homework2-skel

September 23, 2020

NAME: **Nigel Mansell**

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()

1.0.3 Hand-In Procedure

- Execute all cells so they are showing correct results
- Notebook:
  - Submit this file (.ipynb) to the Canvas HW0 dropbox
- PDF:
  - File/Export Notebook As/PDF -> Produces a copy of the notebook in PDF format
  - Submit the PDF file to the Gradescope HW0 dropbox

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

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

FIGWIDTH = 10
FIGHEIGHT = 2

%matplotlib inline
```

2 LOAD DATA

[2]:
```python
fname = '~/demo/data/subject_k1_w10_hw2.csv'

#makes a dataframe from file and prints the info
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]:
```python
""" TODO
Call describe() on the data to get summary statistics
"""
#calling describe on dataframe
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
"""

#calling head on dataframe
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
"""

#calling tail on dataframe
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
"""

#Gets all column names in the form of an array
baby_data_raw.columns.values
```

[6]:
```
array(['time', 'left_wrist_x', 'left_wrist_y', 'left_wrist_z',
       'right_wrist_x', 'right_wrist_y', 'right_wrist_z'], dtype=object)
```

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

#displays if column has NaN or not
baby_data_raw.isna().any()
```

[7]:
```
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
dtype: bool
```

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.

[8]:
```
""" 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:
```

```python
            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'):
        self.method = method

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

    def transform(self, X): # TODO
        '''
        PARAMS:
            X: is a DataFrame
        RETURNS: a DataFrame without NaNs
        '''

        #creates a copy of the dataframe, calls interpolate and bfill to fill␣
 ↪NaN values, return copy
        Xout = X.copy()
        # TODO: Interpolate holes within the data
        Xout = Xout.interpolate()
        # TODO: Fill in the NaNs on the edges of the data
        Xout = Xout.bfill()
        # TODO: return the imputed dataframe
        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
values, before applying the 7-width filter, 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):
        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
```

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

        #pads then uses the function in comment, returns copy
        for attrib in self.attribs:
            values = Xout[attrib].values
            # TODO: pad the data as previously described
            values = np.insert(values, 0, [values[0], values[0], values[0]])
            values = np.append(values, [values[-1], values[-1], values[-1]])
            # TODO: filter the data
            index = 0
            for t in range(3, len(values)-3):
                Xout[attrib][index] = ( w[0]*values[t-3] + w[1]*values[t-2] + w[2]*values[t-1] +
                                       w[3]*values[t]+ w[4]*values[t+1] + w[5]*values[t+2] +
                                       w[6]*values[t+3])
                index+=1
        # TODO: return filtered dataframe
        return Xout


""" 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

[25]:
```
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]
nselected = len(selected_names)
time = baby_data_raw['time'].values
Xsel_raw = baby_data_raw[selected_names].values
```

```
<class 'numpy.ndarray'>
```

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

```
#creating pipe2
pipe2 = Pipeline([
    ('selector',DataFrameSelector(selected_names)),
    ('linear', InterpolationImputer()),
    ('filter', GaussianFilter(selected_names,7, 2)),
    ('derivative', DerivativeComputer(dt=.02))
])
```

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

[11]:
```
""" TODO
Fit both Pipelines to the data and transform the data
"""
#fits and transforms both
baby_data1 = pipe1.fit_transform(baby_data_raw)
baby_data2 = pipe2.fit_transform(baby_data_raw)




""" TODO
Display the summary statistics for the pre-processed data
from both pipelines
"""

