge()
  models['en'] = ElasticNet()
  models['huber'] = HuberRegressor()
  models['lars'] = Lars()
  models['llars'] = LassoLars()
  models['pa'] = PassiveAggressiveRegressor(max_iter=1000, tol=1e-3)
  models['ranscac'] = RANSACRegressor()
  models['sgd'] = SGDRegressor(max_iter=1000, tol=1e-3)
  print('Defined %d models' % len(models))
  return models

# create a feature preparation pipeline for a model
def make_pipeline(model):
  steps = list()
  # standardization
  steps.append(('standardize', StandardScaler()))
  # normalization
  steps.append(('normalize', MinMaxScaler()))
  # the model
  steps.append(('model', model))
  # create pipeline
  pipeline = Pipeline(steps=steps)
  return pipeline

# make a recursive multi-step forecast
def forecast(model, input_x, n_input):
  yhat_sequence = list()
  input_data = [x for x in input_x]
  for j in range(7):
    # prepare the input data
    X = array(input_data[-n_input:]).reshape(1, n_input)
    # make a one-step forecast
    yhat = model.predict(X)[0]
    # add to the result
    yhat_sequence.append(yhat)
    # add the prediction to the input
    input_data.append(yhat)
  return yhat_sequence

# convert windows of weekly multivariate data into a series of total power
def to_series(data):
  # extract just the total power from each week
  series = [week[:, 0] for week in data]
  # flatten into a single series
  series = array(series).flatten()
  return series

# convert history into inputs and outputs
```

```
def to_supervised(history, n_input):
  # convert history to a univariate series
  data = to_series(history)
  X, y = list(), list()
  ix_start = 0
  # step over the entire history one time step at a time
  for i in range(len(data)):
    # define the end of the input sequence
    ix_end = ix_start + n_input
    # ensure we have enough data for this instance
    if ix_end < len(data):
      X.append(data[ix_start:ix_end])
      y.append(data[ix_end])
    # move along one time step
    ix_start += 1
  return array(X), array(y)

# fit a model and make a forecast
def sklearn_predict(model, history, n_input):
  # prepare data
  train_x, train_y = to_supervised(history, n_input)
  # make pipeline
  pipeline = make_pipeline(model)
  # fit the model
  pipeline.fit(train_x, train_y)
  # predict the week, recursively
  yhat_sequence = forecast(pipeline, train_x[-1, :], n_input)
  return yhat_sequence

# evaluate a single model
def evaluate_model(model, train, test, n_input):
  # history is a list of weekly data
  history = [x for x in train]
  # walk-forward validation over each week
  predictions = list()
  for i in range(len(test)):
    # predict the week
    yhat_sequence = sklearn_predict(model, history, n_input)
    # store the predictions
    predictions.append(yhat_sequence)
    # get real observation and add to history for predicting the next week
    history.append(test[i, :])
  predictions = array(predictions)
  # evaluate predictions days for each week
  score, scores = evaluate_forecasts(test[:, :, 0], predictions)
  return score, scores

# load the new file
dataset = read_csv('household_power_consumption_days.csv', header=0, infer_date
# split into train and test
train, test = split_dataset(dataset.values)
# prepare the models to evaluate
models = get_models()
n_input = 7
# evaluate each model
```

```
days = ['sun', 'mon', 'tue', 'wed', 'thr', 'fri', 'sat']
for name, model in models.items():
  # evaluate and get scores
  score, scores = evaluate_model(model, train, test, n_input)
  # summarize scores
  summarize_scores(name, score, scores)
  # plot scores
  pyplot.plot(days, scores, marker='o', label=name)
# show plot
pyplot.legend()
pyplot.show()
```

```
Defined 10 models
lr: [388.388] 411.0, 389.1, 338.0, 370.8, 408.5, 308.3, 471.1
lasso: [386.838] 403.6, 388.9, 337.3, 371.1, 406.1, 307.6, 471.6
ridge: [387.659] 407.9, 388.6, 337.5, 371.2, 407.0, 307.7, 471.7
en: [469.337] 452.2, 451.9, 435.8, 485.7, 460.4, 405.8, 575.1
huber: [392.465] 412.1, 388.0, 337.9, 377.3, 405.6, 306.9, 492.5
lars: [388.388] 411.0, 389.1, 338.0, 370.8, 408.5, 308.3, 471.1
llars: [388.406] 396.1, 387.8, 339.3, 377.8, 402.9, 310.3, 481.9
pa: [402.869] 418.8, 394.9, 344.9, 389.3, 414.3, 316.2, 512.5
ranscac: [429.649] 466.2, 420.0, 359.8, 403.8, 414.5, 379.6, 538.5
sgd: [386.709] 403.7, 390.0, 336.1, 369.7, 406.7, 308.0, 471.0
```


lr
lasso
ridge
en
huber
lars
llars
pa
ranscac
sgd
sun
mon
tue
wed
thr
fri
sat