wave-forecast-hand-in

June 20, 2016

```
In [1]: import os
        import math
        import pandas as pd
        import numpy as np
        import random
        import dateutil.parser
        import datetime
        import matplotlib.pyplot as plt
        import matplotlib.dates
        %matplotlib inline
        from pandas.tools.plotting import scatter_matrix
        from sklearn import linear_model, preprocessing
        import statsmodels.api as sm
        from IPython.display import display
        import warnings
        import seaborn as sns
        sns.set_style('whitegrid')
        import grab_data as grab_data
```

1 Helper Functions

```
In [2]: default_fig_width = 15
        target_var = 'surf_avg'
        def plot_datetime_series(series, title = '', xlabel='Time', ylabel='Average
            if(type(series) is not list):
                series = [series]
            if(type(label) is not list):
                label = [label]
            for i, s in enumerate(series):
                s = s.copy()
                s.sort_index(inplace=True)
                if type(s.index) is not pd.tseries.index.DatetimeIndex:
                    dates = matplotlib.dates.date2num([dateutil.parser.parse(dt) fo
                else:
                    dates = s.index
                plt.plot_date(
                    dates,
```

```
                s.values,
                label[i]
            )
    if title:
        plt.title(title)
    if ylabel != '':
        plt.ylabel(ylabel)
    plt.xlabel(xlabel)
    plt.gcf().autofmt_xdate()
    if len(series) > 1:
        plt.legend([s.name for s in series], loc=2)
    plt.show()

def plot_scatter_correlation(df, target, standardize=True):
    df = df.copy()
    if standardize: df = normalize(df)
    ncols = 3
    nrows = math.ceil(len(df.columns)/ncols)
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, sharex=False, sharey
    fig.subplots_adjust(hspace=.3)
    for i,variable in enumerate(df.columns):
        row = math.floor(i / nrows)
        col = i % 3
        if nrows > 1:
            axis = axes[row,col]
        else:
            axis = axes[col]
        correlation = df[variable].corr(target)
        title = "corr = {:.5f}".format(correlation)
        sns.regplot(df[variable], target, ax=axis)
        axis.set_title(title)
    unused_axes = ncols * nrows - len(df.columns)
    for i in range(1, unused_axes+1):
        if nrows > 1:
            axis = axes[-1,-i]
        else:
            axis = axes[-i]
        fig.delaxes(axis)

def normalize(df_in):
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        return df_in.apply(preprocessing.scale)

def drop_unused_target_variables(df_in):
    df_out = df_in.copy()
    if target_var not in df_out.columns:
        df_out[target_var] = calculate_target_variable(df_out)
```

```
        return df_out[['dew_point','pressure', 'screen_relative_humidity','temp
```

2 3. Preliminary Analysis

2.1 3.1 Variables

```
In [3]: csv_files = ['./data/'+file for file in os.listdir('./data/') if '.csv' in
        df_imported = pd.concat([pd.read_csv(file, index_col=0) for file in csv_fil
        df_imported.index = pd.DatetimeIndex(df_imported.sort_index().index)
        grouped = df_imported.groupby(level=0)
        df_imported = grouped.last()
        df_imported.info(memory_usage=False)
```

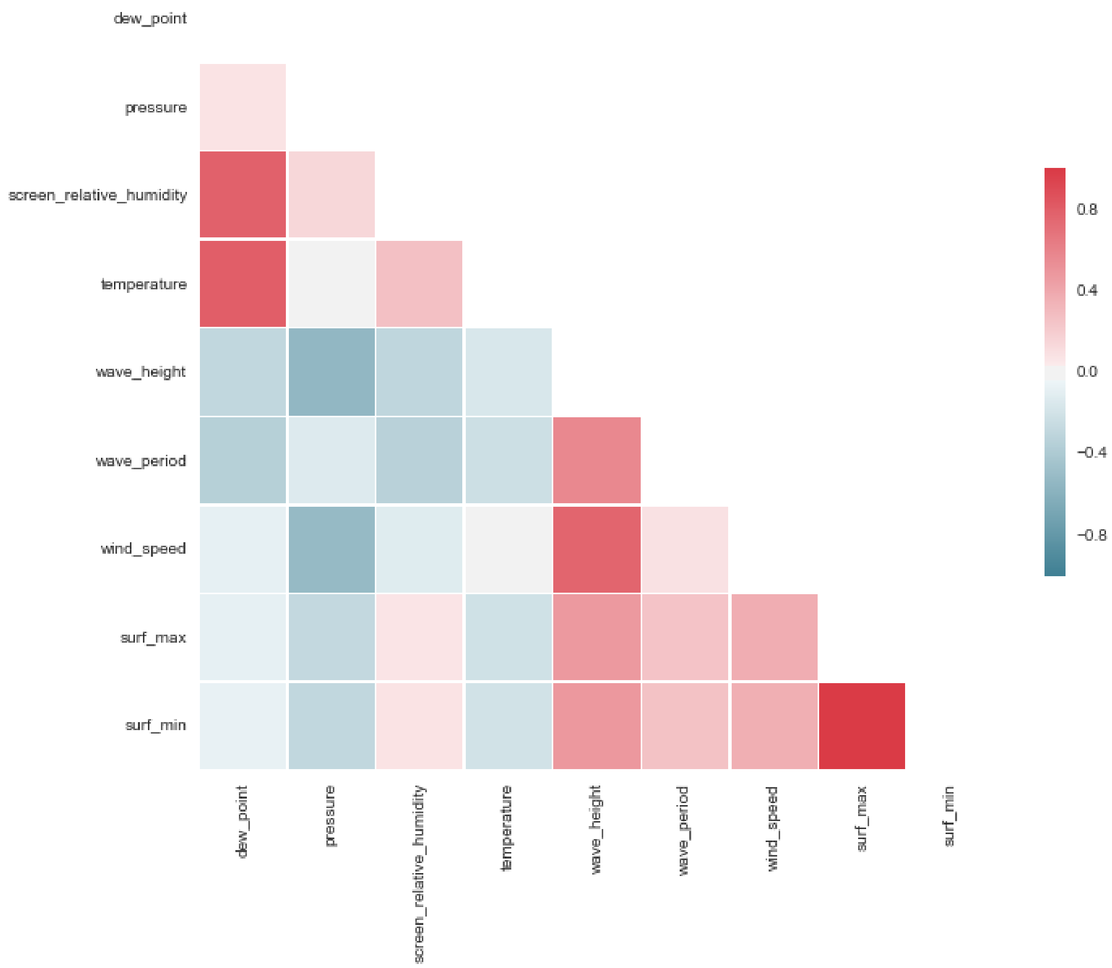
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 760 entries, 2016-05-17 20:00:00 to 2016-06-18 19:00:00
Data columns (total 25 columns):
$                           757 non-null float64
dateStamp                   128 non-null object
dew_point                   757 non-null float64
modelCode                   128 non-null object
modelRun                    128 non-null float64
periodSchedule              128 non-null float64
pressure                    757 non-null float64
screen_relative_humidity    757 non-null float64
sea_temperature             757 non-null float64
surf_max                    128 non-null float64
surf_min                    128 non-null float64
swell_direction1            128 non-null float64
swell_direction2            128 non-null float64
swell_direction3            128 non-null float64
swell_height1               128 non-null float64
swell_height2               128 non-null float64
swell_height3               128 non-null float64
swell_period1               128 non-null float64
swell_period2               128 non-null float64
swell_period3               128 non-null float64
temperature                 757 non-null float64
wave_height                 757 non-null float64
wave_period                 757 non-null float64
wind_direction              757 non-null object
wind_speed                  757 non-null float64
dtypes: float64(22), object(3)
```

2.2 3.2 Gauss Markov Assumptions

2.2.1 3.2.2 No Perfect Collinearity

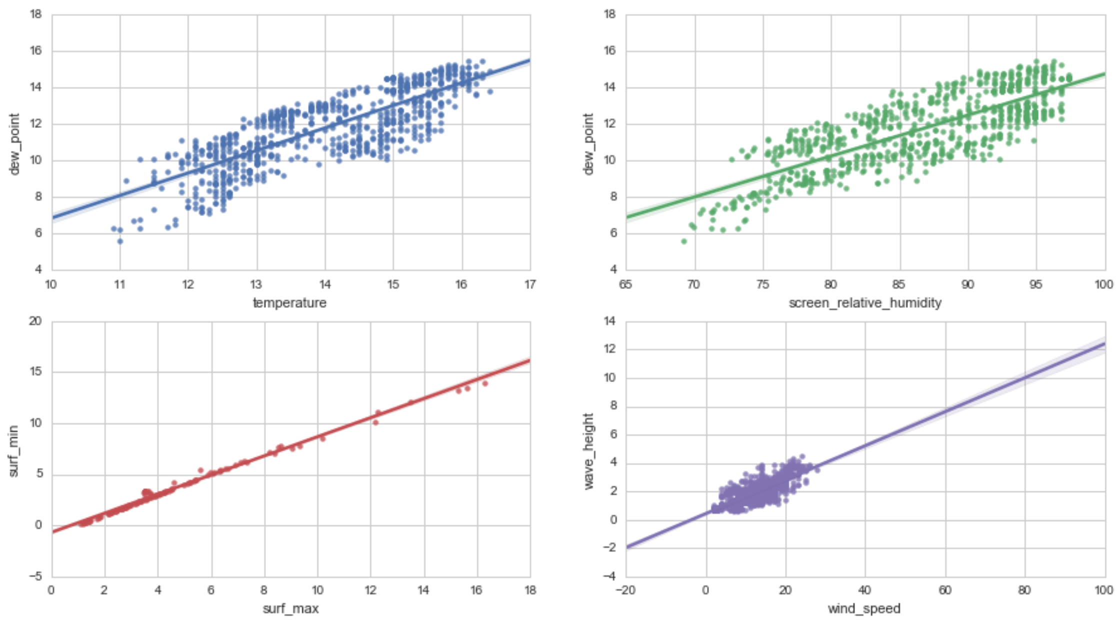
```
In [4]: df_retained = df_imported[['dew_point','pressure', 'screen_relative_humidit
        with sns.axes_style("white"):
```

```
mask = np.zeros_like(df_retained.corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
fig, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(df_retained.corr(), mask=mask, cmap=cmap, vmax=1,
            square=True,
            linewidths=.5, cbar_kws={"shrink": .5}, ax=ax)
```




```
In [5]: fig, ax = plt.subplots(2,2, figsize=(default_fig_width,4*2), sharex=False,
        sns.regplot(df_retained['temperature'],df_retained['dew_point'], ax=ax[0,0]
        sns.regplot(df_retained['screen_relative_humidity'],df_retained['dew_point'
        sns.regplot(df_retained['surf_max'],df_retained['surf_min'], ax=ax[1,0])
        sns.regplot(df_retained['wind_speed'], df_retained['wave_height'], ax=ax[1,
```

```
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x3b564b8a20>
```



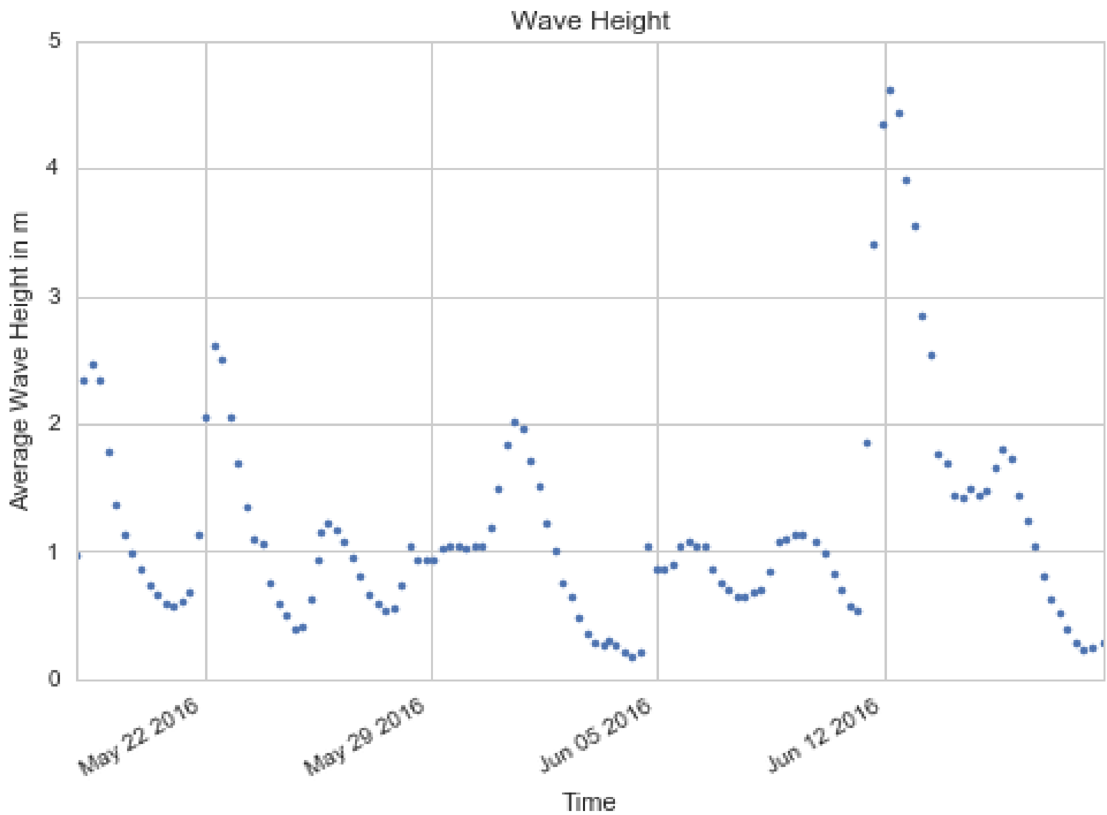
3 4. Transformation and Modelling

3.1 4.1 Target Variable

Calculate the average and convert feet to meters.

```
In [6]: target_var = 'surf_avg'
        def calculate_target_variable(df_in):
            return pd.Series( ((df_in['surf_max'] + df_in['surf_min']) / 2)*0.3048,
        s_target = calculate_target_variable(df_imported)
        print('available target values:', s_target.count())
        plot_datetime_series(s_target, 'Wave Height', 'Time', 'Average Wave Height
```

```
available target values: 128
```

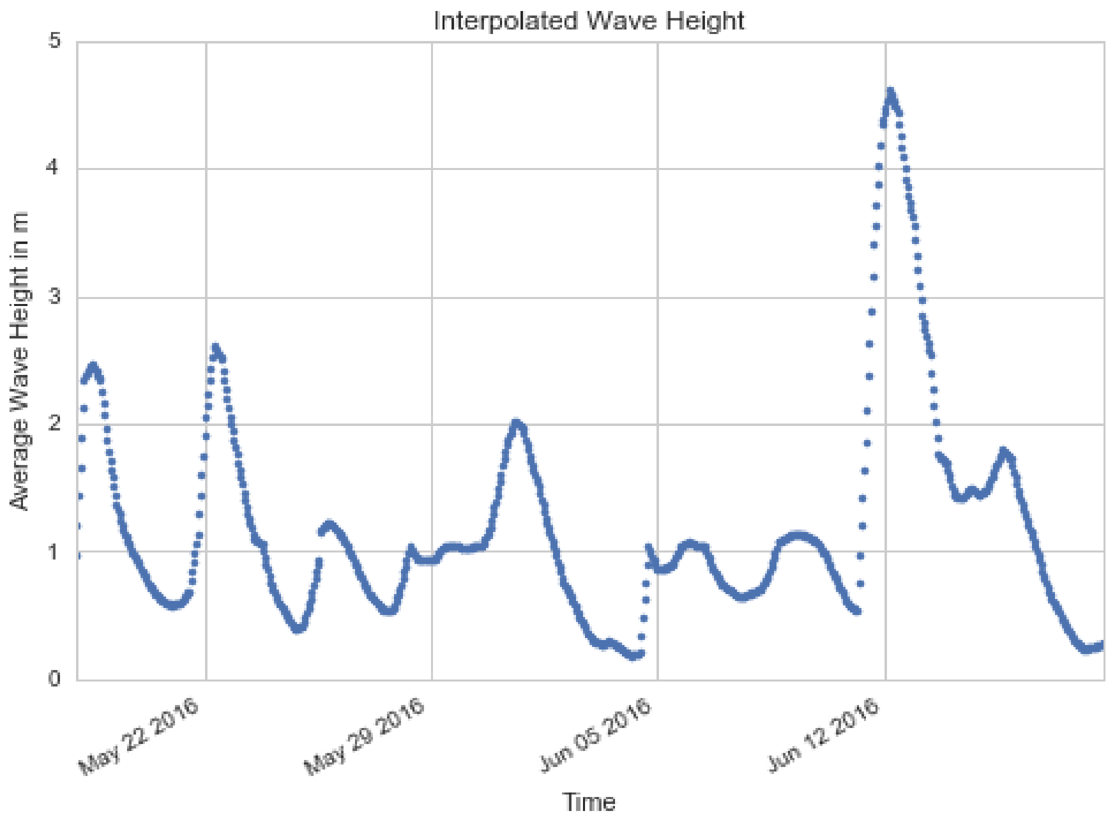


3.2 4.2 Interpolation

Wind direction is omitted since it is non-numerical.

```
In [7]: # Fill few missing values in explanatory data
        def interpolate_missing_values(df_in):
            return df_in.resample('H').interpolate().dropna()
        df_interpolated = pd.concat(
            [
                df_imported[['wave_height', 'wind_speed', 'wave_period', 'pressure'
                s_target
            ], axis=1)
        df_interpolated = interpolate_missing_values(df_interpolated)

        s_target = df_interpolated[target_var]
        print('target values after interpolation:', s_target.count())
        plot_datetime_series(s_target, 'Interpolated Wave Height', 'Time', 'Average
```

```
target values after interpolation: 763
```



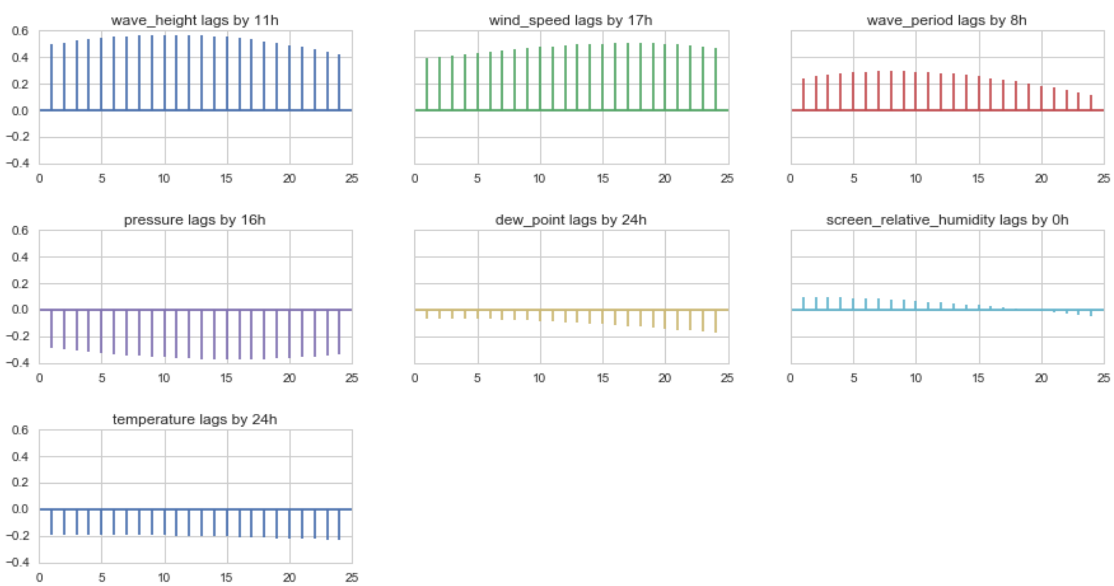
3.3 4.3 Determination of Lag

In order to find out the lag between target variable and each of the explanatory variables, a cross correlation for all lags between 0h and 24h is computed. Afterwards, the correlation of the best fitting lags is looked at more closely.

```
In [8]: def cross_correlation(s1, s2, minlag=0, maxlag=72):
            if((s1.index != s2.index).any()):
                print('Indexes differ!')
                return
            s_result = pd.Series(index=range(minlag, maxlag))
            for lag in range(minlag, maxlag+1):
                s_result[lag] = s1.shift(lag).corr(s2)
            return s_result

        def abs_max_at(s):
            max_index = s.index[0]
            max_value = s.values[0]
            for i, v in s.iteritems():
                if abs(v) > abs(max_value):
                    max_index = i
```

```
                max_value = v
        return max_index
```

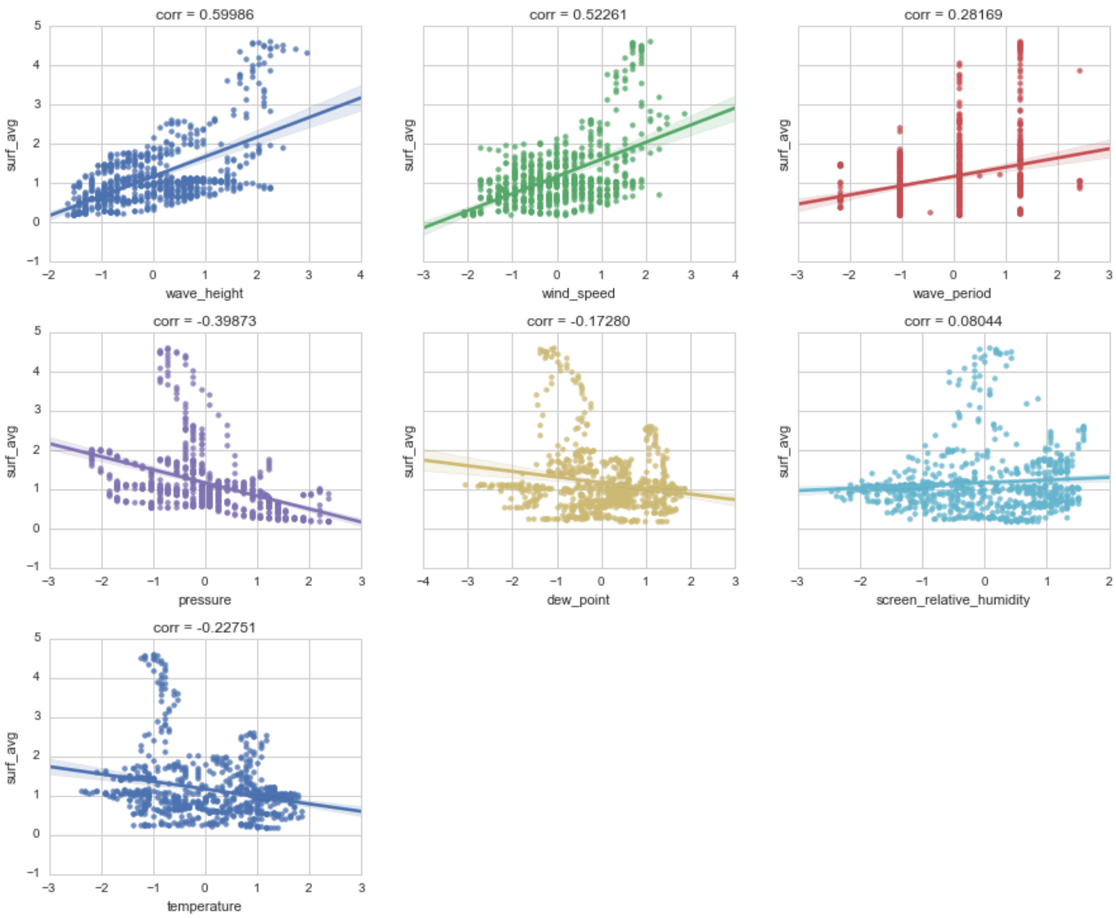
In [9]:
```
df_explanatory = df_interpolated.drop('surf_avg', axis=1)

# for each explanatory variable, plot lags and determine optimum
optimal_lags = pd.Series(index=df_explanatory.columns)
nrows = math.ceil(len(df_explanatory.columns) / 3)
fig, axes = plt.subplots(nrows=nrows, ncols=3, sharey=True, sharex=False, f
fig.subplots_adjust(hspace=.5)
for i,var in enumerate(df_explanatory.columns):
    # style: select column, row and color for chart
    row = math.floor(i / 3)
    col = i % 3
    if nrows == 1: axis = axes[col]
    else: axis = axes[row,col]
    if i < len(sns.color_palette()): color = sns.color_palette()[i]
    else: color = sns.color_palette()[i-len(sns.color_palette())]

    # calculate cross correlation and find optimum, save it for later use
    xcorr = cross_correlation(df_interpolated[var], df_interpolated[target_
    optimal_lags[var] = abs_max_at(xcorr)

    # plot all cross correlations
    axis.vlines(x=xcorr.index, ymin=0, ymax=xcorr.values, color=color)
    axis.axhline(color=color)
    axis.set_title(var + ' lags by ' + "{0:g}".format(optimal_lags[var]) +

# delete unused charts
fig.delaxes(axes[2,1])
fig.delaxes(axes[2,2])
```



The highest correlation seems to be at a lag of about -5 to -15, but is different for each variable. - Wind speed appears to have a very high lag, just like the wind direction. - Wave height and wave period both have a smaller lag.

```
In [10]: def lag(df_in, lags=optimal_lags):
             df_lagged = pd.DataFrame()
             for var, lag in lags.iteritems():
                 if var in df_in.columns:
                     df_lagged = pd.concat(
                         [df_lagged, df_in[var].shift(lag, freq='H', axis=0)],
                         axis=1
                     )
             for var in df_in.columns:
                 if var not in lags.index:
                     df_lagged[var] = df_in[var]
             return df_lagged.dropna()

         df_lagged = pd.concat([
                 lag(df_explanatory, optimal_lags),
                 s_target],
                 axis=1
         ).dropna()
         print('complete training rows left after lagging:', len(df_lagged), '(lost

         # update target variable and lose data not described by lagged independent
         s_target_lagged = df_lagged[target_var]

         plot_scatter_correlation(df_lagged.drop(target_var, axis=1), df_lagged[tar
```

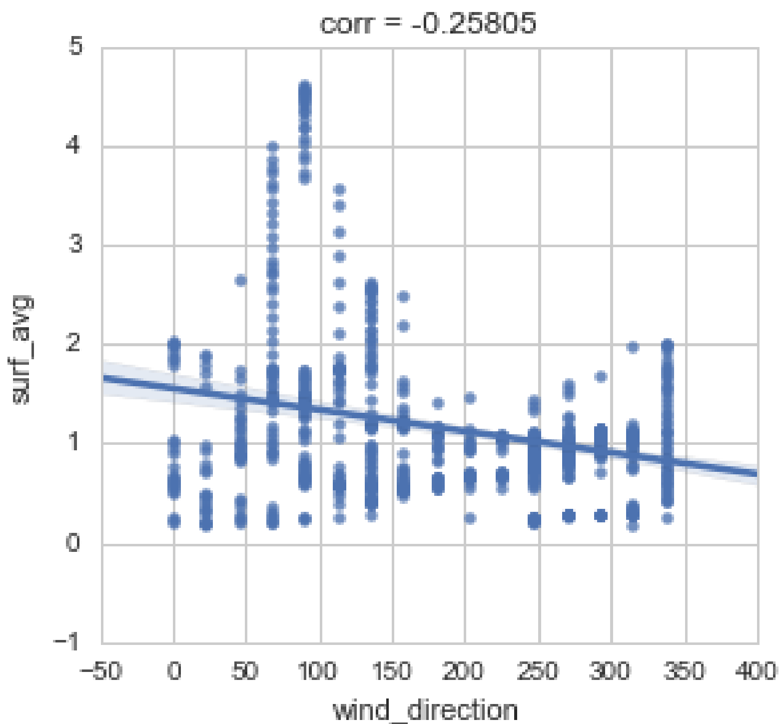
```
complete training rows left after lagging: 739 (lost: 24 )
```



3.4 4.4 Transforming Wind Direction

In [11]:
```
compass_directions = ['N', 'NNE', 'NE', 'ENE', 'E', 'ESE', 'SE', 'SSE', 'S
def compass_to_degrees(s_in):
    s_out = pd.Series(index=s_in.index, name=s_in.name)
    for i,dir in enumerate(s_in.dropna()):
        s_out[i] = (360 / len(compass_directions)) * compass_directions.in
    return s_out
s_wind_direction = compass_to_degrees(df_imported['wind_direction'])

# interpolation is dangerous! 1° to 360° will be interpolated with 180°!
# instead, forward fill to be sure
def fillna_wind_direction(s_in):
    return s_in.resample('H').fillna(method='ffill')
s_wind_direction = fillna_wind_direction(s_wind_direction)

def lag_wind_direction(s_in):
    # tie lag of wind_direction to wind_speed
```

```
        s_out = s_in.shift(optimal_lags['wind_speed'], freq='H', axis=0)
        # crop to fit on rest of data
        return s_out[df_lagged.index[0]:df_lagged.index[-1]]

    s_wind_direction = lag_wind_direction(s_wind_direction)

    plot_scatter_correlation(pd.DataFrame(s_wind_direction), s_target_lagged,
```



3.4.1 4.4.1 Sine Cosine Transformation

This example illustrates how 360° and 0° have a distance of 0 after sine cosine transformation.

```
In [12]: def degrees_to_sin_cos(df_in, columns):
             if(type(columns) is not list):
                 columns = [columns]
             df_radians = df_in[columns].apply(lambda deg: deg * (math.pi / 180))
             df_sin = df_radians.apply(lambda series: series.map(math.sin))
             df_cos = df_radians.apply(lambda series: series.map(math.cos))
             df_sin.columns = [col + '-Sin' for col in df_sin.columns]
             df_cos.columns = [col + '-Cos' for col in df_cos.columns]
             return pd.concat(
                 [
                     df_in.drop(columns, axis=1),
                     df_sin,
```

```
                    df_cos
                ],
                axis=1)

        circular_example = pd.DataFrame({'degrees':[361, 1]})

        display(circular_example)
        display(degrees_to_sin_cos(circular_example, 'degrees'))
```

```
   degrees
0      361
1        1
```

```
   degrees-Sin  degrees-Cos
0     0.017452     0.999848
1     0.017452     0.999848
```

In the following, the sine cosine transformation is applied to actual data.

```
In [13]: def transform_wind_direction_sin_cos(s_in):
             df_out = pd.DataFrame(s_in)
             df_out = degrees_to_sin_cos(df_out, 'wind_direction')
             return df_out
         display(df_imported[['wind_direction']].head())
         print('becomes')
         display(pd.DataFrame(s_wind_direction).head())
         print('becomes')
         df_wind_direction_sin_cos = transform_wind_direction_sin_cos(s_wind_direct
         display(df_wind_direction_sin_cos.head())
         plot_scatter_correlation(
             normalize(df_wind_direction_sin_cos),
             s_target_lagged
         )
```

```
                    wind_direction
2016-05-17 20:00:00             NE
2016-05-17 21:00:00             NE
2016-05-17 22:00:00            ENE
2016-05-17 23:00:00             NE
2016-05-18 00:00:00             NE
```

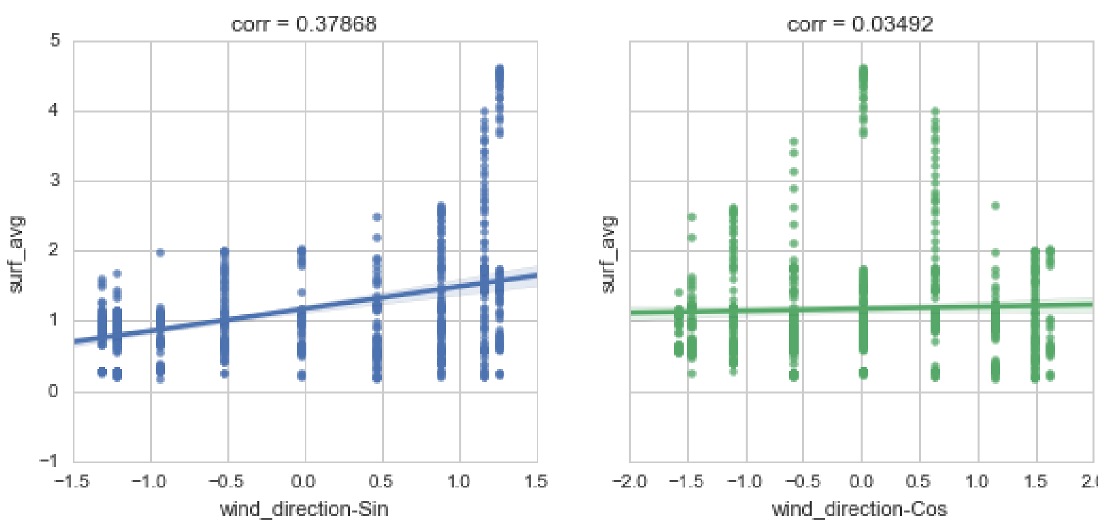
```
becomes
```

```
                     wind_direction
2016-05-19 01:00:00            22.5
```

```
2016-05-19 02:00:00           22.5
2016-05-19 03:00:00           45.0
2016-05-19 04:00:00           22.5
2016-05-19 05:00:00           45.0
```

```
becomes
```

```
                     wind_direction-Sin  wind_direction-Cos
2016-05-19 01:00:00            0.382683            0.923880
2016-05-19 02:00:00            0.382683            0.923880
2016-05-19 03:00:00            0.707107            0.707107
2016-05-19 04:00:00            0.382683            0.923880
2016-05-19 05:00:00            0.707107            0.707107
```



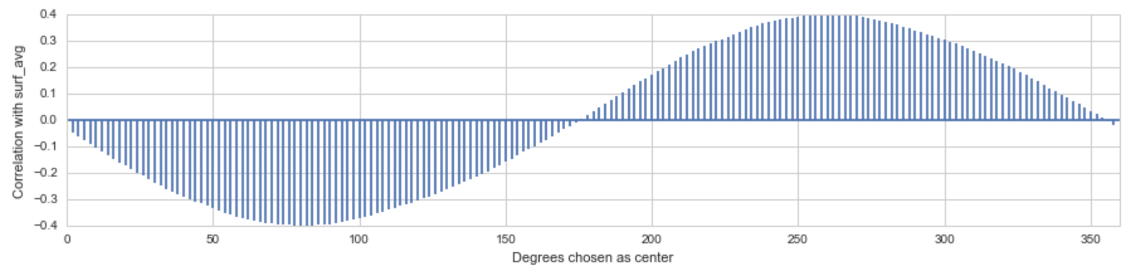
3.4.2 4.4.2 Degree Deviation Transformation

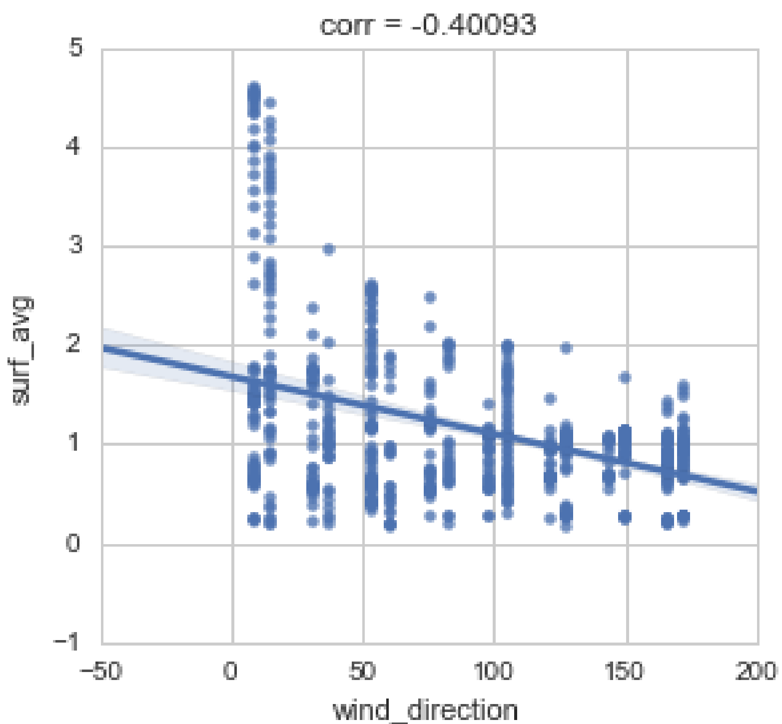
The previous approach gives a numerical view on wind direction where 360 and 0 degrees are equal. However, this doesn't result in a linear correlation, which is expected to exist for wind direction.

```
In [14]: def compute_degree_diff(degrees_base, i):
             diff = abs(degrees_base - i) % 360
             def under_180(alpha):
                 if(alpha > 180):
                     return 360 - alpha
                 else:
                     return alpha
```

```
        return diff.apply(under_180)

    degree_corr = pd.Series(index=range(0,360,2))
    for i in degree_corr.index:
        degree_corr[i] = compute_degree_diff(s_wind_direction, i).corr(s_targe

    def plot_degree_corr(degree_corr):
        fig, ax = plt.subplots(figsize=(math.floor(default_fig_width),3))
        ax.vlines(
            x=degree_corr.index,
            ymin=0,
            ymax=degree_corr.values,
            color=sns.color_palette()[0]
        )
        ax.axhline(color=sns.color_palette()[0])
        ax.set_xlabel('Degrees chosen as center')
        ax.set_ylabel('Correlation with surf_avg')
        ax.set_xbound(None, 360)

    plot_degree_corr(degree_corr)

    max_corr = degree_corr.max()
    max_corr_at = degree_corr[abs(degree_corr) == max_corr].index[0]
    print('Highest absolute correlation is {:.3f}'.format(max_corr) ,'at', max
    print('The inverse is at', 180+max_corr_at, 'degrees.')

    def transform_wind_direction(df_in, max_corr_at=max_corr_at):
        df_out = df_in.copy()
        df_out['wind_direction'] = compass_to_degrees(df_out['wind_direction']
        df_out['wind_direction'] = fillna_wind_direction(df_out['wind_directio
        df_out['wind_direction'] = lag_wind_direction(df_out['wind_direction']
        df_out['wind_direction'] = compute_degree_diff(df_out['wind_direction'
        return df_out.dropna()

    df_transformed = pd.concat([df_lagged, df_imported['wind_direction']], axi
    df_transformed = transform_wind_direction(df_transformed)