#displaying both
display(baby_data1)
display(baby_data2)
```

```
       left_wrist_x  left_wrist_y  left_wrist_z  d_left_wrist_x  \
0          0.163611      0.293503     -0.092803        0.000000
1          0.163611      0.293445     -0.092968        0.000000
2          0.163611      0.293365     -0.093162        0.000000
3          0.163611      0.293285     -0.093356        0.000000
4          0.163611      0.293237     -0.093475       -0.011650
5          0.163378      0.293203     -0.093658       -0.011950
6          0.163139      0.293190     -0.093735       -0.009400
7          0.162951      0.293191     -0.093861       -0.012850
8          0.162694      0.293186     -0.093938       -0.008750
9          0.162519      0.293118     -0.094113       -0.012300
10         0.162273      0.293101     -0.094198       -0.010450
11         0.162064      0.293084     -0.094333       -0.010050
12         0.161863      0.293077     -0.094401       -0.004800
13         0.161767      0.293065     -0.094490       -0.003550
14         0.161696      0.293070     -0.094539        0.001400
15         0.161724      0.293150     -0.094562        0.004700
```

16	0.161818	0.293317	-0.094441	0.009500
17	0.162008	0.293513	-0.094381	0.009800
18	0.162204	0.293671	-0.094338	0.012550
19	0.162455	0.293684	-0.094440	0.012550
20	0.162706	0.293697	-0.094541	0.011850
21	0.162943	0.293628	-0.094772	0.008325
22	0.163110	0.293576	-0.094975	0.008325
23	0.163276	0.293524	-0.095178	-0.003950
24	0.163197	0.293579	-0.095236	-0.003050
25	0.163136	0.293566	-0.095376	-0.008800
26	0.162960	0.293585	-0.095382	-0.010050
27	0.162759	0.293538	-0.095511	-0.010050
28	0.162558	0.293491	-0.095640	-0.009300
29	0.162372	0.293422	-0.095743	-0.014100
...	...	...	...	...
14970	0.373774	0.085690	-0.081843	-0.003700
14971	0.373700	0.085588	-0.082146	-0.001950
14972	0.373661	0.085486	-0.082370	-0.000850
14973	0.373644	0.085377	-0.082788	0.001600
14974	0.373676	0.085400	-0.083032	0.001900
14975	0.373714	0.085329	-0.083364	0.002500
14976	0.373764	0.085266	-0.083650	-0.001350
14977	0.373737	0.085164	-0.084194	-0.007100
14978	0.373595	0.084965	-0.084772	-0.007100
14979	0.373453	0.084766	-0.085350	-0.011950
14980	0.373214	0.084582	-0.086000	-0.016150
14981	0.372891	0.084413	-0.086703	-0.021075
14982	0.372470	0.084140	-0.087417	-0.021075
14983	0.372048	0.083866	-0.088130	-0.016900
14984	0.371710	0.083654	-0.088758	-0.014850
14985	0.371413	0.083326	-0.089609	-0.006200
14986	0.371289	0.083080	-0.090275	-0.006200
14987	0.371165	0.082834	-0.090941	0.001300
14988	0.371191	0.082705	-0.091333	0.002100
14989	0.371233	0.082461	-0.091652	0.002400
14990	0.371281	0.082317	-0.092033	0.005100
14991	0.371383	0.082065	-0.092307	0.006150
14992	0.371506	0.082065	-0.092307	0.006550
14993	0.371637	0.082065	-0.092307	0.000475
14994	0.371646	0.082065	-0.092307	0.000475
14995	0.371656	0.082065	-0.092307	0.003350
14996	0.371723	0.082065	-0.092307	0.003900
14997	0.371801	0.082065	-0.092307	0.003250
14998	0.371866	0.082065	-0.092307	0.002050
14999	0.371907	0.082065	-0.092307	0.000000

	d_left_wrist_y	d_left_wrist_z
0	-0.002900	-0.008250

1	-0.004000	-0.009700
2	-0.004000	-0.009700
3	-0.002400	-0.005950
4	-0.001700	-0.009150
5	-0.000650	-0.003850
6	0.000050	-0.006300
7	-0.000250	-0.003850
8	-0.003400	-0.008750
9	-0.000850	-0.004250
10	-0.000850	-0.006750
11	-0.000350	-0.003400
12	-0.000600	-0.004450
13	0.000250	-0.002450
14	0.004000	-0.001150
15	0.008350	0.006050
16	0.009800	0.003000
17	0.007900	0.002150
18	0.000650	-0.005075
19	0.000650	-0.005075
20	-0.003450	-0.011550
21	-0.002600	-0.010150
22	-0.002600	-0.010150
23	0.002750	-0.002900
24	-0.000650	-0.007000
25	0.000950	-0.000300
