Predicting Health/ Fitness goals using consumer wearable sensing devices

by Devasena Inupakutika

In [474]:

```
#Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
% matplotlib inline
import seaborn as sns
from datetime import timedelta
import csv
import sys, os
from collections import deque
```

Analysis of Fitbit Wearable Device Data written to csv file

In [475]:

```
#Read the date-wise data from 5RJHGY_Fitbit.csv file
df = pd.read_csv('Fitbit-5RJHGY-data.csv')
pd.to_datetime(pd.Series(df['Date']), format="%Y-%m-%d")
df.head()
```

Out[475]:

	Date	Calories Burned	Distance	Floors	Minutes Sedentary	Minutes Lightly Active	Minutes Fairly Active	Minutes Very Active	Activity Calories	Sleep Start Time	Sleep End Time	Minutes Asleep	Minutes Awake	A
0	2017-05-30	1681	2.15	0	1311	117	3	9	467	NaN	NaN	NaN	NaN	N
1	2017-05-31	1892	3.29	0	1198	226	7	1	779	2017-05-31 11:51PM	2017-06-01 6:04AM	368.0	4.0	3.
2	2017-06-01	1986	3.92	0	810	218	25	23	903	NaN	NaN	NaN	NaN	N
3	2017-06-02	1974	4.19	0	848	165	41	28	861	2017-06-02 12:04AM	2017-06-02 6:03AM	316.0	37.0	3.
4	2017-06-03	2168	5.98	0	1200	142	29	69	1062	NaN	NaN	NaN	NaN	N

We can see missing values as represented by NaN above. Hence, counting the number of missing values per column.

Data Preprocessing

In [476]:

```
df.isnull().sum()
```

Out[476]:

```
Date                    0
Calories Burned         0
Distance                0
```

```
Distance                       0
Floors                         0
Minutes Sedentary              0
Minutes Lightly Active         0
Minutes Fairly Active          0
Minutes Very Active            0
Activity Calories              0
Sleep Start Time             317
Sleep End Time               317
Minutes Asleep               317
Minutes Awake                317
Number of Awakenings         317
Time in Bed                  317
Minutes REM Sleep            399
Minutes Light Sleep          399
Minutes Deep Sleep           399
Steps                          0
dtype: int64
```

There are 317 and 399 i.e. all missing values in the sleep related columns of our dataset. Hence, filling with *zero* for missing values.

In [477]:

```
# fill missing values with mean column values
df.fillna(df.mean(), inplace=True)

# mark zero values as missing or NaN
df['Sleep Start Time'].fillna(0,inplace=True)
df['Sleep End Time'].fillna(0,inplace=True)
df['Minutes REM Sleep'].fillna(0,inplace=True)
df['Minutes Light Sleep'].fillna(0,inplace=True)
df['Minutes Deep Sleep'].fillna(0,inplace=True)

'''
df['Minutes Asleep'].fillna(0,inplace=True)
df['Minutes Awake'].fillna(0,inplace=True)
df['Number of Awakenings'].fillna(0,inplace=True)
df['Time in Bed'].fillna(0,inplace=True)
'''
#df[['Sleep Start Time','Sleep End Time','Minutes Asleep','Minutes Awake','Number of
Awakenings','Time in Bed','Minutes REM Sleep','Minutes Light Sleep','Minutes Deep Sleep']] = df[['
Sleep Start Time','Sleep End Time','Minutes Asleep','Minutes Awake','Number of Awakenings','Time i
n Bed','Minutes REM Sleep','Minutes Light Sleep','Minutes Deep Sleep']].replace(0, np.NaN)
df.head()
#df.isnull().sum()
```

Out[477]:

	Date	Calories Burned	Distance	Floors	Minutes Sedentary	Minutes Lightly Active	Minutes Fairly Active	Minutes Very Active	Activity Calories	Sleep Start Time	Sleep End Time	Minutes Asleep	Minut Awa
0	2017-05-30	1681	2.15	0	1311	117	3	9	467	0	0	248.304878	21.8292
1	2017-05-31	1892	3.29	0	1198	226	7	1	779	2017-05-31 11:51PM	2017-06-01 6:04AM	368.000000	4.00000
2	2017-06-01	1986	3.92	0	810	218	25	23	903	0	0	248.304878	21.8292
3	2017-06-02	1974	4.19	0	848	165	41	28	861	2017-06-02 12:04AM	2017-06-02 6:03AM	316.000000	37.0000
4	2017-06-03	2168	5.98	0	1200	142	29	69	1062	0	0	248.304878	21.8292

In [478]:

```
print("Dimensions of the data collected for last 1 year + starting from May 30, 2017: ",df.shape)
```

```
Dimensions of the data collected for last 1 year + starting from May 30, 2017:  (399, 19)
```

Dropping the columns that are not required

In [479]:

```
df.drop(df.columns[[3,9,10,15,16,17]], axis=1, inplace=True)
```

In [480]:

```
df.head()
```

Out[480]:

	Date	Calories Burned	Distance	Minutes Sedentary	Minutes Lightly Active	Minutes Fairly Active	Minutes Very Active	Activity Calories	Minutes Asleep	Minutes Awake	Number of Awakenings	Time B
0	2017-05-30	1681	2.15	1311	117	3	9	467	248.304878	21.829268	2.353659	271.1829
1	2017-05-31	1892	3.29	1198	226	7	1	779	368.000000	4.000000	3.000000	372.0000
2	2017-06-01	1986	3.92	810	218	25	23	903	248.304878	21.829268	2.353659	271.1829
3	2017-06-02	1974	4.19	848	165	41	28	861	316.000000	37.000000	3.000000	358.0000
4	2017-06-03	2168	5.98	1200	142	29	69	1062	248.304878	21.829268	2.353659	271.1829

In [481]:

```
print("Dimensions of the cleaned dataset: ",df.shape)
```

```
Dimensions of the cleaned dataset:  (399, 13)
```

Exploratory Data Analysis

Feature distributions

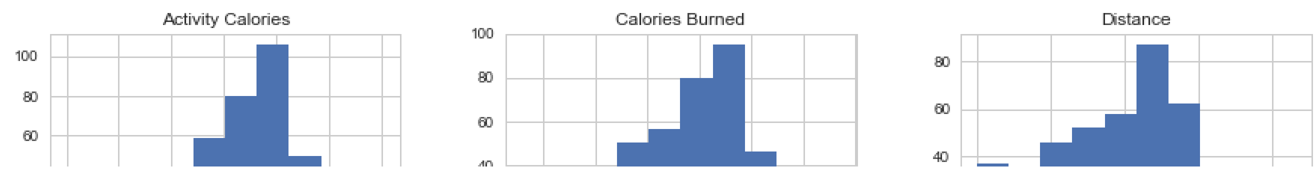
In [482]:

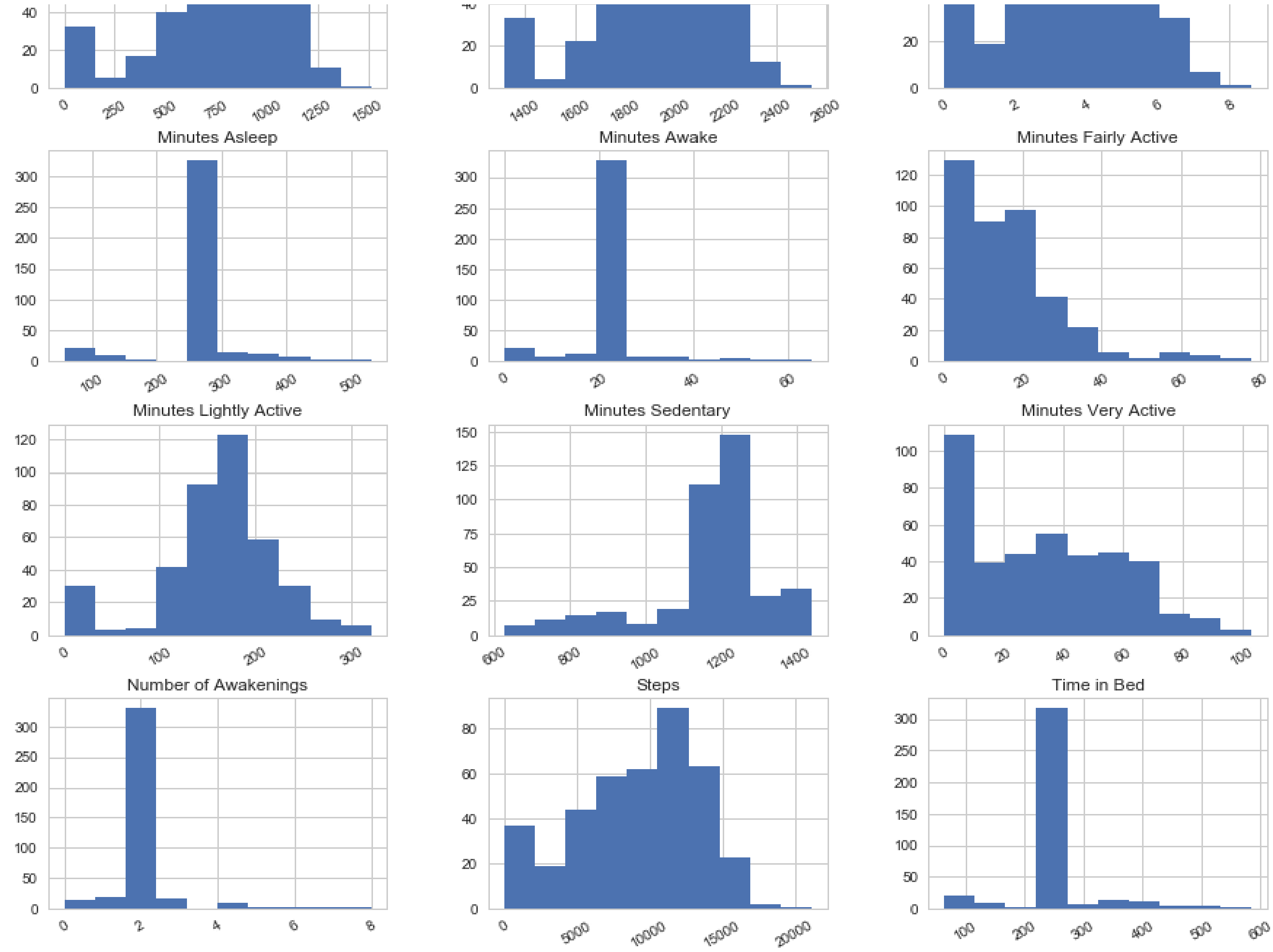
```
print(list(df))
```

```
['Date', 'Calories Burned', 'Distance', 'Minutes Sedentary', 'Minutes Lightly Active', 'Minutes Fa
irly Active', 'Minutes Very Active', 'Activity Calories', 'Minutes Asleep', 'Minutes Awake', 'Numb
er of Awakenings', 'Time in Bed', 'Steps']
```

In [483]:

```
# Looking at the distributions corresponding to each numerical variable in the raw data
df.dtypes
h = df.hist(figsize = (15,20), layout = (6,3), xrot = 30)
plt.savefig('images/raw-data-eda.png', dpi=300)
plt.show()
```





Initial Observations

1. Some of the data is zero: Reasons could be Fitbit is not worn, battery discharge or not synced for 10 consecutive days (In this case: it is mostly sleep data.)
2. Sedentary minutes are longer than activite minutes.
3. But majority calorie burn is due to activity calories which is some exercise or continuous walking or workout.
4. On average sleep is around 4-5 hours.
5. Daily steps vary between 5000 to 16000 which is close to 9-10 miles.

Looking at correlations

Visualizing the important characteristics of a dataset: Observing pair-wise correlations between features

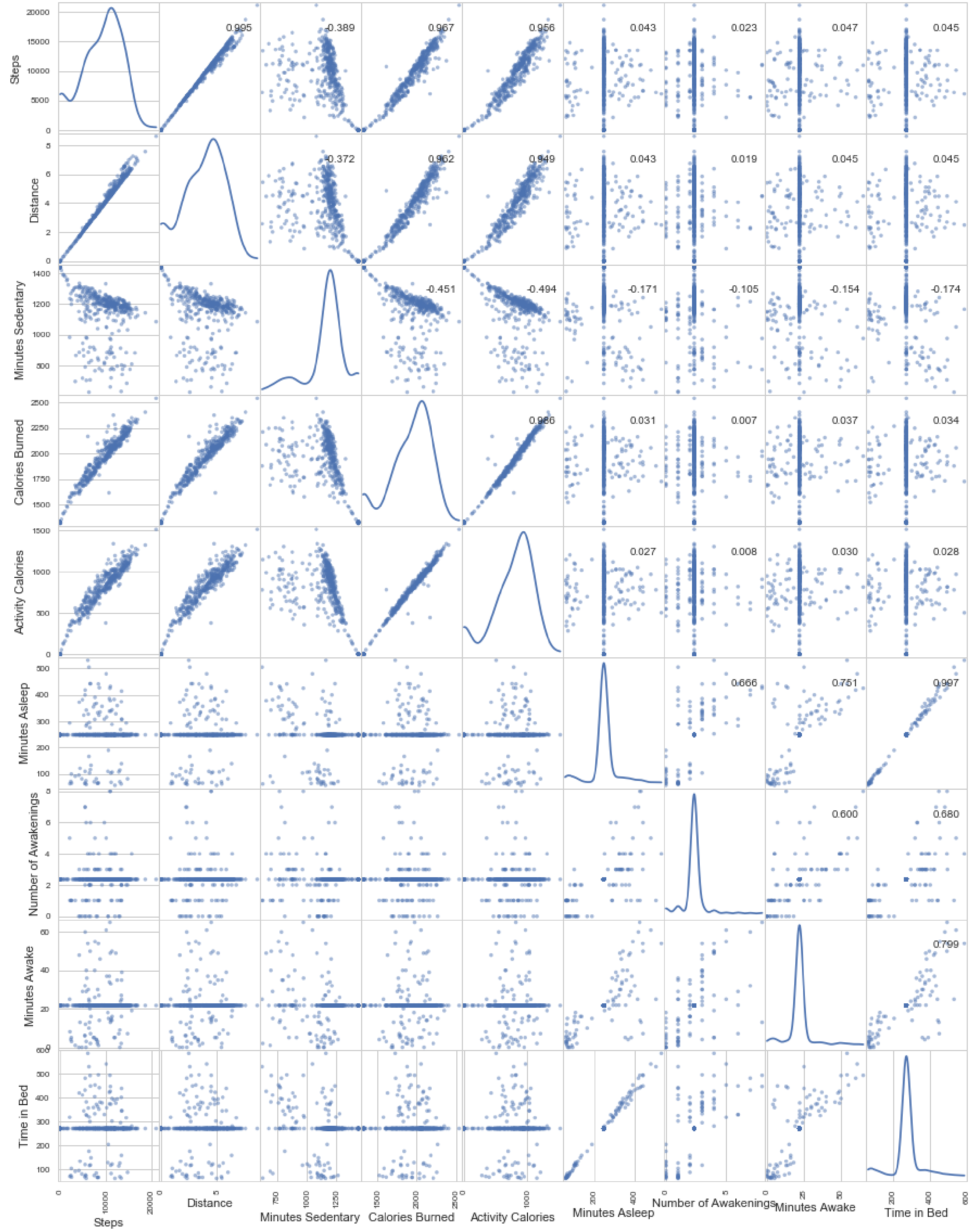
Using the scatter plot below to see how the data is distributed and whether it has any outliers.

In [484]:

```
#sns.set(style='whitegrid', context='notebook')
df_partial = df[['Steps','Distance','Minutes Sedentary','Calories Burned','Activity Calories','Min
utes Asleep','Number of Awakenings',\
                     'Minutes Awake','Time in Bed']]
axes = pd.scatter_matrix(df_partial, figsize = (15,20), alpha=0.5, diagonal='kde')

corr = df_partial.corr().as_matrix()
for i, j in zip(*plt.np.triu_indices_from(axes, k=1)):
    axes[i, j].annotate("%.3f" %corr[i,j], (0.8, 0.8), xycoords='axes fraction', ha='center', va='c
enter')
#sns.pairplot(df[cols], size=2.5)
plt.savefig('images/pairwise-correlation-matrix.png', dpi=300)
plt.show()
```

```
/Users/devasenainupakutika/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3:
FutureWarning: pandas.scatter_matrix is deprecated, use pandas.plotting.scatter_matrix instead
```

```
  This is separate from the ipykernel package so we can avoid doing imports until
/Users/devasenainupakutika/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:5:
FutureWarning: Method .as_matrix will be removed in a future version. Use .values instead.
  """
```



Further Data Munging

In [485]:

```
# Data cleaning and manipulation

# Create a weekday label which says which day of the week
```

```
df['weekday'] = df['Date'].map(lambda x: (datetime.strptime(str(x),"%Y-%m-%d")).weekday() , na_acti
on = 'ignore')
df['day'] = df['Date'].map(lambda x: (datetime.strptime(str(x),"%Y-%m-%d")).date , na_action = 'ign
ore')
df['month'] = df['Date'].map(lambda x: (datetime.strptime(str(x),"%Y-%m-%d")).month , na_action = '
ignore')
# Percentage of awake time to time in bed (related to efficiency)
df['sleep_awake_per'] = df['Minutes Awake']/df['Time in Bed']*100
```

In [486]:

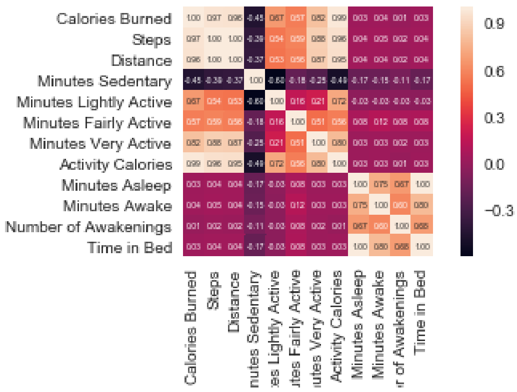
```
# Function to clean up plots
def prepare_plot_area(ax):
    # Remove plot frame lines
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["left"].set_visible(False)

    # X and y ticks on bottom and left
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

# Defining a color pattern that is pleasing
colrcode = [(31, 119, 180), (255, 127, 14),\
             (44, 160, 44), (214, 39, 40), \
             (148, 103, 189),  (140, 86, 75), \
             (227, 119, 194), (127, 127, 127), \
             (188, 189, 34), (23, 190, 207)]

for i in range(len(colrcode)):
    r, g, b = colrcode[i]
    colrcode[i] = (r / 255., g / 255., b / 255.)
```

Data Interaction

Here, we look at the trend shared by predictors, i.e the features that will be used to predict Steps. The correlation matrix is computed and represented as heatmap below:

In [487]:

```
#Plotting correlation matrix as heatmap
cols = ['Calories Burned', 'Steps', 'Distance', 'Minutes Sedentary', 'Minutes Lightly Active', 'Min
utes Fairly Active', 'Minutes Very Active', 'Activity Calories', 'Minutes Asleep', 'Minutes Awake'
, 'Number of Awakenings', 'Time in Bed']
cm = np.corrcoef(df[cols].values.T)
hm = sns.heatmap(cm,
                 cbar=True,
                 annot=True,
                 square=True,
                 fmt='.2f',
                 annot_kws={'size': 5},
                 yticklabels=cols,
                 xticklabels=cols)

plt.tight_layout()
plt.savefig('images/corr_heatmap.png', dpi=300)
plt.show()
```



From above heatmap, we can observe some strong correlation between some sleep predictors. Distance is strongly correlated to Steps and both are inter-correlated to Calories Burned and Activity Calories and also Minutes Very Active, which indicates that my main calorie burn is due to exercise or workout.

Insights from Data Analysis

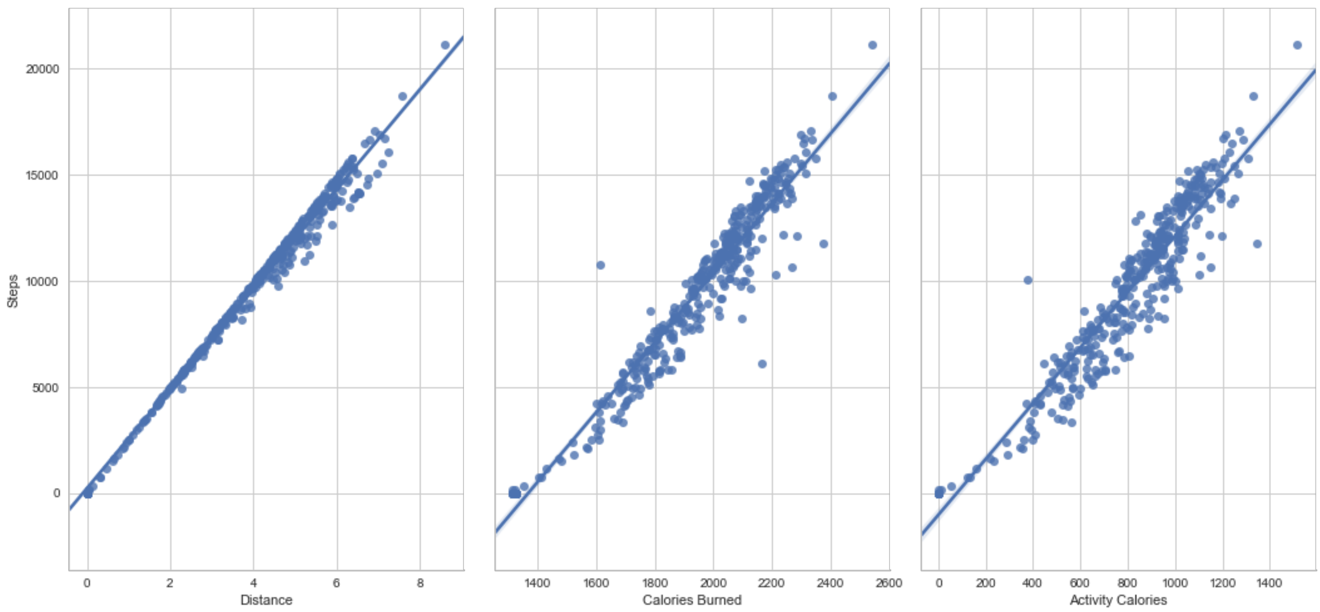
My Fitbit Flex2 data shows some strong correlation between predictors such as Distance, Calories burned, Activity Calories and Minutes very active (1.00, 0.97, 0.96 and 0.88 correlation).

Relation between Steps and Predictors are as shown in below graphs:

Steps Vs Predictors (Distance, Calories Burned and Activity Calories) All in One

In [488]:

```
sns.pairplot(df, x_vars=['Distance','Calories Burned','Activity Calories'], y_vars='Steps', size=7,
aspect=0.7, kind='reg')
plt.savefig('images/allinonestepsvspred.png', dpi=300)
```



Step Variations, Sleep Minutes and Sleep Inefficiency based on Week Days

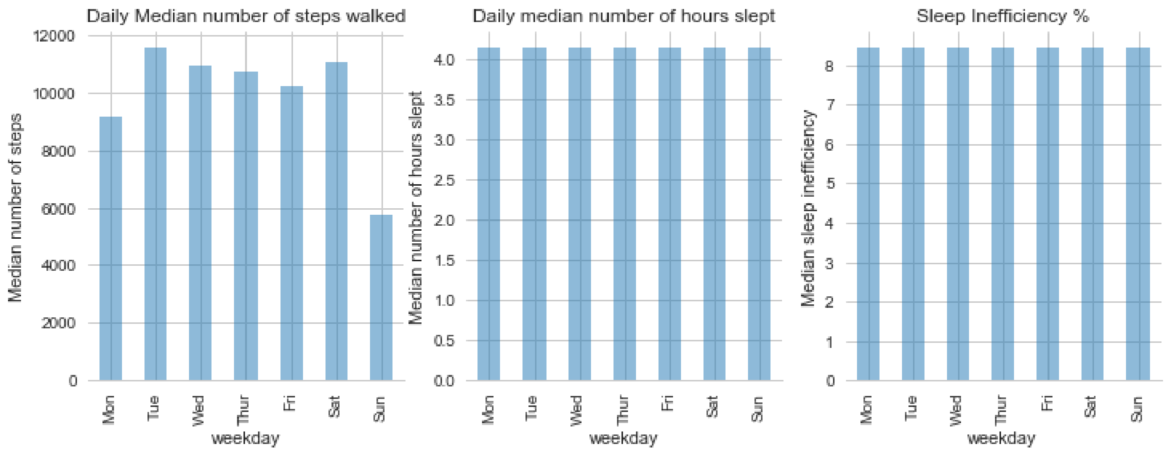
In [489]:

```
# Looking at variations based on weekday
steps_weekday = df['Steps'].groupby(df['weekday']).median()
sleep_minutes_asleep_med = df['Minutes Asleep'].groupby(df['weekday']).median()/60
sleep_eff = (1-df['Minutes Asleep']/df['Time in Bed'])*100
sl = sleep_eff.groupby(df['weekday']).median()
```

In [490]:

```
# Median number of steps
fig,axes = plt.subplots(figsize=(12, 4), nrows=1, ncols=3)

ct = 0
plt.sca(axes[ct])
steps_weekday.plot(kind = 'bar',color = colrcode[0], alpha = 0.5)
plt.ylabel('Median number of steps')
plt.title('Daily Median number of steps walked')
plt.xticks(list(range(7)),['Mon','Tue','Wed','Thur','Fri','Sat','Sun'])
prepare_plot_area(axes[ct])
```

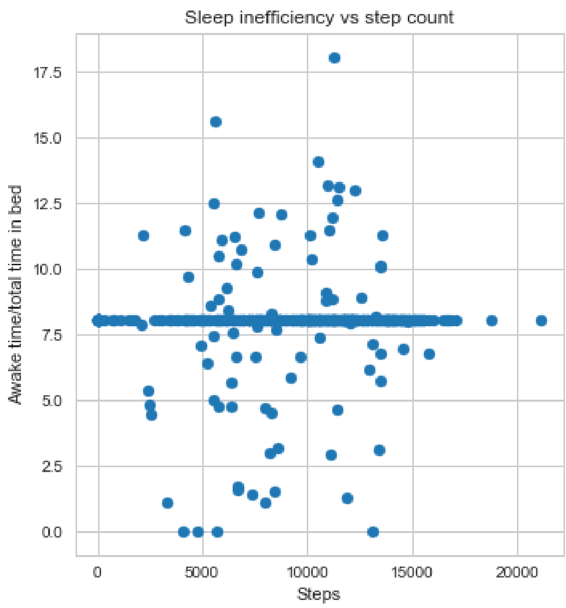
```
# Median number of minutes slept
ct +=1
plt.sca(axes[ct])
sleep_minutes_asleep_med.plot(kind = 'bar',color = colrcode[0], alpha = 0.5)
plt.ylabel('Median number of hours slept')
plt.title('Daily median number of hours slept')
plt.xticks(list(range(7)),['Mon','Tue','Wed','Thur','Fri','Sat','Sun'])
prepare_plot_area(axes[ct])

ct +=1
plt.sca(axes[ct])
sl.plot(kind = 'bar',color = colrcode[0], alpha = 0.5)
plt.ylabel('Median sleep inefficiency')
plt.title('Sleep Inefficiency %')
plt.xticks(list(range(7)),['Mon','Tue','Wed','Thur','Fri','Sat','Sun'])
prepare_plot_area(axes[ct])
plt.savefig('images/Steps-sleep-weekday.png', dpi=300)
```



Correlation between Step Count and Sleep Inefficiency

In [491]:

```
fig = plt.figure(figsize = (12,6))
ax = fig.add_subplot(121)
ax.scatter(df['Steps'],df['sleep_awake_per'],color = colrcode[0])
plt.xlabel('Steps')
plt.ylabel('Awake time/total time in bed')
plt.title('Sleep inefficiency vs step count')
plt.savefig('images/sleepineff-steps.png', dpi=300)
```



Steps Prediction and Evaluation

Since the target variable is **Steps** here. In order to predict **Steps**, we split our data into train (70%) and test (30%) datasets.

Simple Linear Regression Model

In [492]:

```
from sklearn.cross_validation import train_test_split
import statsmodels.formula.api as smf
from sklearn.linear_model import LinearRegression
from sklearn import metrics

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=0)

#Starting it with one feature currently called 'Distance' because of correlation r =1 and estimati
ng the coefficient
# create X and y
feature_cols = ['Distance']
X = df[feature_cols]
y = df.Steps

# instantiate and fit
slr = LinearRegression()
slr.fit(X, y)

# print the coefficients
print(slr.intercept_)
print(slr.coef_)

### STATSMODELS ###

# create a fitted model
lm1 = smf.ols(formula='Steps ~ Distance', data=df).fit()

# print the coefficients
lm1.params
```

```
215.19331101352327
[2349.72926778]
```

Out[492]:

```
Intercept     215.193311
Distance     2349.729268
dtype: float64
```

Using the model for prediction:

y = 215.1933 + 2349.729 * x

In [493]:

```
#For distance of 5 miles
slr.predict(5)
```

Out[493]:

```
array([11963.83964993])
```

In [494]:

```
### STATSMODELS ###

# We have to create a DataFrame since the Statsmodels formula interface expects it
X_new = pd.DataFrame({'Distance': [5]})

# predict for a new observation
lm1.predict(X_new)
```

Out[494]:

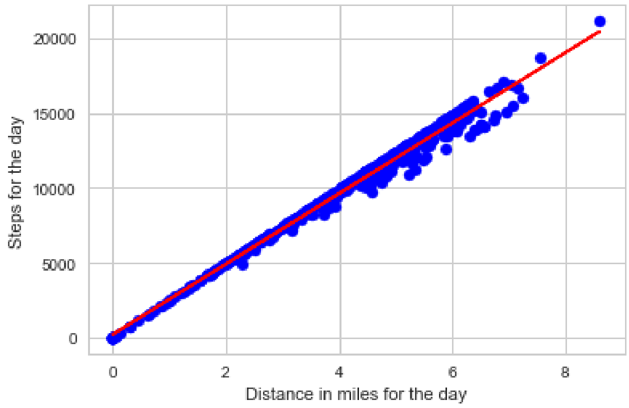
```
0    11963.83965
```

```
dtype: float64
```

Plotting the least squares line

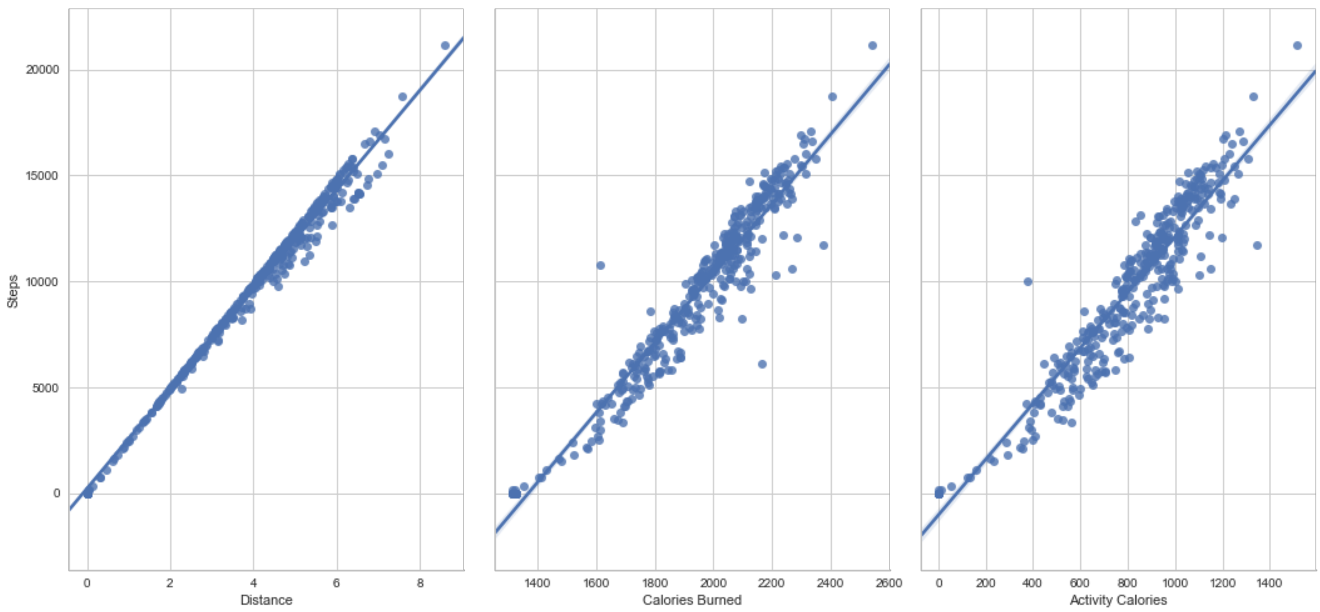
In [495]:

```
def lin_regplot(X,y,model):
    plt.scatter(X,y,c="blue")
    plt.plot(X,model.predict(X),color="red")
    return None

lin_regplot(X,y,slr)
plt.xlabel("Distance in miles for the day")
plt.ylabel("Steps for the day")
plt.savefig('images/lin-reg.png', dpi=300)
plt.show()
```



In [496]:

```
sns.pairplot(df, x_vars=['Distance','Calories Burned','Activity Calories'], y_vars='Steps', size=7,
aspect=0.7, kind='reg')
plt.savefig('images/stepsvspred-model.png', dpi=300)
```



Assessing variable importance using linear regression and test p-values for each predictor

In [497]:

```
# print the p-values for the model coefficients for Distance Predictor
lm1.pvalues

#p-values for other variables and predictors

### STATSMODELS for Calories Burned###
```

```python
#Removing space between columns
df=df.rename(columns={"Calories Burned":"Calories_Burned", "Activity Calories":"Activity_Calories"
,"Minutes Very Active":"Minutes_Very_Active","Minutes Awake":"Minutes_Awake","Time in 
Bed":"Time_in_Bed","Minutes Asleep":"Minutes_Asleep"})
```

In [498]:

```python
df
```

Out[498]:

	Date	Calories_Burned	Distance	Minutes Sedentary	Minutes Lightly Active	Minutes Fairly Active	Minutes_Very_Active	Activity_Calories	Minutes_Aslee
0	2017-05-30	1681	2.15	1311	117	3	9	467	248.304878
1	2017-05-31	1892	3.29	1198	226	7	1	779	368.000000
2	2017-06-01	1986	3.92	810	218	25	23	903	248.304878
3	2017-06-02	1974	4.19	848	165	41	28	861	316.000000
4	2017-06-03	2168	5.98	1200	142	29	69	1062	248.304878
5	2017-06-04	1327	0.00	1440	0	0	0	0	248.304878
6	2017-06-05	1326	0.00	1440	0	0	0	0	248.304878
7	2017-06-06	1326	0.00	1440	0	0	0	0	248.304878
8	2017-06-07	1326	0.00	1440	0	0	0	0	248.304878
9	2017-06-08	1326	0.00	1440	0	0	0	0	248.304878
10	2017-06-09	1326	0.00	1440	0	0	0	0	248.304878
	2017-								

	Date	Calories_Burned	Distance	Minutes Sedentary	Minutes Lightly Active	Minutes Fairly Active	Minutes_Very_Active	Activity_Calories	Minutes_Aslee
11	2017-06-10	1326	0.00	1440	0	0	0	0	248.304878
12	2017-06-11	1326	0.00	1440	0	0	0	0	248.304878
13	2017-06-12	1326	0.00	1440	0	0	0	0	248.304878
14	2017-06-13	1326	0.00	1440	0	0	0	0	248.304878
15	2017-06-14	1326	0.00	1440	0	0	0	0	248.304878
16	2017-06-15	1326	0.00	1440	0	0	0	0	248.304878
17	2017-06-16	1326	0.00	1440	0	0	0	0	248.304878
18	2017-06-17	1326	0.00	1440	0	0	0	0	248.304878
19	2017-06-18	1326	0.00	1440	0	0	0	0	248.304878
20	2017-06-19	1326	0.00	1440	0	0	0	0	248.304878
21	2017-06-20	1326	0.00	1440	0	0	0	0	248.304878
22	2017-06-21	1326	0.00	1440	0	0	0	0	248.304878
23	2017-06-22	1326	0.00	1440	0	0	0	0	248.304878
24	2017-06-23	1326	0.00	1440	0	0	0	0	248.304878
25	2017-06-24	1520	0.96	1268	97	0	0	284	71.000000

	Date	Calories_Burned	Distance	Minutes Sedentary	Minutes Lightly Active	Minutes Fairly Active	Minutes_Very_Active	Activity_Calories	Minutes_Asleep
26	2017-06-25	1674	1.40	864	148	0	0	327	248.304878
27	2017-06-26	1819	3.18	903	142	9	22	652	364.000000
28	2017-06-27	2020	4.48	773	181	15	38	909	432.000000
29	2017-06-28	1675	2.06	902	123	13	7	481	248.304878
...	...	...	...	...	...	...	...	...	...
369	2018-06-03	2121	4.51	968	321	14	20	1106	103.000000
370	2018-06-04	2108	4.14	1186	182	56	16	1002	248.304878
371	2018-06-05	2176	5.97	1180	154	37	69	1072	248.304878
372	2018-06-06	2091	5.39	1147	161	21	41	949	70.000000
373	2018-06-07	2013	4.32	1200	155	68	17	888	248.304878
374	2018-06-08	2239	6.40	1159	201	24	56	1150	248.304878
375	2018-06-09	2068	5.37	1234	128	20	58	896	248.304878
376	2018-06-10	1958	3.32	1140	282	4	14	891	248.304878
377	2018-06-11	2194	5.81	1155	180	43	62	1108	248.304878
378	2018-06-12	2075	5.49	1181	177	19	63	966	248.304878
[illegible]	2018-	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

	Date	Calories_Burned	Distance	Minutes Sedentary	Minutes Lightly Active	Minutes Fairly Active	Minutes_Very_Active	Activity_Calories	Minutes_Aslee
379	06-13	1998	4.41	1219	166	22	33	853	248.304878
380	2018-06-14	1954	4.05	1197	210	11	22	831	248.304878
381	2018-06-15	2212	6.39	1195	131	27	87	1093	248.304878
382	2018-06-16	1987	4.68	1248	123	13	56	824	248.304878
383	2018-06-17	2035	4.14	1150	254	13	23	948	248.304878
384	2018-06-18	2029	3.69	1198	158	65	19	906	248.304878
385	2018-06-19	2168	5.91	1201	157	13	69	1042	248.304878
386	2018-06-20	1927	3.86	1227	173	12	28	776	248.304878
387	2018-06-21	1999	3.80	1190	180	55	15	878	248.304878
388	2018-06-22	2199	6.29	1195	149	35	61	1081	248.304878
389	2018-06-23	2006	4.92	1243	113	32	52	838	248.304878
390	2018-06-24	1844	2.34	1167	273	0	0	749	248.304878
391	2018-06-25	2066	4.99	1209	143	25	63	933	248.304878
392	2018-06-26	1947	4.09	1226	150	37	27	796	248.304878
393	2018-06-27	2148	5.54	1175	188	22	55	1046	248.304878

	Date	Calories_Burned	Distance	Minutes Sedentary	Minutes Lightly Active	Minutes Fairly Active	Minutes_Very_Active	Activity_Calories	Minutes_Aslee
394	2018-06-28	1924	3.88	1234	151	23	32	771	248.304878
395	2018-06-29	2061	5.01	1200	153	37	50	939	248.304878
396	2018-06-30	1956	3.71	1196	196	16	32	820	248.304878
397	2018-07-01	1818	2.33	1126	251	0	0	703	60.000000
398	2018-07-02	1612	4.85	888	98	14	53	785	248.304878

399 rows × 17 columns

In [499]:

```
### STATSMODELS for calories_burned ###

# create a fitted model
lm11 = smf.ols(formula='Steps ~ Calories_Burned', data=df).fit()

# print the coefficients
lm11.params

### STATSMODELS ###

# We have to create a DataFrame since the Statsmodels formula interface expects it
X_new1 = pd.DataFrame({'Calories_Burned': [2000]})

# predict for a new observation
lm11.predict(X_new1)
# print the p-values for the model coefficients for Distance Predictor
lm11.pvalues
```

Out[499]:

```
Intercept          4.626848e-183
Calories_Burned    4.064068e-238
dtype: float64
```

In [500]:

```
### STATSMODELS for Activity_Calories ###

# create a fitted model
lm12 = smf.ols(formula='Steps ~ Activity_Calories', data=df).fit()

# print the coefficients
lm12.params

### STATSMODELS ###

# We have to create a DataFrame since the Statsmodels formula interface expects it
X_new2 = pd.DataFrame({'Activity_Calories': [1000]})

# predict for a new observation
lm12.predict(X_new2)
# print the p-values for the model coefficients for Calories Burned Predictor
lm12.pvalues
```

Out[500]:

```
Intercept              2.119603e-09
Activity_Calories    2.865917e-214
dtype: float64
```

In [501]:

```
### STATSMODELS for Minutes_Very_Active ###

# create a fitted model
lm13 = smf.ols(formula='Steps ~ Minutes_Very_Active', data=df).fit()

# print the coefficients
lm13.params

### STATSMODELS ###

# We have to create a DataFrame since the Statsmodels formula interface expects it
X_new3 = pd.DataFrame({'Minutes_Very_Active': [60]})

# predict for a new observation
lm13.predict(X_new3)
# print the p-values for the model coefficients for Minutes very active Predictor
lm13.pvalues
```

Out[501]:

```
Intercept                4.930023e-84
Minutes_Very_Active    5.748666e-128
dtype: float64
```

In [502]:

```
### STATSMODELS for sleep_awake_per ###

# create a fitted model
lm14 = smf.ols(formula='Steps ~ sleep_awake_per', data=df).fit()

# print the coefficients
lm14.params

### STATSMODELS ###

# We have to create a DataFrame since the Statsmodels formula interface expects it
X_new4 = pd.DataFrame({'sleep_awake_per': [8.00]})

# predict for a new observation
lm14.predict(X_new4)
# print the p-values for the model coefficients for sleep_awake_per Predictor
lm14.pvalues
```

Out[502]:

```
Intercept          1.000395e-14
sleep_awake_per    2.147004e-01
dtype: float64
```

How well model fits the data?

In [503]:

```
slr.score(X,y)
```

Out[503]:

```
0.990445703687681
```

In [504]:

```
### STATSMODELS ###

# print the R-squared value for the model
```

```
# print the R-squared value for the model
lm1.rsquared
```

Out[504]:

```
0.990445703687681
```

Multiple Linear Regression

In [505]:

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

In [506]:

```
# create X and y
feature_cols = ['Distance', 'Calories_Burned', 'Activity_Calories','sleep_awake_per']
X = df[feature_cols]
y = df.Steps

# instantiate and fit
slr2 = LinearRegression()
slr2.fit(X, y)

# print the coefficients
print(slr2.intercept_)
print(slr2.coef_)
# pair the feature names with the coefficients
list(zip(feature_cols, slr2.coef_))
slr2.score(X,y)
```

```
-1057.9283169964856
[2.05827285e+03 7.02975896e-01 1.20758555e+00 1.15121071e+01]
```

Out[506]:

```
0.9918567000076867
```

In [507]:

```
lm14.summary()
```

Out[507]:

OLS Regression Results

Dep. Variable:	Steps	**R-squared:**	0.004
Model:	OLS	**Adj. R-squared:**	0.001
Method:	Least Squares	**F-statistic:**	1.544
Date:	Sat, 07 Jul 2018	**Prob (F-statistic):**	0.215
Time:	16:05:30	**Log-Likelihood:**	-3909.8
No. Observations:	399	**AIC:**	7824.
Df Residuals:	397	**BIC:**	7831.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	7987.4807	992.741	8.046	0.000	6035.794	9939.168
sleep_awake_per	151.5168	121.922	1.243	0.215	-88.177	391.211

Omnibus:	17.710	**Durbin-Watson:**	1.140
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	18.239

Skew:	-0.493	**Prob(JB):**	0.000110
Kurtosis:	2.645	**Cond. No.**	37.5

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Evaluating Multiple Regression Model

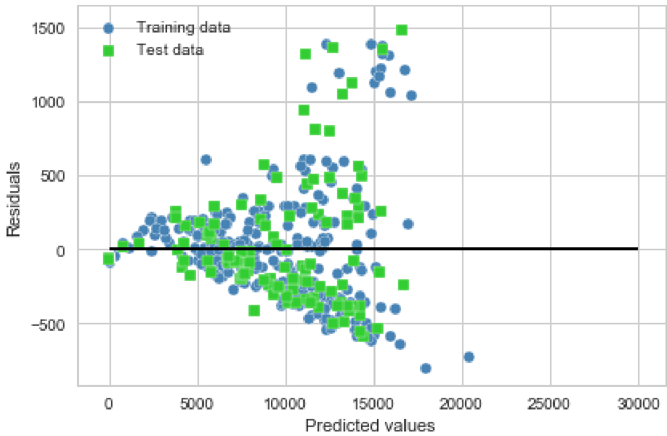
In [508]:

```
feature_cols = ['Distance', 'Calories_Burned', 'Activity_Calories','sleep_awake_per']
X = df[feature_cols]
y = df.Steps
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=0)
# instantiate and fit
slr3 = LinearRegression()
slr3.fit(X_train, y_train)

y_train_pred = slr3.predict(X_train)
y_test_pred = slr3.predict(X_test)
print('MSE train: %.3f, test: %.3f' % (
        mean_squared_error(y_train, y_train_pred),
        mean_squared_error(y_test, y_test_pred)))
print('R^2 train: %.3f, test: %.3f' % (
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)))
```

```
MSE train: 150409.453, test: 168564.853
R^2 train: 0.992, test: 0.991
```

In [509]:

```
plt.scatter(y_train_pred,  y_train_pred - y_train,
            c='steelblue', marker='o', edgecolor='white',
            label='Training data')
plt.scatter(y_test_pred,  y_test_pred - y_test,
            c='limegreen', marker='s', edgecolor='white',
            label='Test data')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0,xmin=0,xmax=30000, color='black', lw=2)
plt.tight_layout()

plt.savefig('images/eval_multiple_linear_regression_model-Residual-Plot.png', dpi=300)
plt.show()
```



Using Regularized Methods for Regression (to tackle problems of Overfitting)

Ridge Regression Model (L2 Penalized Model)

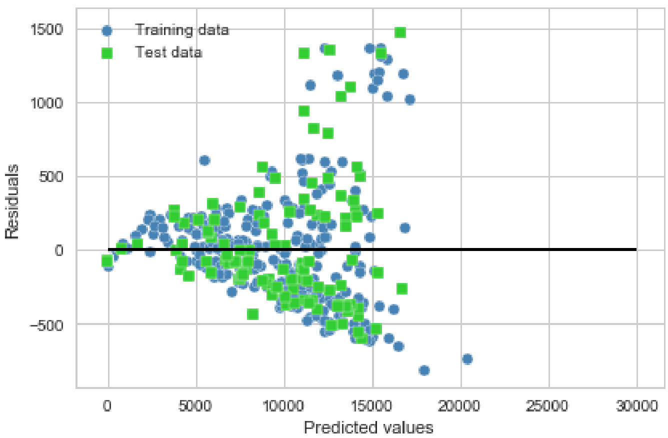
In [510]:

```
from sklearn.linear_model import Ridge
```

In [511]:

```
ridge=Ridge(alpha=1.0)
ridge.fit(X_train, y_train)

y_train_pred = ridge.predict(X_train)
y_test_pred = ridge.predict(X_test)
print('MSE train: %.3f, test: %.3f' % (
        mean_squared_error(y_train, y_train_pred),
        mean_squared_error(y_test, y_test_pred)))
print('R^2 train: %.3f, test: %.3f' % (
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)))
plt.scatter(y_train_pred,  y_train_pred - y_train,
            c='steelblue', marker='o', edgecolor='white',
            label='Training data')
plt.scatter(y_test_pred,  y_test_pred - y_test,
            c='limegreen', marker='s', edgecolor='white',
            label='Test data')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0,xmin=0,xmax=30000, color='black', lw=2)
plt.tight_layout()

plt.savefig('images/Ridge-Residual-Plot.png', dpi=300)
plt.show()
slr3.score(X,y)
```

```
MSE train: 150634.735, test: 167937.944
R^2 train: 0.992, test: 0.991
```



Out[511]:

```
0.9918277596036082
```

In [512]:

```
ridge.score(X,y)
```

Out[512]:

```
0.991829385760251
```

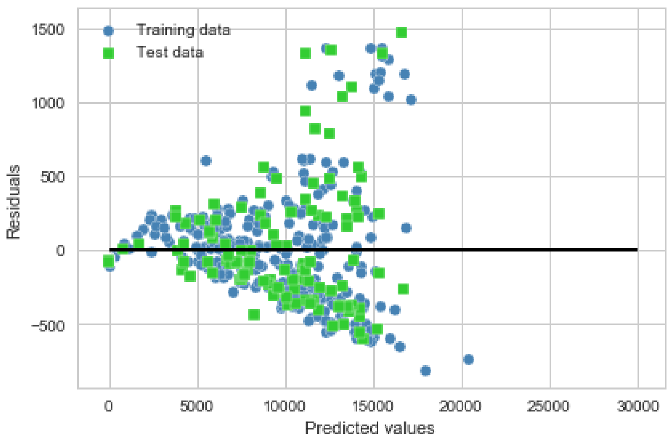
LASSO Regression Model

In [513]:

```
from sklearn.linear_model import Lasso
lasso=Lasso(alpha=1.0)
```

```
lasso.fit(X_train, y_train)

y_train_pred = ridge.predict(X_train)
y_test_pred = ridge.predict(X_test)
print('MSE train: %.3f, test: %.3f' % (
        mean_squared_error(y_train, y_train_pred),
        mean_squared_error(y_test, y_test_pred)))
print('R^2 train: %.3f, test: %.3f' % (
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)))
plt.scatter(y_train_pred,  y_train_pred - y_train,
            c='steelblue', marker='o', edgecolor='white',
            label='Training data')
plt.scatter(y_test_pred,  y_test_pred - y_test,
            c='limegreen', marker='s', edgecolor='white',
            label='Test data')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0,xmin=0,xmax=30000, color='black', lw=2)
plt.tight_layout()

plt.savefig('images/Lasso-Residual-Plot.png', dpi=300)
plt.show()
lasso.score(X,y)
```

```
MSE train: 150634.735, test: 167937.944
R^2 train: 0.992, test: 0.991
```



Out[513]:

```
0.9918299540652497
```

Random Forest Regression

In [514]:

```
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor(n_estimators=1000,criterion="mse",random_state=1,n_jobs=-1)
forest.fit(X_train,y_train)
y_train_pred=forest.predict(X_train)
y_test_pred=forest.predict(X_test)
print('MSE train: %.3f, test: %.3f' % (
        mean_squared_error(y_train, y_train_pred),
        mean_squared_error(y_test, y_test_pred)))
print('R^2 train: %.3f, test: %.3f' % (
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)))
plt.scatter(y_train_pred,
            y_train_pred - y_train,
            c='steelblue',
            edgecolor='white',
            marker='o',
            s=35,
            alpha=0.9,
            label='training data')
plt.scatter(y_test_pred,
```

```
            y_test_pred - y_test,
            c='limegreen',
            edgecolor='white',
            marker='s',
            s=35,
            alpha=0.9,
            label='test data')

plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin=0, xmax=30000, lw=2, color='black')
plt.xlim([0, 30000])
plt.tight_layout()
plt.savefig('images/forest_regression_plot.png', dpi=300)
plt.show()
```

```
MSE train: 27928.392, test: 205092.993
R^2 train: 0.999, test: 0.990
```