    plot_scatter_correlation(df_transformed[['wind_direction']], df_transforme
```

```
Highest absolute correlation is 0.400 at 82 degrees.
The inverse is at 262 degrees.
```





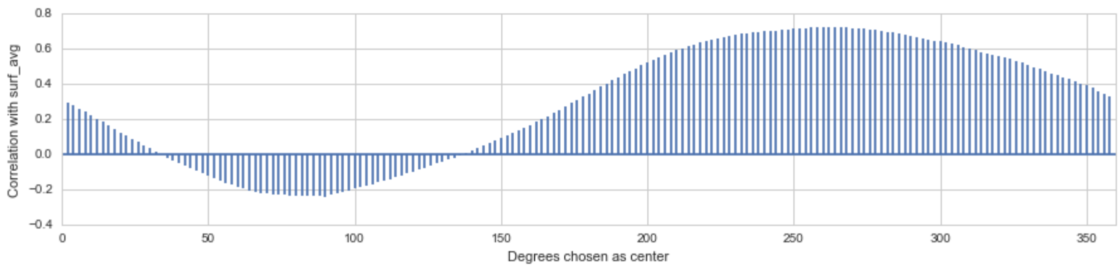
3.4.3 4.4.3 Wind Coefficient

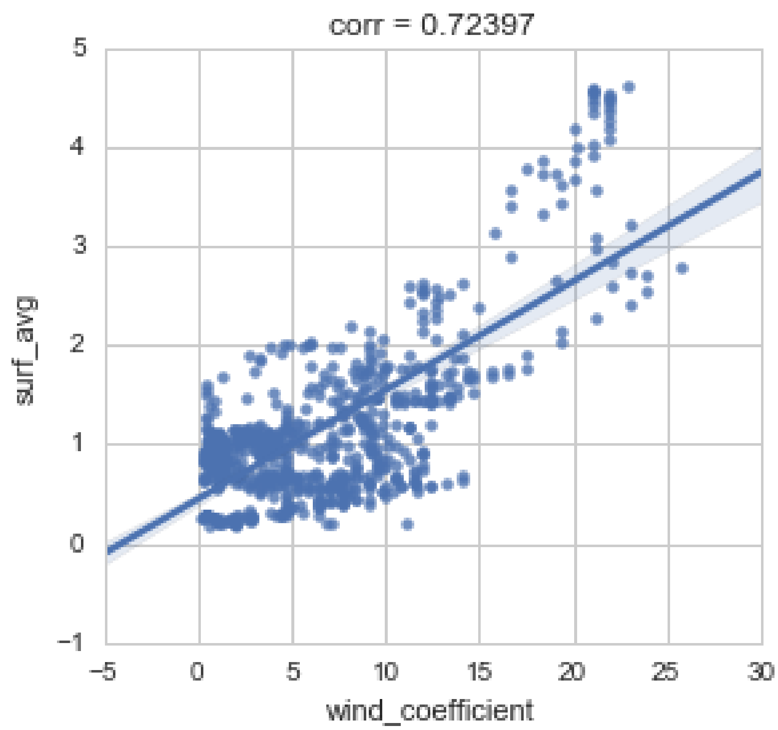
Unite wind direction and wind speed into a single value.

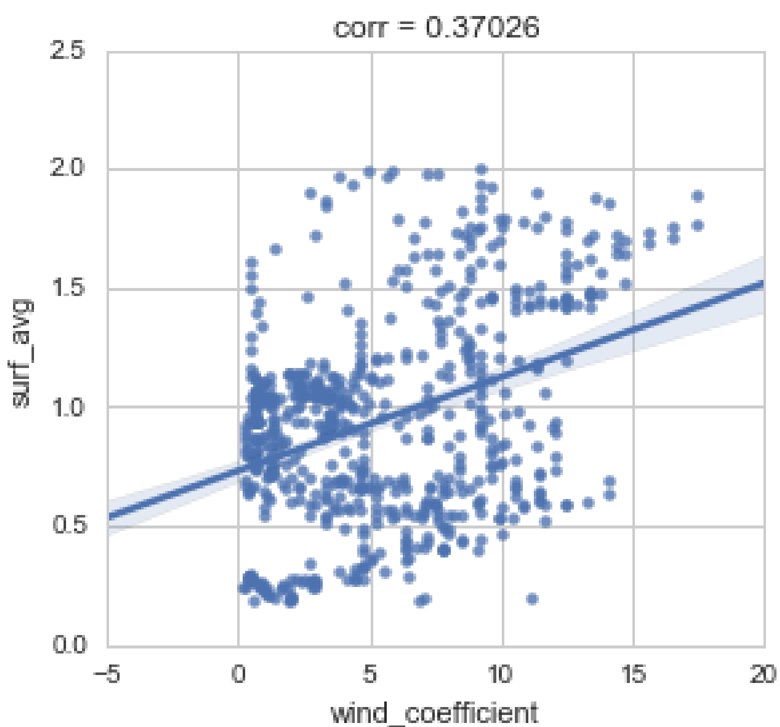
```
In [15]: def get_wind_coefficient(s_dir, s_speed):
             return (s_dir / 180) * s_speed

         coefficient_degree_corr = pd.Series(index=range(0,360,2))
         for i in coefficient_degree_corr.index:
             s_dir = compute_degree_diff(s_wind_direction, i)
             coeff = get_wind_coefficient(s_dir, df_transformed['wind_speed'])
             coefficient_degree_corr[i] = coeff.corr(s_target)
```

```
plot_degree_corr(coefficient_degree_corr)

max_corr = coefficient_degree_corr.max()
max_corr_at = coefficient_degree_corr[abs(coefficient_degree_corr) == max_
print('Highest absolute correlation is {:.3f}'.format(max_corr) ,'at', max

def add_wind_coefficient(df_in):
    df_out = df_in.copy()
    df_out['wind_coefficient'] = get_wind_coefficient(
        compute_degree_diff(s_wind_direction, max_corr_at),
        df_in['wind_speed']
    )
    return df_out

df_transformed = add_wind_coefficient(df_transformed)

plot_scatter_correlation(df_transformed[['wind_coefficient']], df_transfor

# hide outliers
mask_outliers = [(h <= 2) for h in df_transformed[target_var]]
plot_scatter_correlation(df_transformed[['wind_coefficient']][mask_outlier
```

```
Highest absolute correlation is 0.724 at 262 degrees.
```



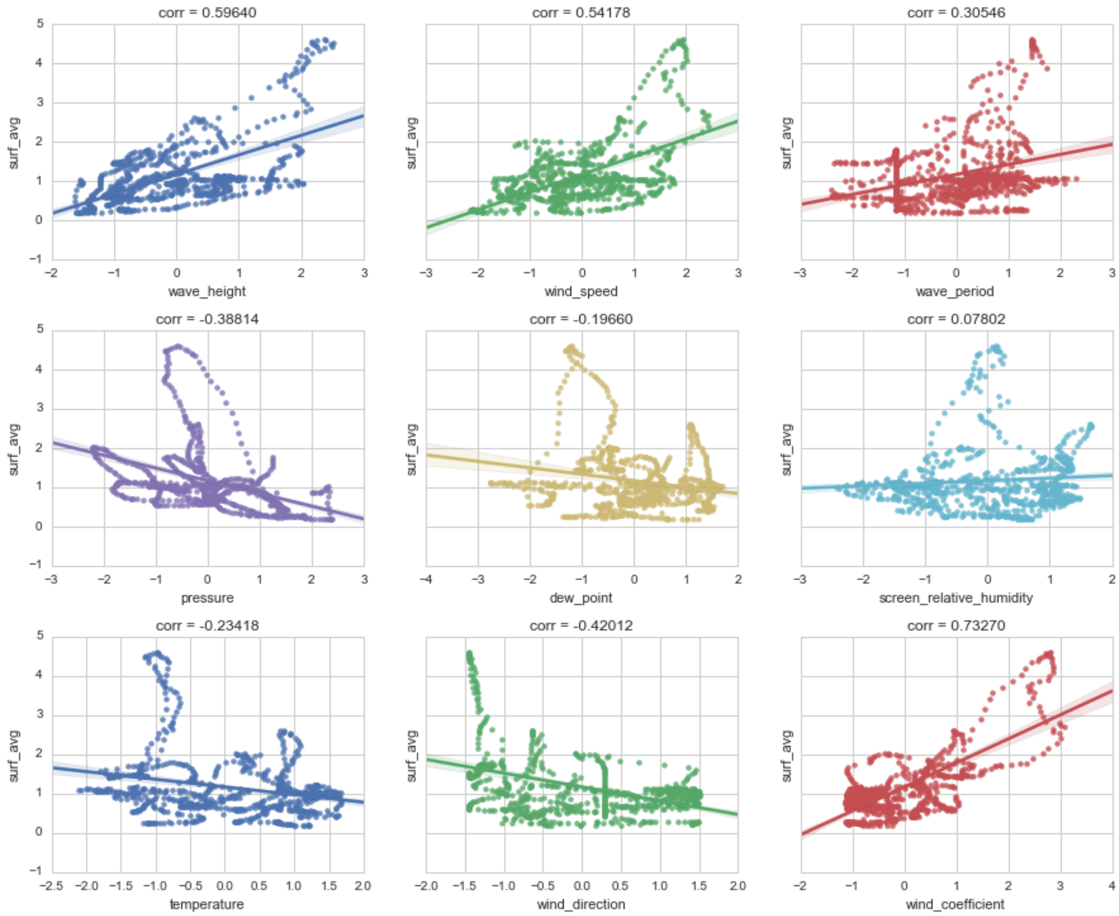

corr = 0.72397
surf_avg
wind_coefficient
5
4
3
2
1
0
−1
−5
0
5
10
15
20
25
30


corr = 0.37026
surf_avg
wind_coefficient
2.5
2.0
1.5
1.0
0.5
0.0
−5
0
5
10
15
20

3.5 4.5 Moving Averages

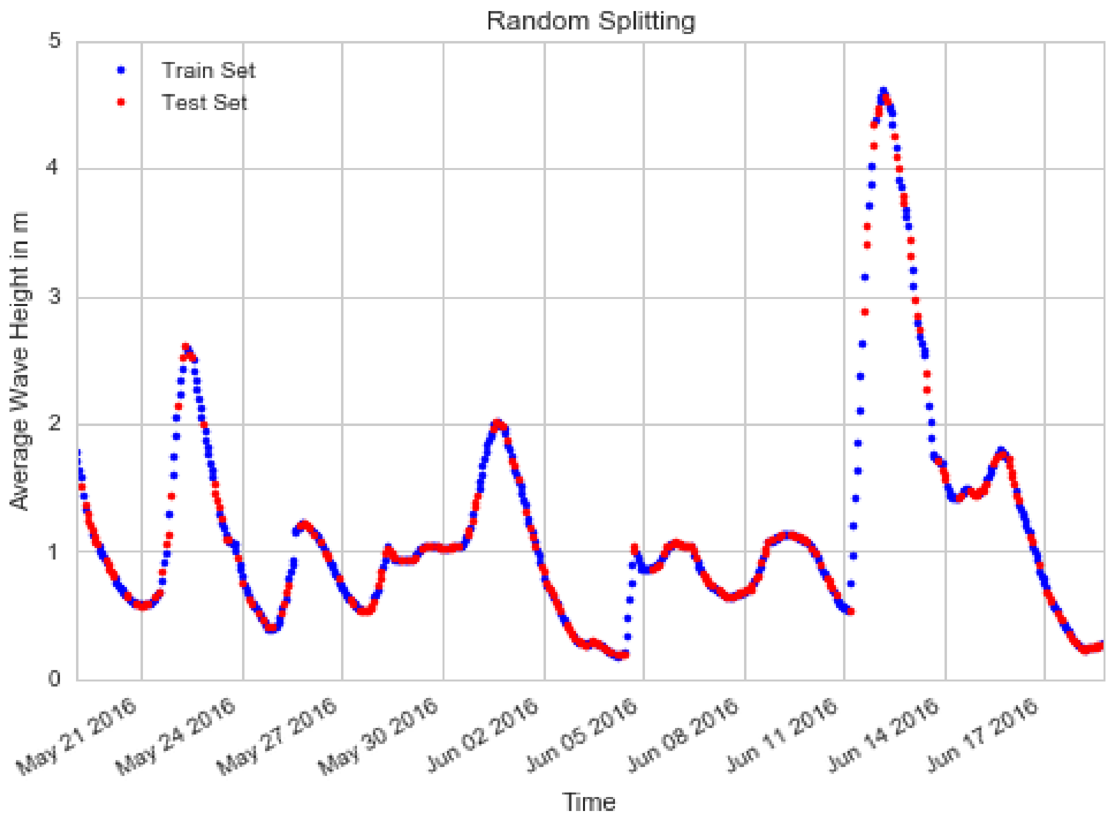
Roll back the lag slightly and apply a rolling mean to smooth out predictions.

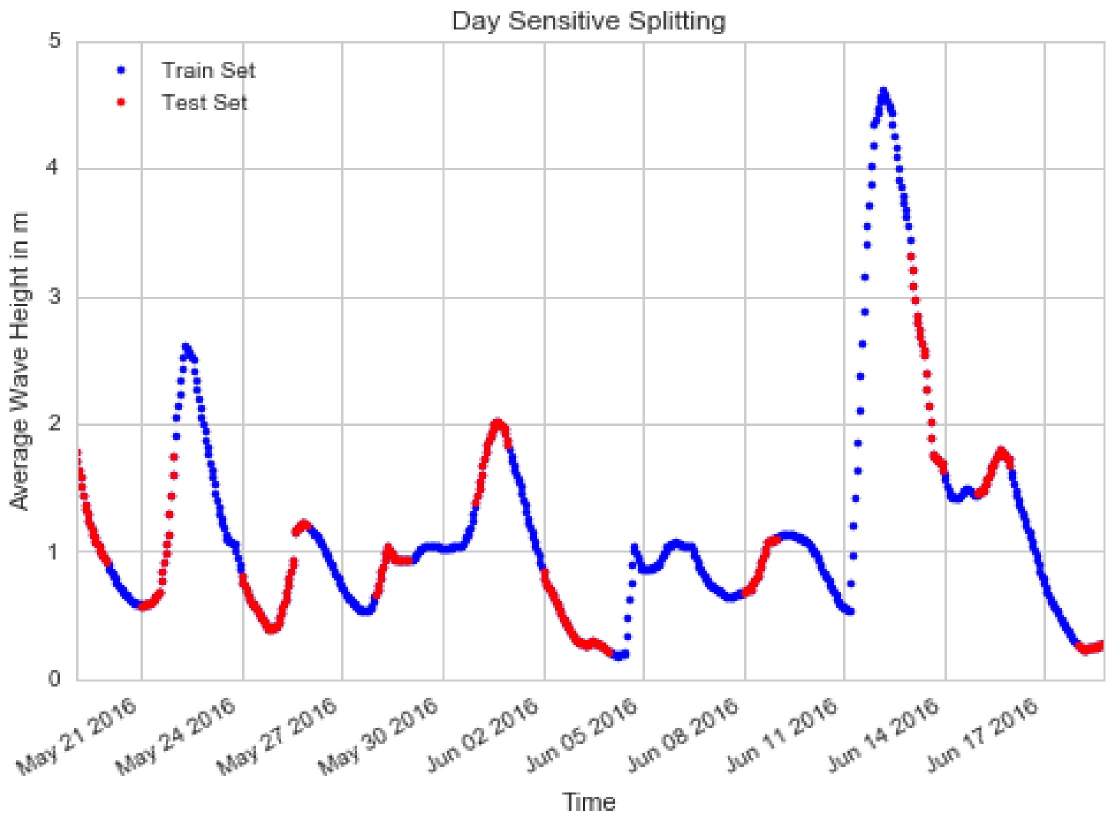
```
In [25]: def moving_averages(df_in, window=12, unshift=0, method='ewma'):
             df_out = df_in.copy()
             for column in df_out.columns:
                 df_out[column] = df_out[column].shift(-unshift, freq='H', axis=0)
                 if method is 'ma':
                     df_out[column] = pd.rolling_mean(df_out[column], window, min_p
                 if method is 'ewma':
                     df_out[column] = pd.ewma(df_out[column], span=window)
             return df_out

         df_ma = moving_averages(df_transformed.copy().drop(target_var, axis=1), 6,
         df_ma[target_var] = s_target
         plot_scatter_correlation(df_ma.drop(target_var,axis=1), df_ma[target_var])
```



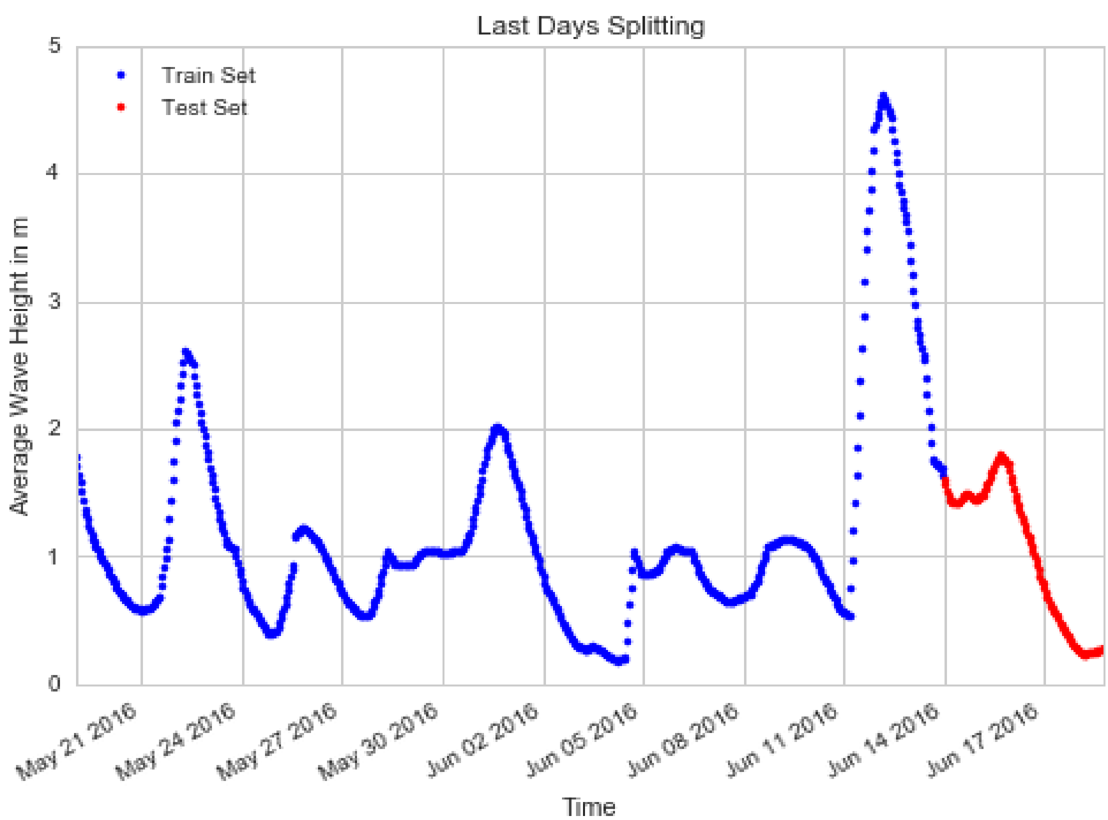
3.6 4.6 Validation

In [17]:
```
# randomly sample a train and test set
def split_dataset(df_in, percent_test):
    total_size = len(df_in)
    test_set_size = math.floor(percent_test * total_size)
    df_train = df_in.copy()
    record_labels = [ df_train.index[randint] for randint in random.sample
    df_test = pd.DataFrame(df_train.loc[record_labels])
    df_train = df_train.drop(record_labels)
    return df_train, df_test

def split_dataset_day_sensitive(df_in, percent_test):
    days = pd.Series(df_in.index).map(pd.Timestamp.date).unique()
    total_size = len(days)
    test_set_size = math.floor(percent_test * total_size)
    df_train = df_in.copy()
    record_labels = [ str(days[randint]) for randint in random.sample(rang
    df_test = pd.concat([df_train[l] for l in record_labels], axis=0)
    for l in record_labels:
        df_train.drop(df_train[l].index)
    return df_train, df_test

def split_dataset_last_days(df_in, days_count):
    days = pd.Series(df_in.index).map(pd.Timestamp.date).unique()
    record_labels = [ str(days[i]) for i in range(-days_count-1, 0) ]
    df_train = df_in.copy()
    df_test = pd.concat([df_train[l] for l in record_labels], axis=0)
    for l in record_labels:
        df_train = df_train.drop(df_train[l].index)
    return df_train, df_test

train_set, test_set = split_dataset(df_lagged, 0.4)
train_set[target_var].name = 'Train Set'
test_set[target_var].name = 'Test Set'
plot_datetime_series([train_set[target_var], test_set[target_var]], label=

train_set, test_set = split_dataset_day_sensitive(df_lagged, 0.4)
train_set[target_var].name = 'Train Set'
test_set[target_var].name = 'Test Set'
plot_datetime_series([train_set[target_var], test_set[target_var]], label=

train_set, test_set = split_dataset_last_days(df_lagged, 4)
train_set[target_var].name = 'Train Set'
test_set[target_var].name = 'Test Set'
plot_datetime_series([train_set[target_var], test_set[target_var]], label=
```


Random Splitting
Train Set
Test Set
Average Wave Height in m
0
1
2
3
4
5
May 21 2016
May 24 2016
May 27 2016
May 30 2016
Jun 02 2016
Jun 05 2016
Jun 08 2016
Jun 11 2016
Jun 14 2016
Jun 17 2016
Time



Day Sensitive Splitting
Train Set
Test Set
Average Wave Height in m
0
1
2
3
4
5
May 21 2016
May 24 2016
May 27 2016
May 30 2016
Jun 02 2016
Jun 05 2016
Jun 08 2016
Jun 11 2016
Jun 14 2016
Jun 17 2016
Time



3.7 4.7 Fitting the Model

```
In [18]: def preprocess(df_in, drop_unused=False, ma=False):
             explanatory_import_columns = ['wave_height', 'wave_period', 'wind_spee
             df_out = df_in.copy()[explanatory_import_columns]
             if(drop_unused):
                 df_out = df_out[['wave_height', 'wind_speed', 'wind_direction']]
             preprocessing_functions = [
                 'lag',
                 'transform_wind_direction',
                 'interpolate_missing_values',
                 'add_wind_coefficient',
                 'normalize'
             ]
             if ma:
                 preprocessing_functions.append('moving_averages')
             for fn in preprocessing_functions:
                 df_out = globals()[fn](df_out)
             if(drop_unused):
                 df_out = df_out[['wave_height', 'wind_coefficient']]
             return df_out.dropna()
```

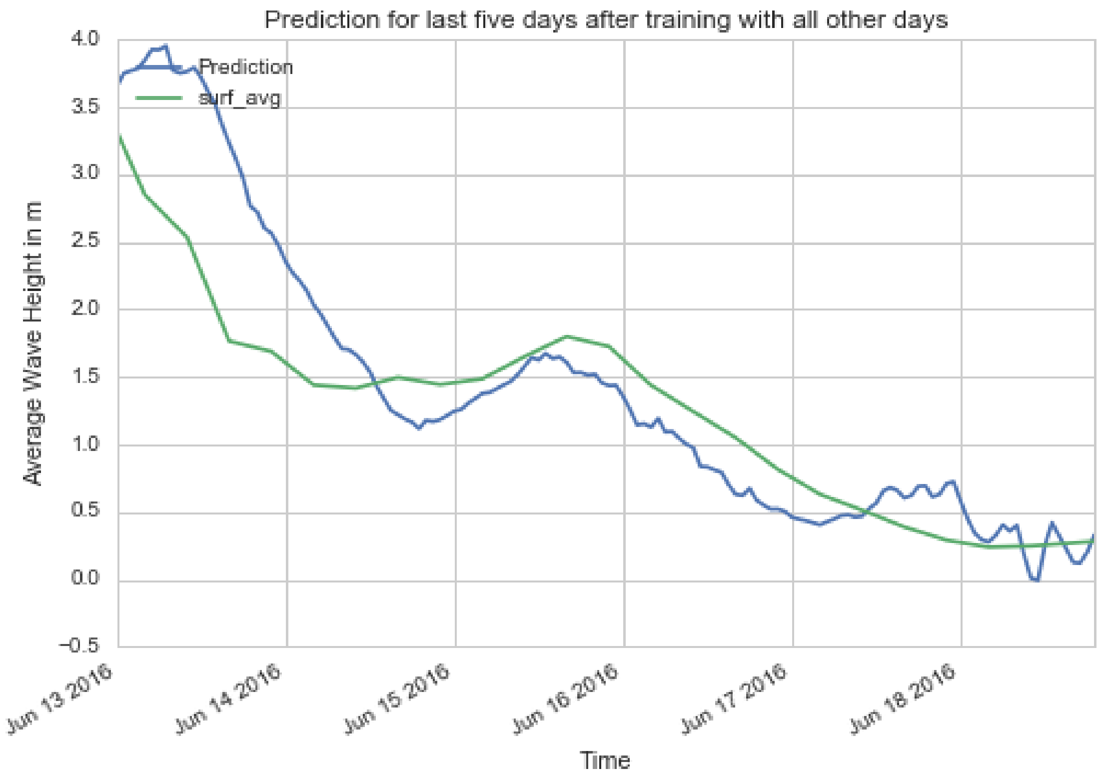
```python
def align_data(df_in, s_in):
    merged = pd.concat([df_in, s_in], axis=1).dropna()
    df_out = merged[df_in.columns]
    s_out = merged[s_in.name]
    return df_out, s_out
def train_linear_model(train_set, target=None):
    train_set = train_set.copy()
    if target is None:
        target = train_set[target_var]
        train_set = train_set.drop(target_var, axis=1)
    else:
        # make sure data is aligned
        train_set, target = align_data(train_set, target)
    model = linear_model.LinearRegression()
    model.fit(
        train_set.as_matrix(),
        target.as_matrix()
    )
    ols = sm.OLS(target, train_set).fit()
    display(ols.summary())
    return model

def predict(test_set, model):
    test_set = test_set.copy()
    if target_var in test_set.columns:
        test_set = test_set.drop(target_var, axis=1)
    return pd.Series(
        model.predict(test_set.as_matrix()),
        name='Prediction',
        index=test_set.index
    )
```

3.7.1 4.7.1 Entire Feature Set

In [19]:
```python
df_train = pd.concat(
    [
        preprocess(df_imported, ma=True, drop_unused=False),
        s_target
    ],
    axis=1
).dropna()

train_set, test_set = split_dataset_last_days(df_train, 5)
model = train_linear_model(train_set)
plot_datetime_series(
    [
        predict(test_set, model),
```

```
                 test_set[target_var]
             ],
             title= 'Prediction for last five days after training with all other da
         )
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:               surf_avg   R-squared:                       0.425
Model:                            OLS   Adj. R-squared:                  0.416
Method:                 Least Squares   F-statistic:                     47.55
Date:                Mon, 20 Jun 2016   Prob (F-statistic):           5.23e-64
Time:                        00:39:45   Log-Likelihood:                -880.49
No. Observations:                 587   AIC:                             1779.
Df Residuals:                     578   BIC:                             1818.
Df Model:                           9
Covariance Type:            nonrobust
=================================================================================
                              coef    std err          t      P>|t|      [95.0% Co
---------------------------------------------------------------------------------
wave_height                 1.9710      0.207      9.538      0.000         1.565
wind_speed                 -1.6388      0.178     -9.213      0.000        -1.988
wave_period                -0.6432      0.137     -4.695      0.000        -0.912
pressure                    0.1647      0.059      2.810      0.005         0.050
dew_point                  -0.0705      0.114     -0.620      0.535        -0.294
screen_relative_humidity   -0.0337      0.063     -0.532      0.595        -0.158
temperature                 0.6120      0.109      5.628      0.000         0.398
wind_direction              1.4844      0.210      7.074      0.000         1.072
wind_coefficient            2.0071      0.231      8.679      0.000         1.553
==============================================================================
Omnibus:                        5.747   Durbin-Watson:                   0.014
Prob(Omnibus):                  0.057   Jarque-Bera (JB):                4.037
Skew:                           0.031   Prob(JB):                        0.133
Kurtosis:                       2.599   Cond. No.                         13.0
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
"""
```



3.7.2 4.7.2 No Moving Averages

```
In [20]: df_train = pd.concat(
             [
                 preprocess(df_imported, ma=False, drop_unused=False),
                 s_target
             ],
             axis=1
         ).dropna()

         train_set, test_set = split_dataset_last_days(df_train, 5)
         model = train_linear_model(train_set)
         plot_datetime_series(
             [
                 predict(test_set, model),
                 test_set[target_var]
             ],
             title= 'Prediction for last five days after training with all other da
         )

<class 'statsmodels.iolib.summary.Summary'>
"""
```

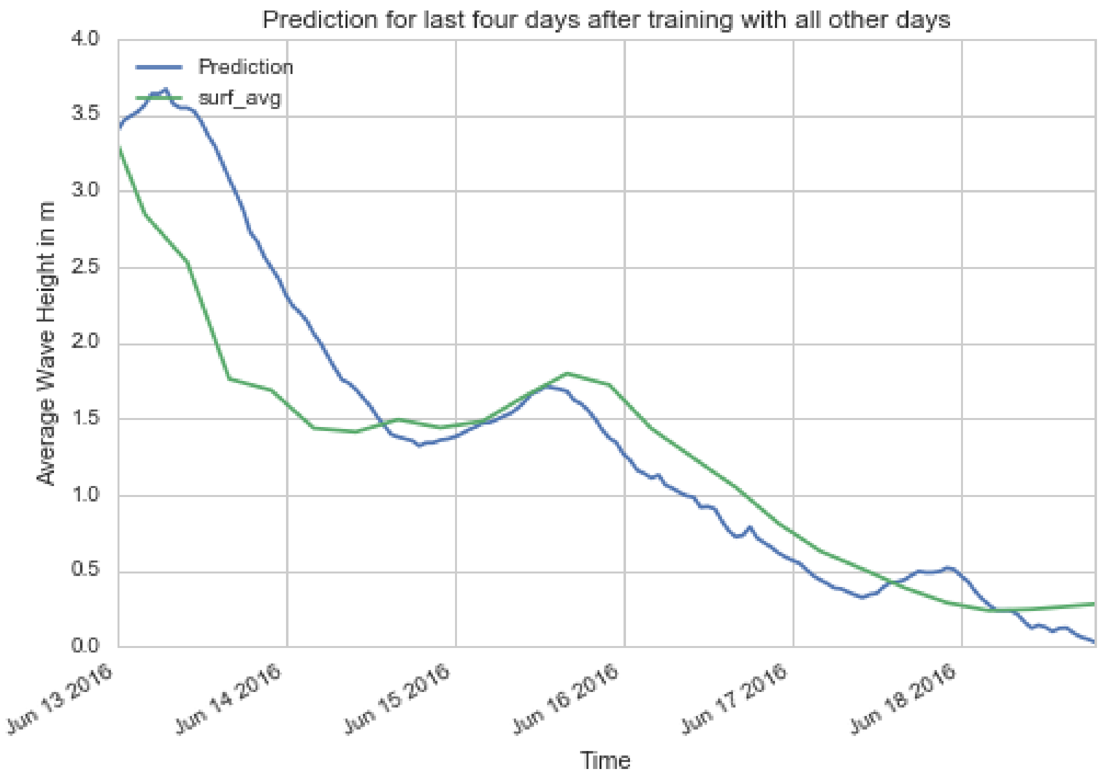
```
                            OLS Regression Results
==============================================================================
Dep. Variable:               surf_avg   R-squared:                       0.383
Model:                            OLS   Adj. R-squared:                  0.374
Method:                 Least Squares   F-statistic:                     39.94
Date:                Mon, 20 Jun 2016   Prob (F-statistic):           2.57e-55
Time:                        00:39:45   Log-Likelihood:                -901.18
No. Observations:                 587   AIC:                             1820.
Df Residuals:                     578   BIC:                             1860.
Df Model:                           9
Covariance Type:            nonrobust
================================================================================
                             coef    std err          t      P>|t|      [95.0% Co
--------------------------------------------------------------------------------
wave_height                1.0127      0.113      8.957      0.000         0.791
wind_speed                -0.6323      0.104     -6.058      0.000        -0.837
wave_period               -0.0523      0.074     -0.707      0.480        -0.198
pressure                   0.0544      0.056      0.972      0.331        -0.056
dew_point                 -0.1076      0.089     -1.207      0.228        -0.283
screen_relative_humidity   0.0115      0.053      0.216      0.829        -0.093
temperature                0.4631      0.090      5.137      0.000         0.286
wind_direction             0.6329      0.127      4.977      0.000         0.383
wind_coefficient           1.0423      0.143      7.294      0.000         0.762
==============================================================================
Omnibus:                        5.691   Durbin-Watson:                   0.105
Prob(Omnibus):                  0.058   Jarque-Bera (JB):                7.679
Skew:                           0.005   Prob(JB):                       0.0215
Kurtosis:                       3.560   Cond. No.                         7.70
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
"""
```

3.7.3 4.7.3 Features based on Problem Knowledge

```
In [21]: df_train = pd.concat(
             [
                 preprocess(df_imported, ma=True, drop_unused=True),
                 s_target
             ],
             axis=1
         ).dropna()

         train_set, test_set = split_dataset_last_days(df_train, 5)
         model = train_linear_model(train_set)
         plot_datetime_series(
             [
                 predict(test_set, model),
                 test_set[target_var]
             ],
             title= 'Prediction for last four days after training with all other da
         )
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               surf_avg   R-squared:                       0.246
Model:                            OLS   Adj. R-squared:                  0.243
Method:             Least Squares   F-statistic:                     96.42
Date:                Mon, 20 Jun 2016   Prob (F-statistic):           5.67e-37
Time:                        00:39:46   Log-Likelihood:                -971.19
No. Observations:                 594   AIC:                             1946.
Df Residuals:                     592   BIC:                             1955.
Df Model:                           2
Covariance Type:            nonrobust
==================================================================================
                       coef    std err          t      P>|t|      [95.0% Conf. Int.
----------------------------------------------------------------------------------
wave_height          0.6746      0.059     11.395      0.000         0.558     0.79
wind_coefficient     0.3149      0.062      5.085      0.000         0.193     0.43
==============================================================================
Omnibus:                        0.902   Durbin-Watson:                   0.002
Prob(Omnibus):                  0.637   Jarque-Bera (JB):                0.891
Skew:                           0.094   Prob(JB):                        0.640
Kurtosis:                       2.981   Cond. No.                         1.27
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
"""
```

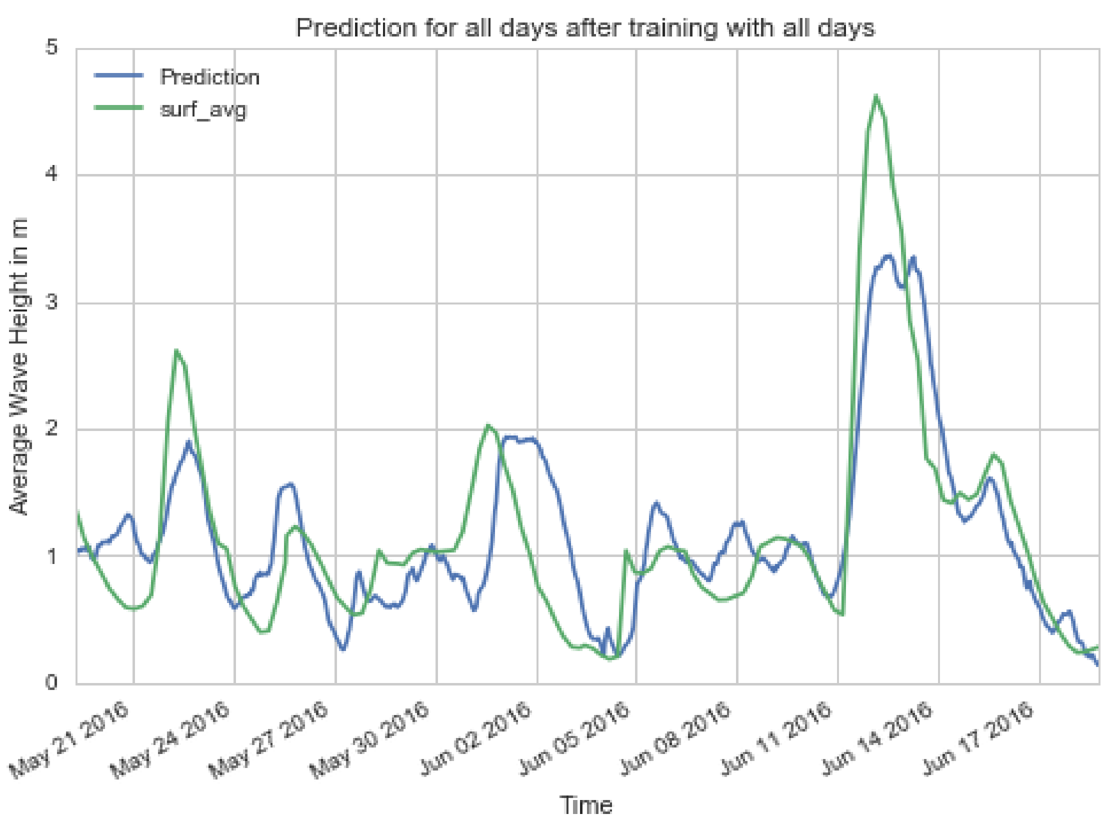


3.7.4 4.7.4 Final Model

```
In [22]: train_set, target = align_data(
             preprocess(df_imported, ma=True, drop_unused=True),
             s_target
         )
         final_model = train_linear_model(train_set, target)
         plot_datetime_series(
             [
                 predict(train_set, final_model),
                 target
             ],
             title= 'Prediction for all days after training with all days'
         )
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:               surf_avg   R-squared:                       0.235
Model:                            OLS   Adj. R-squared:                  0.233
Method:                 Least Squares   F-statistic:                     112.6
```

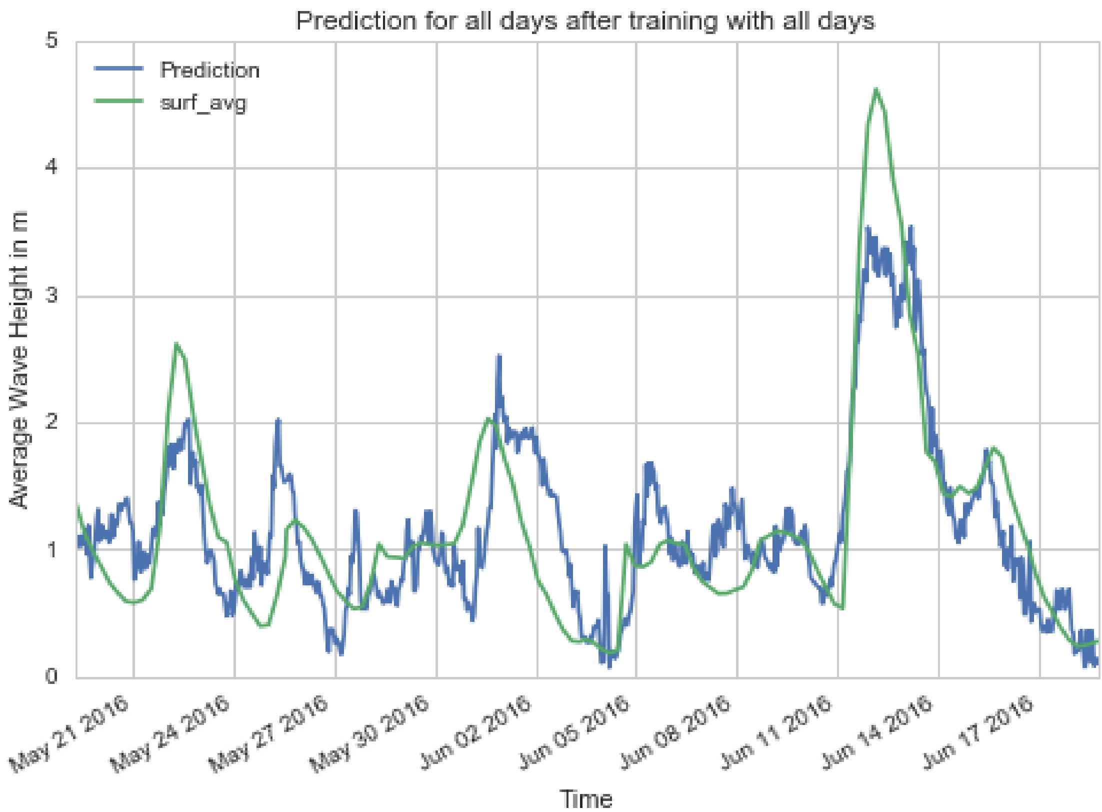
```
Date:                    Mon, 20 Jun 2016   Prob (F-statistic):          2.29e-43
Time:                            00:39:46   Log-Likelihood:               -1207.8
No. Observations:                     734   AIC:                            2420.
Df Residuals:                         732   BIC:                            2429.
Df Model:                               2
Covariance Type:                nonrobust
==============================================================================
                       coef    std err          t      P>|t|      [95.0% Conf. Int.
------------------------------------------------------------------------------
wave_height          0.3693      0.052      7.102      0.000         0.267      0.47
wind_coefficient     0.5475      0.051     10.652      0.000         0.447      0.64
==============================================================================
Omnibus:                       17.090   Durbin-Watson:                   0.002
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               22.441
Skew:                           0.250   Prob(JB):                     1.34e-05
Kurtosis:                       3.695   Cond. No.                         1.34
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
"""
```



Final Model without Moving Averages

```
In [23]: train_set, target = align_data(
             preprocess(df_imported, ma=False, drop_unused=True),
             s_target
         )
         final_model = train_linear_model(train_set, target)
         plot_datetime_series(
             [
                 predict(train_set, final_model),
                 target
             ],
             title= 'Prediction for all days after training with all days'
         )
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:               surf_avg   R-squared:                       0.235
Model:                            OLS   Adj. R-squared:                  0.233
Method:                 Least Squares   F-statistic:                     112.6
Date:                Mon, 20 Jun 2016   Prob (F-statistic):           2.32e-43
Time:                        00:39:46   Log-Likelihood:                -1207.8
No. Observations:                 734   AIC:                             2420.
Df Residuals:                     732   BIC:                             2429.
Df Model:                           2
Covariance Type:            nonrobust
================================================================================
                      coef    std err          t      P>|t|      [95.0% Conf. Int.
--------------------------------------------------------------------------------
wave_height         0.3570      0.049      7.356      0.000         0.262      0.45
wind_coefficient    0.5007      0.049     10.317      0.000         0.405      0.59
==============================================================================
Omnibus:                        1.670   Durbin-Watson:                   0.014
Prob(Omnibus):                  0.434   Jarque-Bera (JB):                1.509
Skew:                           0.097   Prob(JB):                        0.470
Kurtosis:                       3.106   Cond. No.                         1.36
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
"""
```



Prediction for all days after training with all days
Prediction
surf_avg
Average Wave Height in m
5
4
3
2
1
0
May 21 2016
May 24 2016
May 27 2016
May 30 2016
Jun 02 2016
Jun 05 2016
Jun 08 2016
Jun 11 2016
Jun 14 2016
Jun 17 2016
Time