26	-0.002350	-0.006450
27	-0.002350	-0.006450
28	-0.003450	-0.005150
29	-0.000300	-0.000600
...	...	...
14970	-0.005100	-0.015150
14971	-0.005100	-0.011200
14972	-0.005450	-0.020900
14973	0.001150	-0.012200
14974	-0.003550	-0.016600
14975	-0.003150	-0.014300
14976	-0.005100	-0.027200
14977	-0.009950	-0.028900
14978	-0.009950	-0.028900
14979	-0.009200	-0.032500
14980	-0.008450	-0.035150
14981	-0.013675	-0.035675
14982	-0.013675	-0.035675
14983	-0.010600	-0.031400
14984	-0.016400	-0.042550
14985	-0.012300	-0.033300
14986	-0.012300	-0.033300
14987	-0.006450	-0.019600

14988	-0.012200	-0.015950
14989	-0.007200	-0.019050
14990	-0.012600	-0.013700
14991	0.000000	0.000000
14992	0.000000	0.000000
14993	0.000000	0.000000
14994	0.000000	0.000000
14995	0.000000	0.000000
14996	0.000000	0.000000
14997	0.000000	0.000000
14998	0.000000	0.000000
14999	0.000000	0.000000

[15000 rows x 6 columns]

	left_wrist_x	left_wrist_y	left_wrist_z	d_left_wrist_x \
0	0.163611	0.293454	-0.092930	0.000000
1	0.163611	0.293414	-0.093034	-0.001033
2	0.163590	0.293364	-0.093168	-0.002654
3	0.163537	0.293312	-0.093315	-0.004538
4	0.163446	0.293265	-0.093468	-0.006805
5	0.163310	0.293229	-0.093606	-0.008588
6	0.163139	0.293200	-0.093736	-0.010108
7	0.162936	0.293178	-0.093855	-0.010991
8	0.162717	0.293156	-0.093973	-0.010829
9	0.162500	0.293135	-0.094085	-0.010186
10	0.162296	0.293114	-0.094195	-0.009302
11	0.162110	0.293096	-0.094296	-0.007476
12	0.161961	0.293089	-0.094385	-0.005258
13	0.161856	0.293108	-0.094441	-0.002108
14	0.161813	0.293159	-0.094470	0.001097
15	0.161835	0.293243	-0.094468	0.004389
16	0.161923	0.293347	-0.094455	0.007123
17	0.162066	0.293457	-0.094445	0.009350
18	0.162253	0.293550	-0.094464	0.010478
19	0.162462	0.293611	-0.094525	0.010826
20	0.162679	0.293631	-0.094638	0.009379
21	0.162866	0.293624	-0.094780	0.007215
22	0.163011	0.293602	-0.094939	0.003927
23	0.163089	0.293584	-0.095085	0.000266
24	0.163094	0.293568	-0.095218	-0.003244
25	0.163030	0.293556	-0.095331	-0.006034
26	0.162909	0.293540	-0.095433	-0.008585
27	0.162737	0.293519	-0.095522	-0.009834
28	0.162540	0.293481	-0.095607	-0.011094
29	0.162319	0.293439	-0.095675	-0.011601
...	...	...	...	...

14970	0.373806	0.085737	-0.081805	-0.003118
14971	0.373744	0.085616	-0.082124	-0.001914
14972	0.373706	0.085517	-0.082436	-0.000673
14973	0.373692	0.085441	-0.082741	0.000124
14974	0.373695	0.085372	-0.083066	-0.000008
14975	0.373694	0.085296	-0.083427	-0.000876
14976	0.373677	0.085202	-0.083839	-0.002708
14977	0.373623	0.085084	-0.084302	-0.005433
14978	0.373514	0.084937	-0.084833	-0.008801
14979	0.373338	0.084764	-0.085422	-0.012289
14980	0.373092	0.084565	-0.086064	-0.014978
14981	0.372793	0.084350	-0.086726	-0.016589
14982	0.372461	0.084117	-0.087417	-0.016679
14983	0.372127	0.083872	-0.088123	-0.015369
14984	0.371820	0.083616	-0.088833	-0.012466
14985	0.371571	0.083364	-0.089514	-0.008757
14986	0.371396	0.083122	-0.090144	-0.005102
14987	0.371294	0.082898	-0.090711	-0.001861
14988	0.371256	0.082681	-0.091205	0.000909
14989	0.371274	0.082495	-0.091591	0.002794
14990	0.371330	0.082340	-0.091883	0.003848
14991	0.371407	0.082221	-0.092082	0.003882
14992	0.371485	0.082135	-0.092211	0.003718
14993	0.371559	0.082087	-0.092283	0.003438
14994	0.371628	0.082065	-0.092307	0.003035
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
0	-0.002032	-0.005176
1	-0.002479	-0.006693
2	-0.002599	-0.007381
3	-0.002366	-0.007645
4	-0.001789	-0.006904
5	-0.001438	-0.006467
6	-0.001136	-0.005942
7	-0.001075	-0.005937
8	-0.001052	-0.005563
9	-0.001050	-0.005522
10	-0.000925	-0.005032
11	-0.000354	-0.004443
12	0.000962	-0.002823
13	0.002541	-0.001439
14	0.004231	0.000107
15	0.005188	0.000615

```
16          0.005500        0.000518
17          0.004667       -0.000945
18          0.003023       -0.003036
19          0.000993       -0.005665
20         -0.000337       -0.007092
21         -0.001113       -0.007982
22         -0.000896       -0.007299
23         -0.000785       -0.006634
24         -0.000585       -0.005668
25         -0.000831       -0.005083
26         -0.001048       -0.004422
27         -0.001875       -0.004275
28         -0.002106       -0.003408
29         -0.002713       -0.003575
...              ...             ...
14970      -0.006075       -0.015953
14971      -0.004921       -0.015615
14972      -0.003832       -0.015228
14973      -0.003409       -0.016269
14974      -0.003822       -0.018061
14975      -0.004717       -0.020590
14976      -0.005876       -0.023133
14977      -0.007355       -0.026556
14978      -0.008675       -0.029468
14979      -0.009948       -0.032085
14980      -0.010755       -0.033102
14981      -0.011625       -0.034571
14982      -0.012268       -0.035293
14983      -0.012790       -0.035486
14984      -0.012584       -0.034075
14985      -0.012108       -0.031479
14986      -0.011206       -0.028326
14987      -0.010841       -0.024697
14988      -0.009286       -0.019340
14989      -0.007773       -0.014591
14990      -0.005961       -0.009958
14991      -0.004305       -0.006455
14992      -0.002363       -0.003565
14993      -0.001117       -0.001215
14994       0.000000        0.000000
14995       0.000000        0.000000
14996       0.000000        0.000000
14997       0.000000        0.000000
14998       0.000000        0.000000
14999       0.000000        0.000000

[15000 rows x 6 columns]
```

[12]:
```python
""" TODO
Display the first few values for the pre-processed data
from both pipelines
"""

#displaying both heads
display(baby_data1.head())
display(baby_data2.head())
```

```
   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
```

```
   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
```

[13]:
```python
""" TODO
Display the last few values for the pre-processed data
from both pipelines
"""
#displaying both tails
display(baby_data1.tail())
display(baby_data2.tail())
```

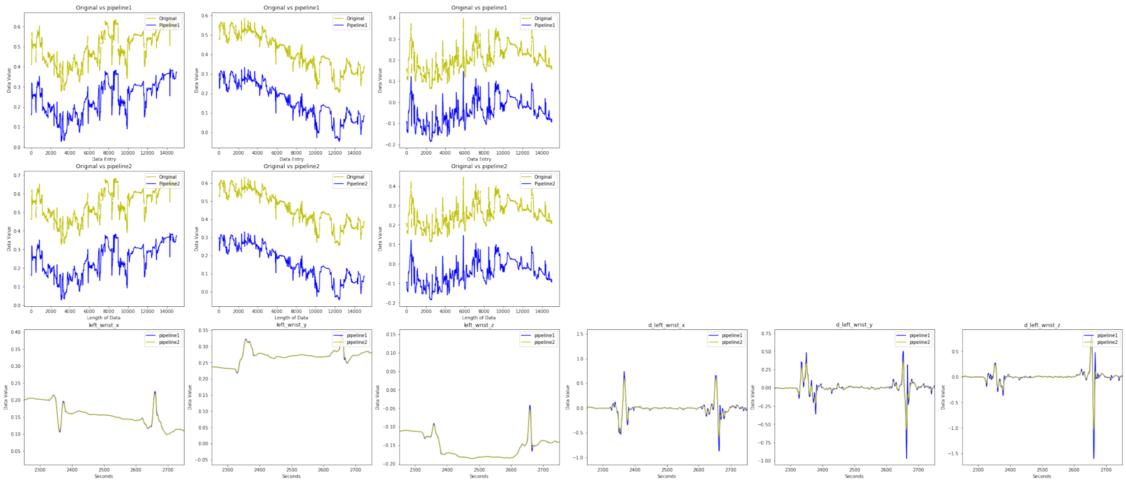
```
       left_wrist_x  left_wrist_y  left_wrist_z  d_left_wrist_x  \
```

```
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


       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
```

[37]:

```python
""" 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.
"""

#creates a copy of data to drop time
noTime = baby_data_raw.copy()
```

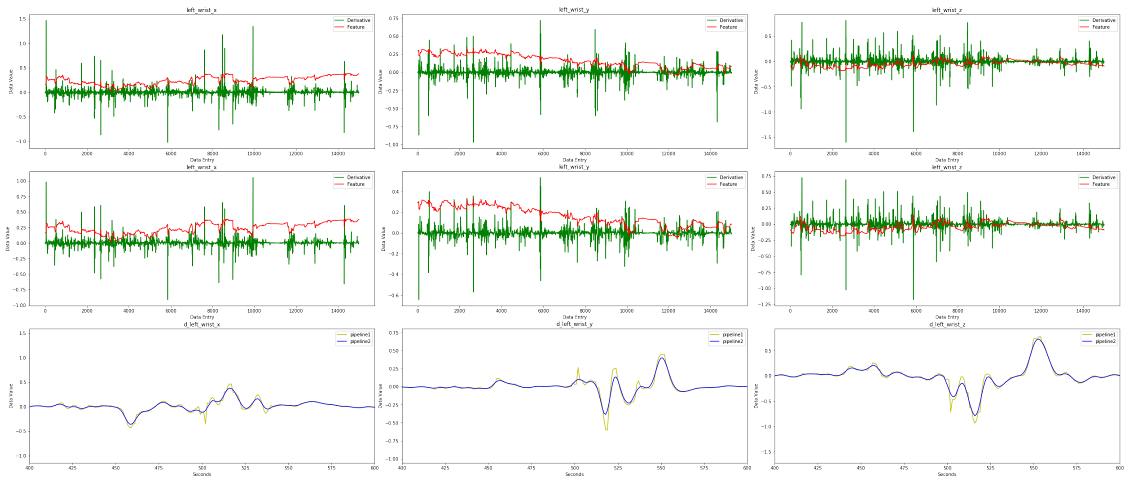
```python
fig, axs = plt.subplots(3, len(noTime.columns.values)-1, figsize=(35, 15))
del noTime['time']
fig.tight_layout(pad=3.0)

#geting column names
origColNames = noTime.columns.values
babyDataNames = baby_data1.columns.values
from matplotlib import transforms
limit1 = np.where(time == 45)
limit2 = np.where(time == 55)

#loops through names to plot the data, transform is used to shit upward.
for i in range(len(origColNames)):
    #to clean up empty subplots
    if(origColNames[i] == babyDataNames[i]):
        axs[0][i].set_xlabel('Data Entry')
        axs[0][i].set_ylabel('Data Value')
        axs[0][i].set_title("Original vs pipeline1")
        transform = transforms.Affine2D().translate(0, 0.250) + axs[0][i].transData
        axs[0][i].plot(noTime[origColNames[i]], color='y', label='Original', transform=transform)
        axs[0][i].plot(baby_data1[babyDataNames[i]], color='b', label='Pipeline1')
        axs[0][i].legend(loc="upper right")
        axs[1][i].set_xlabel('Length of Data')
        transform = transforms.Affine2D().translate(0, 0.30) + axs[1][i].transData
        axs[1][i].set_ylabel('Data Value')
        axs[1][i].set_title("Original vs pipeline2")
        axs[1][i].plot(noTime[origColNames[i]], color='y', label='Original', transform=transform)
        axs[1][i].plot(baby_data2[babyDataNames[i]], color='b', label='Pipeline2')
        axs[1][i].legend(loc="upper right")
    else:
        fig.delaxes(axs[0][i])
        fig.delaxes(axs[1][i])
    axs[2][i].set_xlim(limit1[0], limit2[-1])
    axs[2][i].set_xlabel('Seconds')
    axs[2][i].set_ylabel('Data Value')
    axs[2][i].set_title(babyDataNames[i])
    axs[2][i].plot(baby_data1[babyDataNames[i]], color='b', label='pipeline1')
    axs[2][i].plot(baby_data2[babyDataNames[i]], color='y', label='pipeline2')
    axs[2][i].legend(loc="upper right")
```



[38]:

```
""" 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.
"""
#firstHalf are non derivative, lastHalf are derivative
firstHalf = babyDataNames[:3]
lastHalf = babyDataNames[3:]
fig, axs = plt.subplots(3, 3, figsize=(35, 15))
fig.tight_layout(pad=3.0)
#plots data
limit1 = np.where(time == 8)
limit2 = np.where(time == 12)
for i in range(len(firstHalf)):
    axs[0][i].set_title(firstHalf[i])
    axs[0][i].set_xlabel('Data Entry')
    axs[0][i].set_ylabel('Data Value')
    axs[0][i].plot(baby_data1[lastHalf[i]], color='g', label="Derivative")
    axs[0][i].plot(baby_data1[firstHalf[i]], color='r',label="Feature")
    axs[0][i].legend(loc="upper right")
    axs[1][i].set_title(firstHalf[i])
    axs[1][i].set_xlabel('Data Entry')
    axs[1][i].set_ylabel('Data Value')
    axs[1][i].plot(baby_data2[lastHalf[i]], color='g', label="Derivative")
    axs[1][i].plot(baby_data2[firstHalf[i]], color='r', label="Feature")
    axs[1][i].legend(loc="upper right")
```

```
    axs[2][i].set_xlim(limit1[0], limit2[-1])
    axs[2][i].set_xlabel('Seconds')
    axs[2][i].set_ylabel('Data Value')
    axs[2][i].set_title(lastHalf[i])
    axs[2][i].plot(baby_data1[lastHalf[i]], color='y', label="pipeline1")
    axs[2][i].plot(baby_data2[lastHalf[i]], color='b', label="pipeline2")
    axs[2][i].legend(loc="upper right")
```



[ ]:

[ ]: