homework2

September 18, 2019

NAME: Jacob Duvall

1 Homework 2

1.0.1 Objectives

- Object orientation in Python
- Constructing Data Pre-processing Pipelines
  - Imputing
  - Filtering
  - Simple Numerical Methods
- Do not save work within the ml_practices folder
  - create a folder in your home directory for assignments, and copy the templates there

1.0.2 General References

- Sci-kit Learn Pipelines
- Sci-kit Learn Impute
- Sci-kit Learn Preprocessing
- Pandas Interpolate
- Pandas fillna()

[1]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin

FIGWIDTH = 10
FIGHEIGHT = 2

%matplotlib inline
```

2 LOAD DATA

[2]:
```
fname = '~/ml_practices/imports/datasets/baby1/subject_k1_w10_hw2.csv'
baby_data_raw = pd.read_csv(fname)
baby_data_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 7 columns):
time             15000 non-null float64
left_wrist_x     13458 non-null float64
left_wrist_y     13454 non-null float64
left_wrist_z     13454 non-null float64
right_wrist_x    13514 non-null float64
right_wrist_y    13514 non-null float64
right_wrist_z    13514 non-null float64
dtypes: float64(7)
memory usage: 820.4 KB
```

[3]:
```
""" TODO
Call describe() on the data to get summary statistics
"""

baby_data_raw.describe()
```

[3]:
```
               time  left_wrist_x  left_wrist_y  left_wrist_z  right_wrist_x  \
count  15000.000000  13458.000000  13454.000000  13454.000000   13514.000000
mean     149.990000      0.243580      0.162076     -0.044767       0.271218
std       86.605427      0.084823      0.093114      0.060566       0.055190
min        0.000000      0.027525     -0.046680     -0.186060       0.081230
25%       74.995000      0.177911      0.096319     -0.082849       0.238649
50%      149.990000      0.251879      0.154445     -0.045112       0.277340
75%      224.985000      0.308732      0.245144     -0.004720       0.314673
max      299.980000      0.389957      0.334027      0.147053       0.396959

       right_wrist_y  right_wrist_z
count   13514.000000   13514.000000
mean       -0.120768      -0.207248
std         0.047123       0.054263
min        -0.275120      -0.311197
25%        -0.140773      -0.245453
50%        -0.111330      -0.216992
75%        -0.085764      -0.158773
max        -0.040851      -0.007693
```

[4]:
```
""" TODO
Call head() on the data to observe the first few examples
"""
```

```
baby_data_raw.head()
```

[4]:

```
   time  left_wrist_x  left_wrist_y  left_wrist_z  right_wrist_x  \
0  0.00           NaN      0.293503     -0.092803       0.314738
1  0.02           NaN      0.293445     -0.092968       0.315143
2  0.04           NaN           NaN           NaN       0.315974
3  0.06           NaN      0.293285     -0.093356       0.316709
4  0.08      0.163611      0.293237     -0.093475       0.317206

   right_wrist_y  right_wrist_z
0      -0.113438      -0.154972
1      -0.113476      -0.154807
2      -0.113521      -0.154429
3      -0.113555      -0.154063
4      -0.113534      -0.153886
```

[5]:

```
""" TODO
Call tail() on the data to observe the last few examples
"""
baby_data_raw.tail()
```

[5]:

```
         time  left_wrist_x  left_wrist_y  left_wrist_z  right_wrist_x  \
14995  299.90      0.371656           NaN           NaN       0.202332
14996  299.92      0.371723           NaN           NaN       0.202157
14997  299.94      0.371801           NaN           NaN       0.201895
14998  299.96      0.371866           NaN           NaN       0.201533
14999  299.98      0.371907           NaN           NaN       0.201166

       right_wrist_y  right_wrist_z
14995      -0.073395      -0.310776
14996      -0.073288      -0.310726
14997      -0.073102      -0.310798
14998      -0.072929      -0.310848
14999      -0.072672      -0.310929
```

[6]:

```
""" TODO
Display the column names for the data
"""
for column in baby_data_raw.columns:
    print(column)
```

```
time
left_wrist_x
left_wrist_y
left_wrist_z
right_wrist_x
right_wrist_y
right_wrist_z
```

[7]:
```python
""" TODO
Determine whether any data are NaN. Use isna() and
any() to obtain a summary of which features have at
least one missing value
"""

print('Does the feature have NaN values?')
for baby in baby_data_raw:
    print(baby, ":", np.any(np.isnan(baby_data_raw[baby])))
```

```
Does the feature have NaN values?
time : False
left_wrist_x : True
left_wrist_y : True
left_wrist_z : True
right_wrist_x : True
right_wrist_y : True
right_wrist_z : True
```

3 Create Pipeline Elements

In the lecture, some of the Pipeline components might have taken in or returned numpy arrays and others pandas DataFrames. For this assignment, transform methods for all the Pipeline components will take input as a pandas DataFrame and return a DataFrame.

[ ]:
```python

```

[8]:
```python
""" PROVIDED
Pipeline component object for selecting a subset of specified features
"""
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribs):
        self.attribs = attribs

    def fit(self, x, y=None):
        return self

    def transform(self, X):
        '''
        PARAMS:
            X: is a DataFrame
        RETURNS: a DataFrame of the selected attributes
        '''
        return X[self.attribs]


""" TODO
Complete the Pipeline component object for interpolating and filling in
```

```
gaps within the data. Whenever data are missing inbetween valid values,
use interpolation to fill in the gaps. For example,
    1.2 NaN NaN 1.5
becomes
    1.2 1.3 1.4 1.5

Whenever data are missing on the edges of the data, fill in the gaps
with the first available valid value. For example,
    NaN NaN 2.3 3.6 3.2 NaN
becomes
    2.3 2.3 2.3 3.6 3.2 3.2
The transform() method should fill in the holes and the edge cases.
"""
class InterpolationImputer(BaseEstimator, TransformerMixin):
    def __init__(self, method='quadratic'): #Dr fagg said should be linear?
        self.method = method

    def fit(self, x, y=None):
        return self

    def transform(self, X):
        '''
        PARAMS:
            X: is a DataFrame
        RETURNS: a DataFrame without NaNs
        '''
        Xcopy = X.copy()
        Xout = Xcopy.interpolate()

        for features in Xout:
            count = 0
            filler = 0
            for i in Xout[features]:
                if np.isnan(i):
                    count = count + 1
                else:
                    filler = i
                    break
            if count > 0:
                for j in range(count):
                    Xout[features][j] = filler

        return Xout


""" TODO
Complete the Pipeline component object for smoothing specific features
```

```
using a gaussian kernel. Use the following formula to apply the filter:
    x'[t] = ( w[0]*x[t-3] + w[1]*x[t-2] + w[2]*x[t-1] + w[3]*x[t]
           + w[4]*x[t+1] + w[5]*x[t+2] + w[6]*x[t+3]) //
    DISCLAIMER: if you implement this computation on more than one line,
                make sure to place parentheses around the entire expression
                such that the interpreter reads the lines as all part of
                one expression
This can be implemented similarly to how the derivative is computed.
Additionally, pad both ends of x with three instances of the adjacent
value, before filtering, to maintain the original signal length and
smoothness. For example,
                1.3 2.1 4.4 4.1 3.2
would be padded as
    1.3 1.3 1.3 1.3 2.1 4.4 4.1 3.2 3.2 3.2 3.2
"""

def computeweights(length=3, sig=1):
    '''
    Computes the weights for a Gaussian filter kernel
    PARAMS:
        length: the number of terms in the filter kernel
        sig: the standard deviation (i.e. the scale) of the Gaussian
    RETURNS: a list of filter weights for the Gaussian kernel
    '''
    x = np.linspace(-2.5, 2.5, length)
    kernel = stats.norm.pdf(x, scale=sig)
    return kernel / kernel.sum()

class GaussianFilter(BaseEstimator, TransformerMixin):
    def __init__(self, attribs=None, kernelsize=3, sig=1): # kernelsize is the␣
 ↪7 elements, and sigma of 2
        self.attribs = attribs
        self.kernelsize = kernelsize
        self.sig = sig
        self.weights = computeweights(length=kernelsize, sig=sig)
        print("KERNEL WEIGHTS", self.weights)

    def fit(self, x, y=None):
        return self

    def transform(self, X): # TODO
        '''
        PARAMS:
            X: is a DataFrame
        RETURNS: a DataFrame with the smoothed signals
        '''
        w = self.weights
```

```
        Xreturn = X.copy()
        Xout = X.copy()
        if self.attribs == None:
            self.attribs = Xout.columns

        count = 0
        lister = list()
        lister1 = list()
        for feature in self.attribs:
            count = count + 1
            lister.append(X[feature][0])
            lister1.append(X[feature][len(X)-1])

        # indexing reference: https://pandas-docs.github.io/pandas-docs-travis/
↪user_guide/indexing.html
        for item in range(3):
            Xout.loc[-1] = lister
            Xout.index = Xout.index + 1
            Xout = Xout.sort_index()
            Xout.loc[len(Xout)] = lister1

        for field in self.attribs:
            values = Xout[field].values

            gaussian = (self.weights[0]*values[:-6] + self.weights[1]*values[1:
↪-5] +
                   self.weights[2]*values[2:-4] + self.weights[3]*values[3:-3] +
                   self.weights[4]*values[4:-2] + self.weights[5]*values[5:-1] +
                   self.weights[6]*values[6:])

            np.append(gaussian,0)

            name = field

            Xreturn[name] = pd.Series(gaussian)


        return Xreturn


""" PROVIDED
Pipeline component object for computing the derivative for specified features
"""
class DerivativeComputer(BaseEstimator, TransformerMixin):
    def __init__(self, attribs=None, prefix='d_', dt=1.0):
        self.attribs = attribs
        self.prefix = prefix
```

```
        self.dt = dt

    def fit(self, x, y=None):
        return self

    def transform(self, X):
        '''
        PARAMS:
            X: is a DataFrame
        RETURNS: a DataFrame with additional features for the derivatives
        '''
        Xout = X.copy()
        if self.attribs == None:
            self.attribs = Xout.columns

        for attrib in self.attribs:
            vals = Xout[attrib].values
            diff = vals[1:] - vals[0:-1]
            deriv = diff / self.dt
            deriv = np.append(deriv, 0)
            attrib_name = self.prefix + attrib
            Xout[attrib_name] = pd.Series(deriv)

        return Xout
```

4 Construct Pipeline

[9]:
```
selected_names = ['left_wrist_x', 'left_wrist_y', 'left_wrist_z']
selected_inds = [baby_data_raw.columns.get_loc(name) for name in␣
 ↪selected_names] #[1,2,3]
nselected = len(selected_names) # 3
time = baby_data_raw['time'].values # [0.0000e+00 2.0000e-02 4.0000e-02 ... 2.
 ↪9994e+02 2.9996e+02 2.9998e+02]
Xsel_raw = baby_data_raw[selected_names].values
```

[10]:
```
"""
TODO
Create a pipeline that:
1. Selects a subset of features
2. Fills gaps within the data by linearly interpolating the values
   in between existing data and fills the remaining gaps at the edges
   of the data with the first or last valid value
3. Compute the derivatives of the selected features. The data are
   sampled at 50 Hz, therefore, the period or elapsed time (dt) between
   the samples is .02 seconds (dt=.02)
"""
```

```
pipe1 = Pipeline([
    ('selector', DataFrameSelector(selected_names)),
    ('linear imputer', InterpolationImputer(method = 'linear')),
    ('derivative', DerivativeComputer(selected_names, dt=.02))
])


#pipe1 = baby_data_raw[selected_names]
#imp = InterpolationImputer(method='linear')
#pipe1_imp = imp.transform(pipe1)

#der = DerivativeComputer(selected_names, dt = .02)
#pipe1_der = der.transform(pipe1_imp)


""" TODO
Create a pipeline that:
1. Selects a subset of features
2. Fills gaps within the data by linearly interpolating the values
   in between existing data and fills the remaining gaps at the edges
   of the data with the first or last valid value
3. Smooth the data with a Gaussian Filter. Use a standard deviation
   of 2 and a kernel size of 7 for the filter
4. Compute the derivatives of the selected features. The data are
   sampled at 50 Hz, therefore, the period or elapsed time (dt) between
   the samples is .02 seconds (dt=.02)
"""

pipe2 = Pipeline([
    ('selector', DataFrameSelector(selected_names)),
    ('linear imputer', InterpolationImputer(method = 'linear')),
    ('gaussian filter', GaussianFilter(attribs = selected_names, kernelsize=7,␣
 ↪sig = 2)),
    ('derivative', DerivativeComputer(selected_names, dt=.02))
])
#pipe2 = baby_data_raw[selected_names]
#imp2 = InterpolationImputer(method = 'linear')
#pipe2_imp = imp2.transform(pipe2)

#gaussian = GaussianFilter(sig = 2, kernelsize=7, attribs=selected_names)
```

```
KERNEL WEIGHTS [0.08868144 0.13687641 0.17759311 0.19369807 0.17759311
0.13687641
 0.08868144]
```

[19]:
```python
""" TODO
Fit both Pipelines to the data and transform the data
"""
pipe1.fit(baby_data_raw)
baby_data1 = pipe1.transform(baby_data_raw)
pipe2.fit(baby_data_raw)
baby_data2 = pipe2.transform(baby_data_raw)

""" TODO
Display the summary statistics for the pre-processed data
from both pipelines
"""
baby_data1.describe()
```

[19]:
```
       left_wrist_x  left_wrist_y  left_wrist_z  d_left_wrist_x  \
count  15000.000000  15000.000000  15000.000000    15000.000000
mean       0.244186      0.161478     -0.044664        0.000694
std        0.084979      0.093011      0.060630        0.082732
min        0.027525     -0.046680     -0.186060       -1.024850
25%        0.178381      0.096099     -0.082856       -0.012800
50%        0.254316      0.153330     -0.044753        0.000750
75%        0.308836      0.244393     -0.004493        0.014775
max        0.389957      0.334027      0.147053        1.469050

       d_left_wrist_y  d_left_wrist_z
count    15000.000000    15000.000000
mean        -0.000705        0.000002
std          0.058960        0.087525
min         -0.970700       -1.600800
25%         -0.011800       -0.018100
50%         -0.001000       -0.001650
75%          0.010150        0.014550
max          0.717350        0.810550
```

[20]:
```python
baby_data2.describe()
```

[20]:
```
       left_wrist_x  left_wrist_y  left_wrist_z  d_left_wrist_x  \
count  15000.000000  15000.000000  15000.000000    15000.000000
mean       0.244186      0.161478     -0.044664        0.000694
std        0.084935      0.092992      0.060562        0.073687
min        0.027684     -0.046085     -0.185986       -0.910723
25%        0.178182      0.096089     -0.082861       -0.011618
50%        0.254310      0.153358     -0.044708        0.000842
75%        0.308846      0.244420     -0.004485        0.013784
max        0.387130      0.331056      0.146256        1.052638

       d_left_wrist_y  d_left_wrist_z
count    15000.000000    15000.000000
```

```
mean       -0.000705     0.000002
std         0.050054     0.077370
min        -0.642914    -1.177039
25%        -0.011303    -0.016420
50%        -0.001050    -0.001751
75%         0.009236     0.012902
max         0.533001     0.725959
```

[13]:
```python
""" TODO
Display the first few values for the pre-processed data
from both pipelines
"""
baby_data1.head()
```

[13]:
```
   left_wrist_x  left_wrist_y  left_wrist_z  d_left_wrist_x  d_left_wrist_y  \
0      0.163611      0.293503     -0.092803         0.00000         -0.0029
1      0.163611      0.293445     -0.092968         0.00000         -0.0040
2      0.163611      0.293365     -0.093162         0.00000         -0.0040
3      0.163611      0.293285     -0.093356         0.00000         -0.0024
4      0.163611      0.293237     -0.093475        -0.01165         -0.0017

   d_left_wrist_z
0        -0.00825
1        -0.00970
2        -0.00970
3        -0.00595
4        -0.00915
```

[14]:
```python
baby_data2.head()
```

[14]:
```
   left_wrist_x  left_wrist_y  left_wrist_z  d_left_wrist_x  d_left_wrist_y  \
0      0.163611      0.293454     -0.092930        0.000000       -0.002032
1      0.163611      0.293414     -0.093034       -0.001033       -0.002479
2      0.163590      0.293364     -0.093168       -0.002654       -0.002599
3      0.163537      0.293312     -0.093315       -0.004538       -0.002366
4      0.163446      0.293265     -0.093468       -0.006805       -0.001789

   d_left_wrist_z
0       -0.005176
1       -0.006693
2       -0.007381
3       -0.007645
4       -0.006904
```

[15]:
```python
""" TODO
Display the last few values for the pre-processed data
from both pipelines
"""
baby_data1.tail()
```

[15]:
```
       left_wrist_x  left_wrist_y  left_wrist_z  d_left_wrist_x  \
14995      0.371656      0.082065     -0.092307         0.00335
14996      0.371723      0.082065     -0.092307         0.00390
14997      0.371801      0.082065     -0.092307         0.00325
14998      0.371866      0.082065     -0.092307         0.00205
14999      0.371907      0.082065     -0.092307         0.00000

       d_left_wrist_y  d_left_wrist_z
14995             0.0             0.0
14996             0.0             0.0
14997             0.0             0.0
14998             0.0             0.0
14999             0.0             0.0
```

[16]:
```
baby_data2.tail()
```

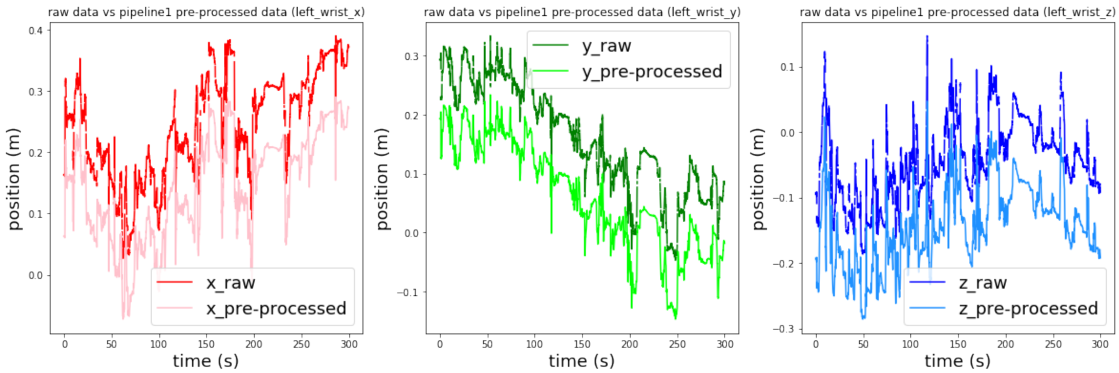
[16]:
```
       left_wrist_x  left_wrist_y  left_wrist_z  d_left_wrist_x  \
14995      0.371689      0.082065     -0.092307        0.002698
14996      0.371743      0.082065     -0.092307        0.002315
14997      0.371789      0.082065     -0.092307        0.002187
14998      0.371833      0.082065     -0.092307        0.001805
14999      0.371869      0.082065     -0.092307        0.000000

       d_left_wrist_y  d_left_wrist_z
14995             0.0             0.0
14996             0.0             0.0
14997             0.0             0.0
14998             0.0             0.0
14999             0.0             0.0
```

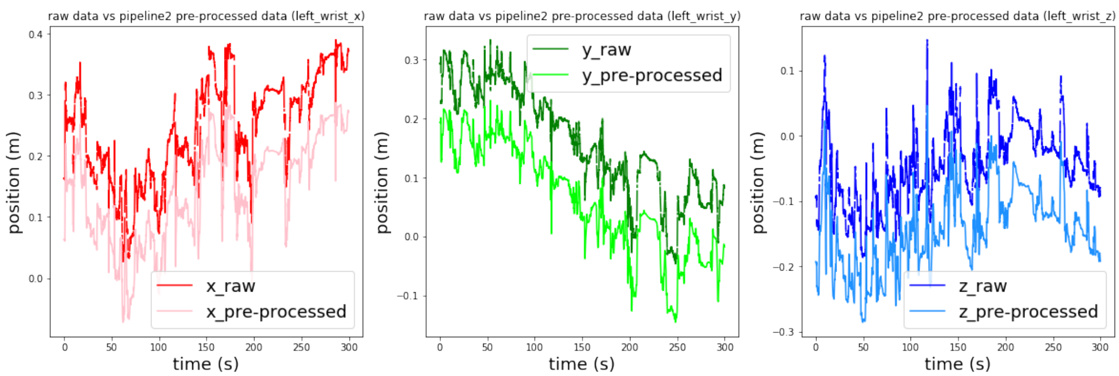
[17]:
```
""" TODO
Construct plots comparing the raw data to the pre-processed data
for each selected feature from both pipelines. For each selected
feature, create a figure displaying the raw data and the cleaned
data in the same subplot. The raw data should be shifted upwards
to clearly observe where the gaps are filled in the cleaned data.
There should be three subplots per feature figure. Each subplot
is in a separate row.
    subplot(1) will compare the original raw data to the pipeline1
               pre-processed data
    subplot(2) will compare the original raw data to the pipeline2
               pre-processed data
    subplot(3) will compare pipeline1 to pipeline2. Set the x limit
               to 45 and 55 seconds
For all subplots, include axis labels, legends and titles.
"""

FIGURESIZE=(10,6)
FONTSIZE=18
```

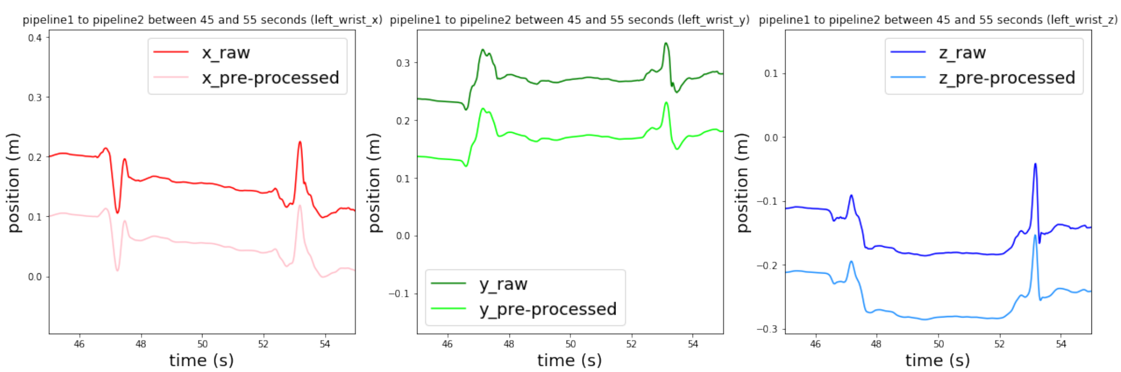
```
plt.figure(figsize = (FIGURESIZE[0]*2, FIGURESIZE[1]))
plt.subplot(131)
plt.plot(baby_data_raw['time'],baby_data_raw['left_wrist_x'],'r')
plt.plot(baby_data_raw['time'],baby_data1['left_wrist_x']-.1,'pink')
plt.title('raw data vs pipeline1 pre-processed data (left_wrist_x)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['x_raw', 'x_pre-processed'], fontsize = FONTSIZE)

plt.subplot(132)
plt.plot(baby_data_raw['time'],baby_data_raw['left_wrist_y'],'g')
plt.plot(baby_data_raw['time'], baby_data1['left_wrist_y']-.1, 'lime')
plt.title('raw data vs pipeline1 pre-processed data (left_wrist_y)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['y_raw', 'y_pre-processed'], fontsize = FONTSIZE)

plt.subplot(133)
plt.plot(baby_data_raw['time'],baby_data_raw['left_wrist_z'],'b')
plt.plot(baby_data_raw['time'], baby_data1['left_wrist_z']-.1, 'dodgerblue')
plt.title('raw data vs pipeline1 pre-processed data (left_wrist_z)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['z_raw', 'z_pre-processed'], fontsize = FONTSIZE)


plt.figure(figsize = (FIGURESIZE[0]*2, FIGURESIZE[1]))
plt.subplot(131)
plt.plot(baby_data_raw['time'],baby_data_raw['left_wrist_x'],'r')
plt.plot(baby_data_raw['time'],baby_data2['left_wrist_x']-.1,'pink')
plt.title('raw data vs pipeline2 pre-processed data (left_wrist_x)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['x_raw', 'x_pre-processed'], fontsize = FONTSIZE)

plt.subplot(132)
plt.plot(baby_data_raw['time'],baby_data_raw['left_wrist_y'],'g')
plt.plot(baby_data_raw['time'], baby_data2['left_wrist_y']-.1, 'lime')
plt.title('raw data vs pipeline2 pre-processed data (left_wrist_y)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['y_raw', 'y_pre-processed'], fontsize = FONTSIZE)

plt.subplot(133)
plt.plot(baby_data_raw['time'],baby_data_raw['left_wrist_z'],'b')
plt.plot(baby_data_raw['time'], baby_data2['left_wrist_z']-.1, 'dodgerblue')
```

```
plt.title('raw data vs pipeline2 pre-processed data (left_wrist_z)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['z_raw', 'z_pre-processed'], fontsize = FONTSIZE)




plt.figure(figsize = (FIGURESIZE[0]*2, FIGURESIZE[1]))
plt.subplot(131)
plt.plot(baby_data_raw['time'],baby_data1['left_wrist_x'],'r')
plt.plot(baby_data_raw['time'],baby_data2['left_wrist_x']-.1,'pink')
plt.title('pipeline1 to pipeline2 between 45 and 55 seconds (left_wrist_x)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.xlim(45,55)
plt.legend(['x_raw', 'x_pre-processed'], fontsize = FONTSIZE)

plt.subplot(132)
plt.plot(baby_data_raw['time'],baby_data1['left_wrist_y'],'g')
plt.plot(baby_data_raw['time'], baby_data2['left_wrist_y']-.1, 'lime')
plt.title('pipeline1 to pipeline2 between 45 and 55 seconds (left_wrist_y)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.xlim(45,55)
plt.legend(['y_raw', 'y_pre-processed'], fontsize = FONTSIZE)

plt.subplot(133)
plt.plot(baby_data_raw['time'],baby_data1['left_wrist_z'],'b')
plt.plot(baby_data_raw['time'], baby_data2['left_wrist_z']-.1, 'dodgerblue')
plt.title('pipeline1 to pipeline2 between 45 and 55 seconds (left_wrist_z)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.xlim(45,55)
plt.legend(['z_raw', 'z_pre-processed'], fontsize = FONTSIZE)
```

[17]:
```
<matplotlib.legend.Legend at 0x7ffb05596518>
```





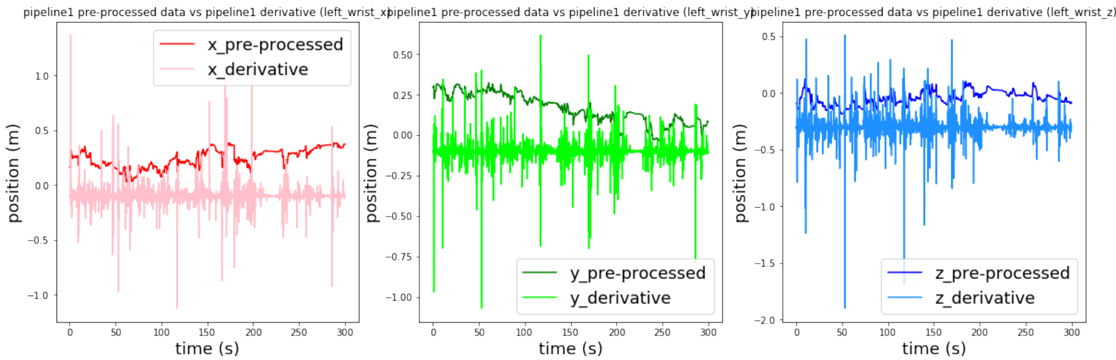


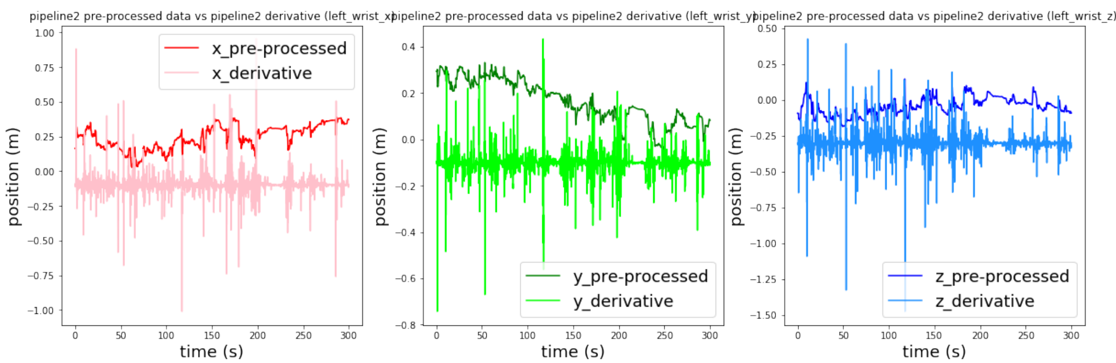
[18]:
```
""" TODO
Construct plots for each feature presenting the feature and its
derivative from both pipelines. Each figure should have
3 subplots:
    1: the pipeline1 feature data and cooresponding derivative
    2: the pipeline2 feature data and corresponding derivative
```

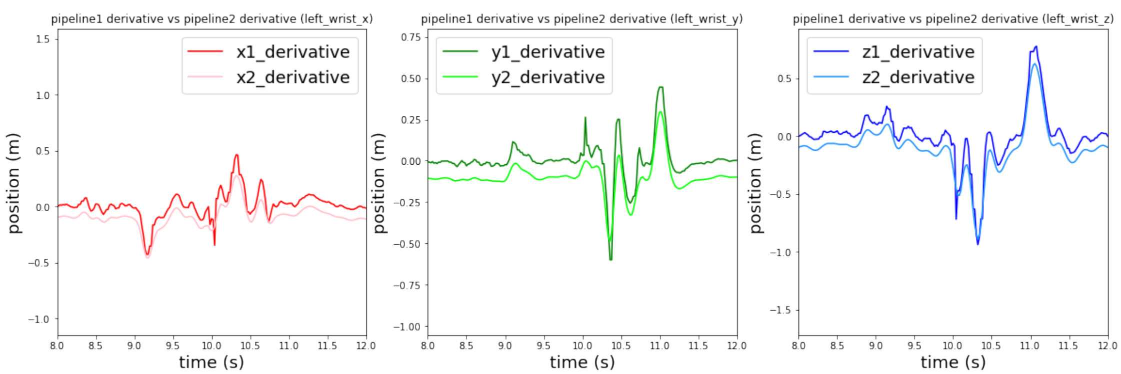
```
    3: pipeline1 derivative and pipeline2 derivative. Set the x limit
       to 8 and 12 seconds.
For all subplots, include axis labels, legends and titles.
"""


FIGURESIZE=(10,6)
FONTSIZE=18

plt.figure(figsize = (FIGURESIZE[0]*2, FIGURESIZE[1]))
plt.subplot(131)
plt.plot(baby_data_raw['time'],baby_data1['left_wrist_x'],'r')
plt.plot(baby_data_raw['time'],baby_data1['d_left_wrist_x']-.1,'pink')
plt.title('pipeline1 pre-processed data vs pipeline1 derivative (left_wrist_x)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['x_pre-processed', 'x_derivative'], fontsize = FONTSIZE)

plt.subplot(132)
plt.plot(baby_data_raw['time'],baby_data1['left_wrist_y'],'g')
plt.plot(baby_data_raw['time'], baby_data1['d_left_wrist_y']-.1, 'lime')
plt.title('pipeline1 pre-processed data vs pipeline1 derivative (left_wrist_y)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['y_pre-processed', 'y_derivative'], fontsize = FONTSIZE)

plt.subplot(133)
plt.plot(baby_data_raw['time'],baby_data1['left_wrist_z'],'b')
plt.plot(baby_data_raw['time'], baby_data1['d_left_wrist_z']-.3, 'dodgerblue')
plt.title('pipeline1 pre-processed data vs pipeline1 derivative (left_wrist_z)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['z_pre-processed', 'z_derivative'], fontsize = FONTSIZE)


plt.figure(figsize = (FIGURESIZE[0]*2, FIGURESIZE[1]))
plt.subplot(131)
plt.plot(baby_data_raw['time'],baby_data2['left_wrist_x'],'r')
plt.plot(baby_data_raw['time'],baby_data2['d_left_wrist_x']-.1,'pink')
plt.title('pipeline2 pre-processed data vs pipeline2 derivative (left_wrist_x)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['x_pre-processed', 'x_derivative'], fontsize = FONTSIZE)

plt.subplot(132)
plt.plot(baby_data_raw['time'],baby_data2['left_wrist_y'],'g')
plt.plot(baby_data_raw['time'], baby_data2['d_left_wrist_y']-.1, 'lime')
```

```
plt.title('pipeline2 pre-processed data vs pipeline2 derivative (left_wrist_y)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['y_pre-processed', 'y_derivative'], fontsize = FONTSIZE)

plt.subplot(133)
plt.plot(baby_data_raw['time'],baby_data2['left_wrist_z'],'b')
plt.plot(baby_data_raw['time'], baby_data2['d_left_wrist_z']-.3, 'dodgerblue')
plt.title('pipeline2 pre-processed data vs pipeline2 derivative (left_wrist_z)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.legend(['z_pre-processed', 'z_derivative'], fontsize = FONTSIZE)




plt.figure(figsize = (FIGURESIZE[0]*2, FIGURESIZE[1]))
plt.subplot(131)
plt.plot(baby_data_raw['time'],baby_data1['d_left_wrist_x'],'r')
plt.plot(baby_data_raw['time'],baby_data2['d_left_wrist_x']-.1,'pink')
plt.title('pipeline1 derivative vs pipeline2 derivative (left_wrist_x)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.xlim(8,12)
plt.legend(['x1_derivative', 'x2_derivative'], fontsize = FONTSIZE)

plt.subplot(132)
plt.plot(baby_data_raw['time'],baby_data1['d_left_wrist_y'],'g')
plt.plot(baby_data_raw['time'], baby_data2['d_left_wrist_y']-.1, 'lime')
plt.title('pipeline1 derivative vs pipeline2 derivative (left_wrist_y)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.xlim(8,12)
plt.legend(['y1_derivative', 'y2_derivative'], fontsize = FONTSIZE)

plt.subplot(133)
plt.plot(baby_data_raw['time'],baby_data1['d_left_wrist_z'],'b')
plt.plot(baby_data_raw['time'], baby_data2['d_left_wrist_z']-.1, 'dodgerblue')
plt.title('pipeline1 derivative vs pipeline2 derivative (left_wrist_z)')
plt.ylabel('position (m)',fontsize = FONTSIZE)
plt.xlabel('time (s)',fontsize = FONTSIZE)
plt.xlim(8,12)
plt.legend(['z1_derivative', 'z2_derivative'], fontsize = FONTSIZE)
```

[18]:
```
<matplotlib.legend.Legend at 0x7ffb04aa9710>
```


pipeline1 pre-processed data vs pipeline1 derivative (left_wrist_x)
x_pre-processed
x_derivative
pipeline1 pre-processed data vs pipeline1 derivative (left_wrist_y)
y_pre-processed
y_derivative
pipeline1 pre-processed data vs pipeline1 derivative (left_wrist_z)
z_pre-processed
z_derivative
position (m)
time (s)


pipeline2 pre-processed data vs pipeline2 derivative (left_wrist_x)
x_pre-processed
x_derivative
pipeline2 pre-processed data vs pipeline2 derivative (left_wrist_y)
y_pre-processed
y_derivative
pipeline2 pre-processed data vs pipeline2 derivative (left_wrist_z)
z_pre-processed
z_derivative
position (m)
time (s)


pipeline1 derivative vs pipeline2 derivative (left_wrist_x)
x1_derivative
x2_derivative
pipeline1 derivative vs pipeline2 derivative (left_wrist_y)
y1_derivative
y2_derivative
pipeline1 derivative vs pipeline2 derivative (left_wrist_z)
z1_derivative
z2_derivative
position (m)
time (s)

[ ]: