Polar

September 30, 2020

1 Imports

[1]:
```
import json
from functools import reduce
from os import listdir
from os.path import isfile, join
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
from pylab import rcParams
from scipy.stats import shapiro
from statsmodels.graphics.gofplots import qqplot
from statsmodels.stats.diagnostic import het_breuschpagan
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

2 Setup

[2]:
```
# Set figure size
rcParams['figure.figsize'] = (4, 4)

# Folder for images
Path('img').mkdir(parents=True, exist_ok=True)

# Nice float format
pd.options.display.float_format = "{:,.2f}".format
```

3 Data description

Last year I purchased a Polar watch that tracks my vitals during workouts. I used the Polar Flow website to obtain a copy of my data. For privacy reasons I shall not be sharing the dataset.

[3]:
```
path = './data/'
```

First, we create a list of files in the download.

[4]:
```
files = [f for f in listdir(path) if isfile(join(path, f))]
```

We shall only consider files containing the string `'training-session'`.

[5]:
```
files = [f for f in files if 'training-session' in f]
```

The number of files under consideration is:

[6]:
```
len(files)
```

[6]: 284

We loop over each of the files and them to a list.

[7]:
```
data = []

for f in files:
    with open(join(path, f)) as f:
        d = json.load(f)
        data.append(d)
```

We define a function to extract statistics about heart rate measured during the workouts.

[8]:
```
quantiles = [0.01, 0.25, 0.5, 0.75, 0.99]
```

[9]:
```
def extract_hr_info(workout, quantiles):

    stats = {'heartRateAvg2': np.nan,
             'heartRateStd': np.nan}

    for q in quantiles:
        stats[f'heartRateQ' + str(int(q * 100))] = np.nan

    # Check if data exists
    try:
        heart_rates = workout['exercises'][0]['samples']['heartRate']
    except KeyError:
        return stats

    # Loop over measurements
    hr_data = []
    for hr in heart_rates:

        # Check if actually measured hr
        if 'value' in hr:
            hr_data.append(hr['value'])
```

```
    stats['heartRateAvg2'] = np.mean(hr_data)
    stats['heartRateStd'] = np.std(hr_data)

    for q in quantiles:
        stats[f'heartRateQ' + str(int(q * 100))] = np.quantile(hr_data, q)

    return stats
```

We extract the relevant information from the items in the list.

[10]:
```
workouts = []

for d in data:
    basic = d['exercises'][0]
    hr = extract_hr_info(workout=d,
                         quantiles=quantiles)

    workouts.append({**basic, **hr})
```

Finally we create a dataframe containing the workout information.

[11]:
```
df = pd.DataFrame(workouts)
```

4 Data structure

We find the following columns in the dataframe.

[12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284 entries, 0 to 283
Data columns (total 24 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   startTime       284 non-null    object
 1   stopTime        284 non-null    object
 2   timezoneOffset  284 non-null    int64
 3   duration        284 non-null    object
 4   sport           284 non-null    object
 5   kiloCalories    283 non-null    float64
 6   heartRate       283 non-null    object
 7   zones           284 non-null    object
 8   samples         284 non-null    object
 9   heartRateAvg2   283 non-null    float64
 10  heartRateStd    283 non-null    float64
 11  heartRateQ1     283 non-null    float64
 12  heartRateQ25    283 non-null    float64
 13  heartRateQ50    283 non-null    float64
```

```
 14  heartRateQ75     283 non-null    float64
 15  heartRateQ99     283 non-null    float64
 16  distance         130 non-null    float64
 17  latitude         130 non-null    float64
 18  longitude        130 non-null    float64
 19  ascent           120 non-null    float64
 20  descent          121 non-null    float64
 21  speed            130 non-null    object
 22  autoLaps         102 non-null    object
 23  laps             2 non-null      object
dtypes: float64(13), int64(1), object(10)
memory usage: 53.4+ KB
```

We remove columns that containt data from features I do not use in my training.

Due to privacy concerns I shan't be extracting longitudinal and latitudinal data.

[13]:
```
df = df.drop(['zones', 'samples', 'autoLaps',
              'laps', 'latitude', 'longitude',
              'ascent', 'descent'], axis=1)
```

[14]:
```
df.head()
```

[14]:
```
                 startTime                 stopTime  timezoneOffset  \
0  2019-05-24T13:18:14.000  2019-05-24T14:58:44.125             120
1  2019-05-04T12:03:34.000  2019-05-04T13:21:38.500             120
2  2019-04-12T12:48:57.000  2019-04-12T12:59:10.750             120
3  2019-06-12T13:13:09.000  2019-06-12T13:23:15.500             120
4  2019-05-24T14:59:06.000  2019-05-24T15:29:08.750             120

      duration             sport  kiloCalories  \
0  PT6030.125S  STRENGTH_TRAINING        658.00
1  PT4684.500S  STRENGTH_TRAINING        373.00
2   PT613.750S  TREADMILL_RUNNING         62.00
3   PT606.500S  TREADMILL_RUNNING         71.00
4  PT1802.750S  TREADMILL_RUNNING        416.00

                                 heartRate  heartRateAvg2  heartRateStd  \
0  {'min': 72, 'avg': 105, 'max': 136}         104.77         11.28
1   {'min': 71, 'avg': 99, 'max': 138}          98.65         12.51
2   {'min': 71, 'avg': 97, 'max': 107}          97.07          8.00
3  {'min': 67, 'avg': 105, 'max': 121}         105.24         11.25
4  {'min': 84, 'avg': 144, 'max': 170}         143.85         18.47

   heartRateQ1  heartRateQ25  heartRateQ50  heartRateQ75  heartRateQ99  \
0        77.00         99.00        105.00        111.00        132.00
1        74.00         91.00         97.00        106.00        126.00
2        72.00         94.00         97.00        104.00        107.00
```

```
3        67.96        98.00        104.00        118.00        121.00
4        87.00       133.00        146.00        158.00        169.00

   distance speed
0       nan   NaN
1       nan   NaN
2       nan   NaN
3       nan   NaN
4       nan   NaN
```

5 Missing Values

The watch tracks different information for different workouts. For example when walking it tracks location but when walking on a treadmill it doesn't, hence there is quite a lot of missing data.

[15]:
```
missing = (df.isna().sum() / df.shape[0] * 100)
missing.name = 'Percent missing'
missing = missing.to_frame()
missing = missing.sort_values('Percent missing', ascending=False)
missing = missing[missing['Percent missing'] > 0]
np.round(missing, 2)
```

[15]:
```
               Percent missing
distance                 54.23
speed                    54.23
kiloCalories              0.35
heartRate                 0.35
heartRateAvg2             0.35
heartRateStd              0.35
heartRateQ1               0.35
heartRateQ25              0.35
heartRateQ50              0.35
heartRateQ75              0.35
heartRateQ99              0.35
```

6 Transforms

We apply certain transforms to make the data easier to work with. First we convert strings to datetimes.

[16]:
```
df['startTime'] = pd.to_datetime(df['startTime'])
df['stopTime'] = pd.to_datetime(df['stopTime'])
```

We calculate the total duration of each individual workout in minutes.

```
[17]: df['totalTime'] = (df['stopTime'] - df['startTime'])
df['totalTime'] = df['totalTime'].apply(lambda x: round(x.seconds / 60, 2))
df.drop('duration', axis=1, inplace=True)
```

We extract maximum, average and minimum heart rate values from the `heartRate` column.

```
[18]: df['heartRateMax'] = df['heartRate'].apply(lambda x: x['max'] if isinstance(x,␣
 ↪dict) else np.nan)
df['heartRateAvg'] = df['heartRate'].apply(lambda x: x['avg'] if isinstance(x,␣
 ↪dict) else np.nan)
df['heartRateMin'] = df['heartRate'].apply(lambda x: x['min'] if isinstance(x,␣
 ↪dict) else np.nan)
df.drop('heartRate', axis=1, inplace=True)
```

We assume that if there is no `distance` then the workout was indoors:

```
[19]: df['indoors'] = df['distance'].apply(lambda x: True if pd.isnull(x) else False)
df = df.drop(['distance', 'speed'], axis=1)
```

We are going to map sports to different `activityType`'s. We will map strength training to `True` and cardiovascular work to `False`.

```
[20]: def sport_to_activity_type(x):
    if 'strength' in x.lower():
        return True
    else:
        return False
```

```
[21]: df['activityType'] = df['sport'].apply(sport_to_activity_type)
```

We extract a list of unique `sport` values:

```
[22]: sports = sorted(list(df['sport'].unique()))
```

We reorder the alphabetically

```
[23]: order = sorted(df.columns.to_list())
```

```
[24]: df = df[order]
```

We check if there are any more `NaN`'s in the data.

```
[25]: df.isna().sum()
```

```
[25]: activityType      0
heartRateAvg      1
heartRateAvg2     1
heartRateMax      1
heartRateMin      1
```

```
heartRateQ1        1
heartRateQ25       1
heartRateQ50       1
heartRateQ75       1
heartRateQ99       1
heartRateStd       1
indoors            0
kiloCalories       1
sport              0
startTime          0
stopTime           0
timezoneOffset     0
totalTime          0
dtype: int64
```

There is one row with `NaN`'s. This might due to my watch having little battery left to make the measurements.

[26]:
```python
df = df.dropna()
```

We proceed to sort the data with the latest workouts at the top of the dataframe.

[27]:
```python
sort_cols = ['startTime','startTime']
df = df.sort_values(sort_cols, ascending=False)
df = df.reset_index(drop=True)
```

We verify that the datatypes are correct.

[28]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 283 entries, 0 to 282
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   activityType     283 non-null    bool
 1   heartRateAvg     283 non-null    float64
 2   heartRateAvg2    283 non-null    float64
 3   heartRateMax     283 non-null    float64
 4   heartRateMin     283 non-null    float64
 5   heartRateQ1      283 non-null    float64
 6   heartRateQ25     283 non-null    float64
 7   heartRateQ50     283 non-null    float64
 8   heartRateQ75     283 non-null    float64
 9   heartRateQ99     283 non-null    float64
 10  heartRateStd     283 non-null    float64
 11  indoors          283 non-null    bool
 12  kiloCalories     283 non-null    float64
```

```
 13  sport           283 non-null    object
 14  startTime       283 non-null    datetime64[ns]
 15  stopTime        283 non-null    datetime64[ns]
 16  timezoneOffset  283 non-null    int64
 17  totalTime       283 non-null    float64
dtypes: bool(2), datetime64[ns](2), float64(12), int64(1), object(1)
memory usage: 36.1+ KB
```

7 Analysis

Given that we have produced a clean dataset we can proceed to analyse a few aspects.

7.1 Time span

The date of the first workout is:

[29]:
```python
str(df['startTime'].min())
```

[29]:
```
'2019-02-20 20:46:35'
```

The date of the last workout is:

[30]:
```python
str(df['startTime'].max())
```

[30]:
```
'2020-03-29 21:50:21'
```

Workouts measured:

[31]:
```python
len(df)
```

[31]:
```
283
```

7.2 Descriptive statistics

[32]:
```python
df.drop('timezoneOffset', axis=1).describe()
```

[32]:
```
       heartRateAvg  heartRateAvg2  heartRateMax  heartRateMin  heartRateQ1  \
count        283.00         283.00        283.00        283.00       283.00
mean         105.19         105.24        128.34         76.74        80.61
std           11.87          11.86         18.25          8.99         8.40
min           82.00          81.98         93.00         53.00        54.00
25%           96.00          96.42        115.00         70.00        75.00
50%          103.00         103.42        125.00         77.00        80.00
75%          111.00         111.26        138.50         83.00        86.00
max          148.00         148.35        178.00         99.00       107.00

       heartRateQ25  heartRateQ50  heartRateQ75  heartRateQ99  heartRateStd  \
count        283.00        283.00        283.00        283.00        283.00
```

```
mean          98.24      105.55      112.64      125.84       10.52
std           10.69       12.26       14.36       18.06        4.46
min           77.00       82.00       87.00       92.00        2.96
25%           91.00       97.00      102.00      113.00        7.58
50%           97.00      104.00      111.00      123.00       10.00
75%          103.00      112.00      119.00      135.00       12.25
max          146.00      151.00      160.00      177.00       27.10

       kiloCalories  totalTime
count        283.00     283.00
mean         315.98      42.83
std          218.75      29.65
min           29.00       5.00
25%          121.50      15.92
50%          277.00      36.45
75%          441.50      65.29
max        1,067.00     172.73
```

7.3 Kilocalories burned in total

First we count the total `kiloCalories` I burned during the period in question.

[33]:
```python
total_calories = df['kiloCalories'].sum()
print(total_calories)
```

```
89421.0
```

We convert this number to kilograms of body fat. According to this article it equates to

[34]:
```python
round(total_calories / 7700, 2)
```

[34]:
```
11.61
```

7.4 Kilocalories burned by sport

[35]:
```python
by_sport = df[['kiloCalories', 'sport']].groupby('sport', as_index=False)
by_sport = by_sport.sum()
by_sport['sport'] = by_sport['sport'].apply(lambda x: x.lower())
by_sport['kiloCalories'] = by_sport['kiloCalories'].astype(int)
by_sport = by_sport.rename(columns={'kiloCalories': 'total kilocalories'})
by_sport = by_sport.sort_values('total kilocalories', ascending=False)
by_sport
```

[35]:

	sport	total kilocalories
4	walking	33080
2	strength_training	31547
3	treadmill_running	19825
0	cycling	4029

```
1              running                       940
```

7.5 Kilocalories burned over time

Next we produce a plot of `kiloCalories` burned over a two month period in 2019. First we extract the relevant data.

[36]:
```
start = pd.to_datetime('2019-04-1')
stop = pd.to_datetime('2019-06-1')

daily = df[['startTime', 'kiloCalories']]
mask = (daily['startTime'] >= start) & (daily['startTime'] < stop)
daily = daily[mask]
daily['startTime'] = daily['startTime'].dt.date
daily = daily.groupby('startTime', as_index=False)
daily = daily.sum()
daily = daily.sort_values('startTime', ascending=False)
daily['startTime'] = pd.to_datetime(daily['startTime'])
daily = daily.reset_index(drop=True)
```

We create a dataframe with all the dates to perform a left join and fill the `NaN`'s with zeroes.

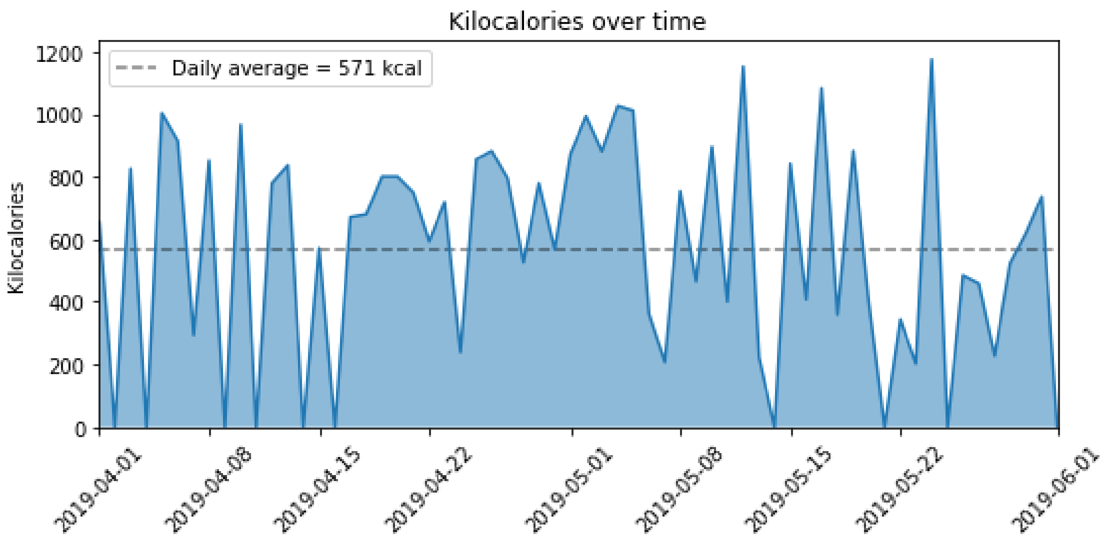
[37]:
```
dates = pd.date_range(start, stop)
dates = dates.to_frame()
dates = dates.reset_index(drop=True)
dates.columns = ['startTime']
```

[38]:
```
daily = pd.merge(dates, daily, on='startTime', how='left')
daily = daily.fillna(0)
```

Finally we produce the figure:

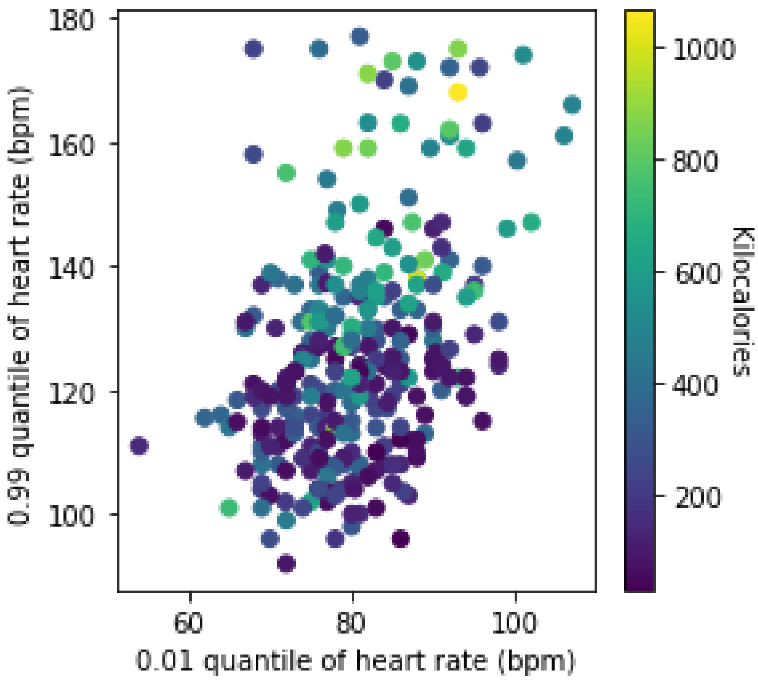
[39]:
```
width = 800
height = 400
dpi = 100

plt.figure(figsize=(width/dpi, height/dpi))
plt.plot(daily['startTime'], daily['kiloCalories'])

plt.fill_between(x=daily['startTime'],
                 y1=0,
                 y2=daily['kiloCalories'],
                 alpha=1/2)

daily_avg = daily['kiloCalories'].mean()

plt.hlines(xmin=daily['startTime'].min(),
           xmax=daily['startTime'].max(),
```

```
            y=daily_avg,
            linestyle='dashed',
            label=f'Daily average = {round(daily_avg)} kcal',
            alpha=1/2)

plt.title('Kilocalories over time')
plt.xticks(rotation=45, horizontalalignment='center')
plt.xlim(daily['startTime'].min(), daily['startTime'].max())
plt.ylim(0, daily['kiloCalories'].max() * 1.05)
plt.ylabel('Kilocalories')
plt.legend(loc='best')
plt.tight_layout()
plt.savefig('./img/kilocalories_ts.png')
plt.show()
```



8 Kilocalories by intensity

[40]:
```
plt.scatter(df['heartRateQ1'], df['heartRateQ99'], c=df['kiloCalories'])
plt.xlabel('0.01 quantile of heart rate (bpm)')
plt.ylabel('0.99 quantile of heart rate (bpm)')

cbar = plt.colorbar()
cbar.set_label('Kilocalories', rotation=270)
plt.savefig('./img/intensity_scatter.png')
plt.show()
```



8.1 Workouts by sport

We check how many workouts I completed.

[41]:
```
stats = df[['sport', 'startTime']]
stats = stats.groupby(['sport'], as_index=False)
stats = stats.count()
stats = stats.rename(columns={'sport': 'Sport',
                              'startTime': 'Count'})
stats = stats.sort_values('Count', ascending=False)
stats.head()
```

[41]:

	Sport	Count
4	WALKING	105
3	TREADMILL_RUNNING	90
2	STRENGTH_TRAINING	62
0	CYCLING	24
1	RUNNING	2

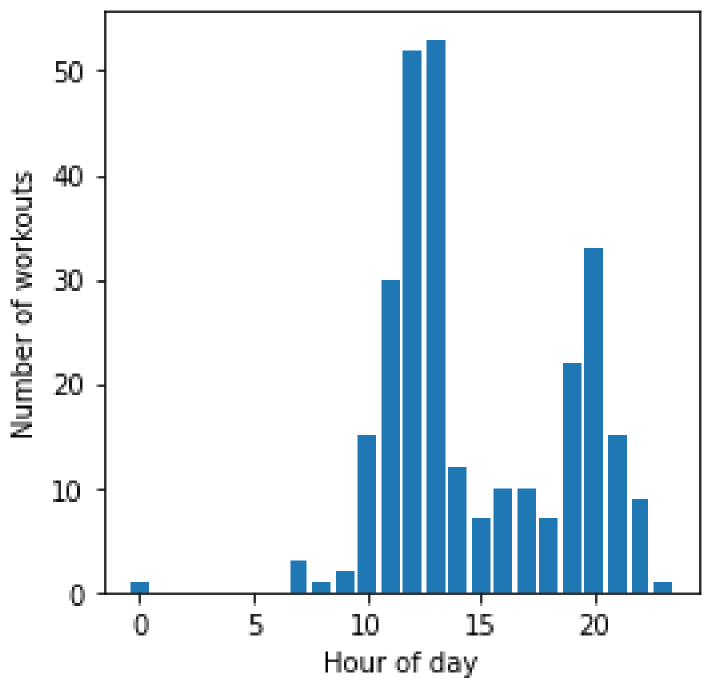
8.2 By hour of day

We count workouts by hour of day.

[42]:
```
by_hour = df[['startTime', 'sport']].copy()
by_hour['startHour'] = by_hour['startTime'].dt.hour
```

```
by_hour = by_hour.drop('startTime', axis=1)
by_hour = by_hour.groupby('startHour', as_index=False)
by_hour = by_hour.count()

all_hours = pd.DataFrame(range(0, 24), columns=['startHour'])

by_hour = pd.merge(all_hours, by_hour, how='left')
by_hour = by_hour.fillna(0)
by_hour = by_hour.sort_values('startHour')
by_hour = by_hour.rename(columns={'startHour': 'Hour of day',
                                  'sport': 'Total workouts'})
```

[43]:
```
plt.bar(by_hour['Hour of day'], by_hour['Total workouts'])
plt.ylabel('Number of workouts')
plt.xlabel('Hour of day')
plt.savefig('./img/workouts_by_hour_of_day.png')
plt.show()
```



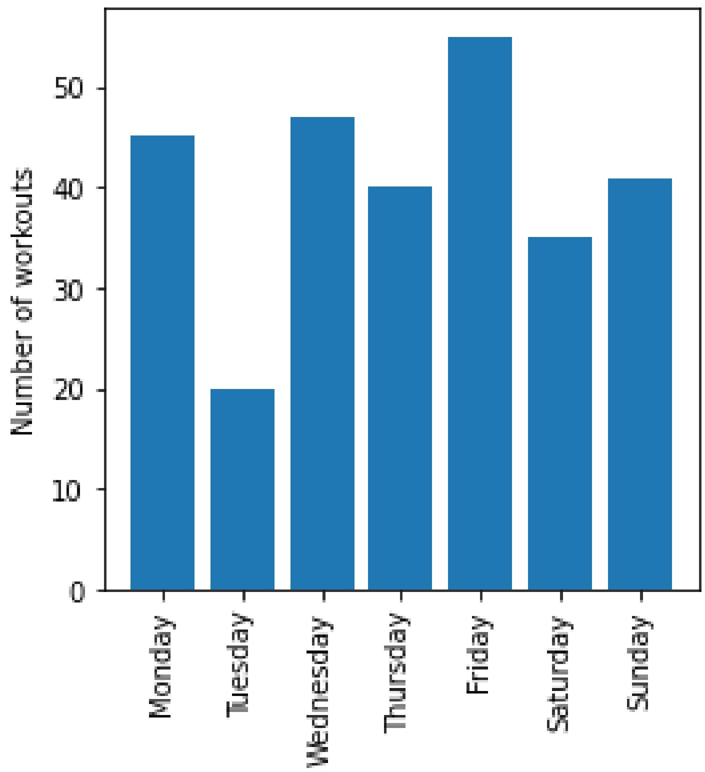
8.3 By day of week

We count workouts by day of week.

[44]:
```
by_day = df[['startTime', 'sport']].copy()
by_day['Day of week'] = pd.to_datetime(by_day['startTime']).dt.day_name()
by_day['Day number'] = pd.to_datetime(by_day['startTime']).dt.dayofweek
```

```
by_day = by_day.groupby(['Day of week', 'Day number'], as_index=False)
by_day = by_day.count()
by_day = by_day.drop('startTime', axis=1)
by_day = by_day.sort_values('Day number')
by_day = by_day.rename(columns={'sport': 'Total Workouts'})
```

[45]:
```
plt.bar(by_day['Day of week'], by_day['Total Workouts'])
plt.xticks(rotation=90)
plt.ylabel('Number of workouts')
plt.savefig('./img/workouts_by_day_of_week.png')
plt.show()
```

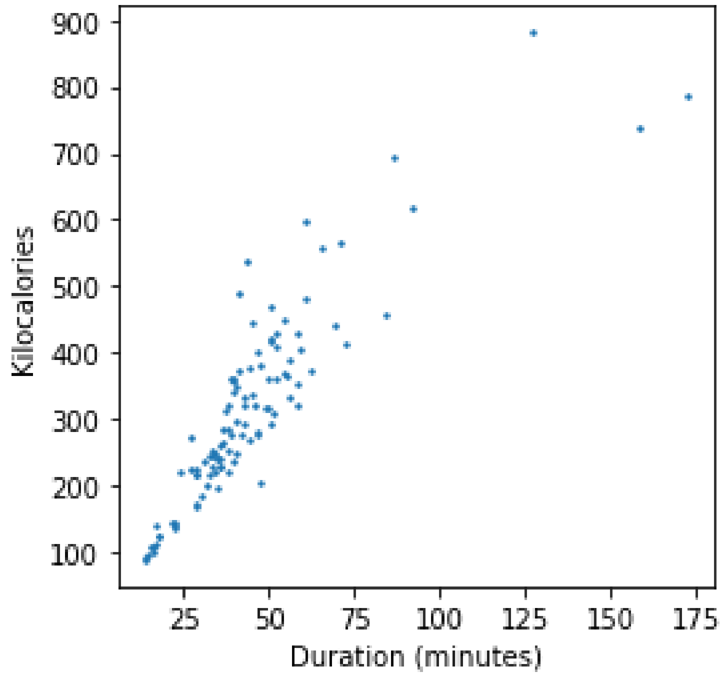


8.4 Scatter plot of walks data

We plot `totalTime` versus `kiloCalories`. As can be seen their seems to exist a linear relationship between the two.

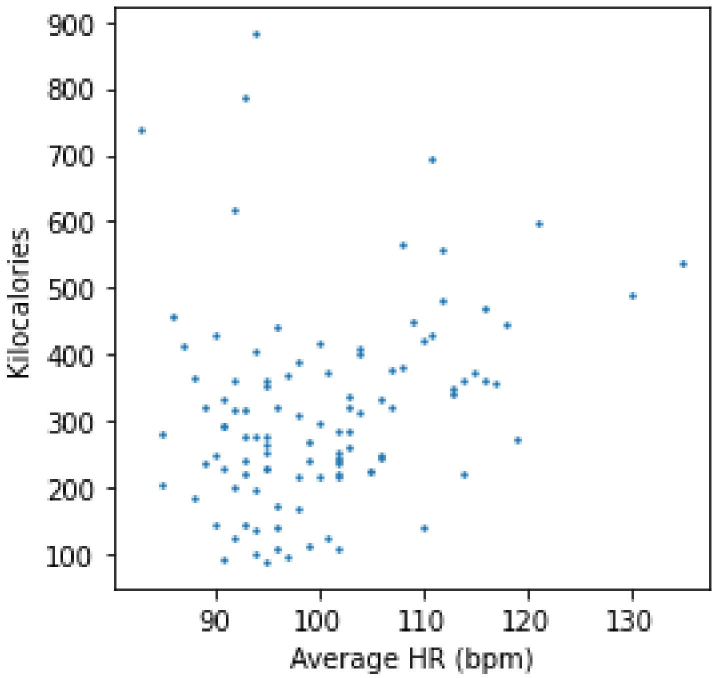
[46]:
```
walking = df[df['sport'] == 'WALKING']
plt.scatter(walking['totalTime'], walking['kiloCalories'], s=2)
plt.xlabel('Duration (minutes)')
plt.ylabel('Kilocalories')
plt.savefig('./img/walks_kilocalories_vs_time.png')
```

```
plt.show()
```



We plot `heartRateAvg` against `kiloCalories`. Again we see a linear relationship although there are a couple of outliers

[47]:
```
walking = df[df['sport'] == 'WALKING']
plt.scatter(walking['heartRateAvg'], walking['kiloCalories'], s=2)
plt.ylabel('Kilocalories')
plt.xlabel('Average HR (bpm)')
plt.savefig('./img/walks_kilocalories_vs_avg_hr.png')
plt.show()
```



9 Regression

9.1 Data preparation

Now we proceed to build a regression model to predict `kiloCalories` burned during a workout. First we create a subset of the original data.

[48]:
```
reg_df = df[['kiloCalories', 'totalTime',
             'heartRateQ99', 'activityType',
             'indoors', 'sport']].copy()
```

We remove the rows where `sport` is `RUNNING` because there were only two workouts recorded during the period in question.

[49]:
```
reg_df = reg_df[reg_df['sport'] != 'RUNNING']
reg_df = reg_df.drop('sport', axis=1)
```

We convert binary features to integers for statsmodels.

[50]:
```
reg_df['indoors'] = reg_df['indoors'].astype(int)
reg_df['activityType'] = reg_df['activityType'].astype(int)
```

9.1.1 Outliers

The data is cleansed of outliers using interquartile range.

[51]:
```python
def is_outlier_iqr(series, k=1.5):
    """
    Check if value is an outlier
    using interquartile range.
    """

    q1 = series.quantile(0.25)
    q3 = series.quantile(0.75)
    iqr = q3 - q1
    is_outlier = (series <= q1 - k * iqr) | (q3 + k * iqr <= series)

    return is_outlier
```

[52]:
```python
time_mask = is_outlier_iqr(series=reg_df['totalTime'])
kcal_mask = is_outlier_iqr(series=reg_df['kiloCalories'])
hr_mask = is_outlier_iqr(series=reg_df['heartRateQ99'])
```

[53]:
```python
reg_df = reg_df[~(time_mask | kcal_mask | hr_mask)]
```

9.1.2 Transforms

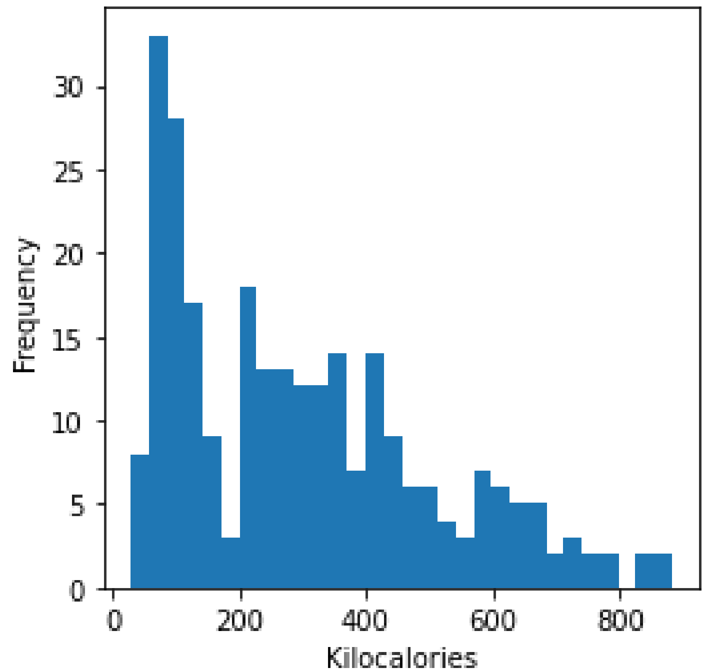
We apply the log transform to `heartRateQ99` hoping to reduce variance.

[54]:
```python
reg_df['heartRateQ99'] = np.log(reg_df['heartRateQ99'])
```

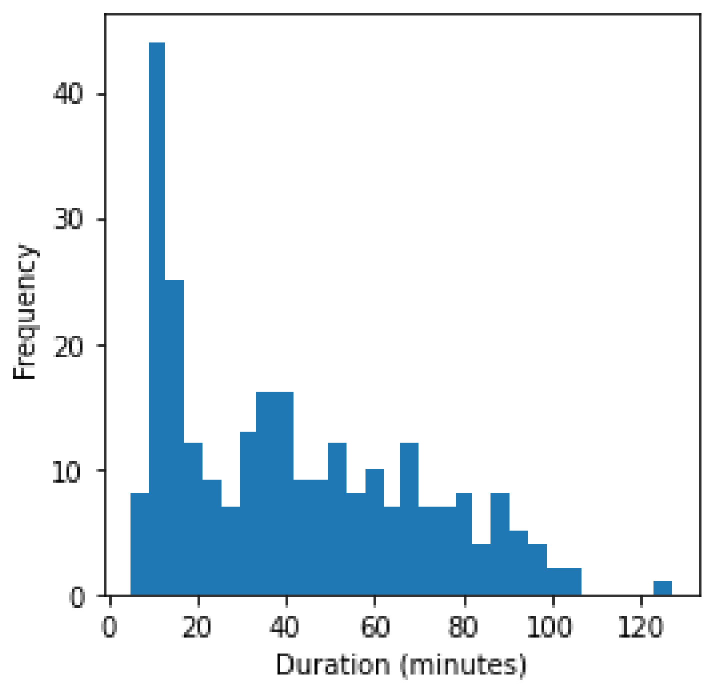
9.1.3 Histograms of features

We proceed to visualize histograms of each of the variables.

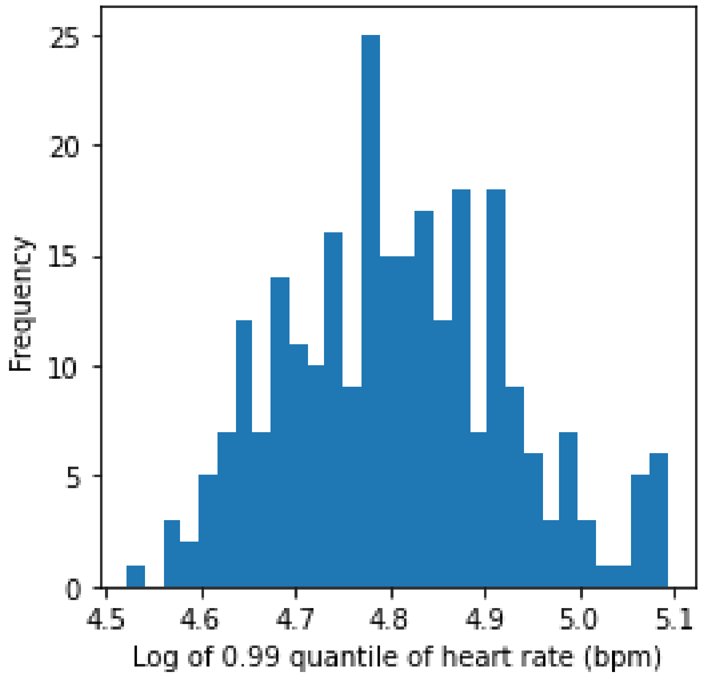
[55]:
```python
plt.hist(reg_df['kiloCalories'], bins=30)
plt.xlabel('Kilocalories')
plt.ylabel('Frequency')
plt.savefig('./img/kilocalories_histogram.png')
plt.show()
```



[56]:
```
plt.hist(reg_df['totalTime'], bins=30)
plt.xlabel('Duration (minutes)')
plt.ylabel('Frequency')
plt.savefig('./img/duration_histogram.png')
plt.show()
```



[57]:
```
plt.hist(reg_df['heartRateQ99'], bins=30)
plt.xlabel('Log of 0.99 quantile of heart rate (bpm)')
plt.ylabel('Frequency')
plt.savefig('./img/average_hr_histogram.png')
plt.show()
```

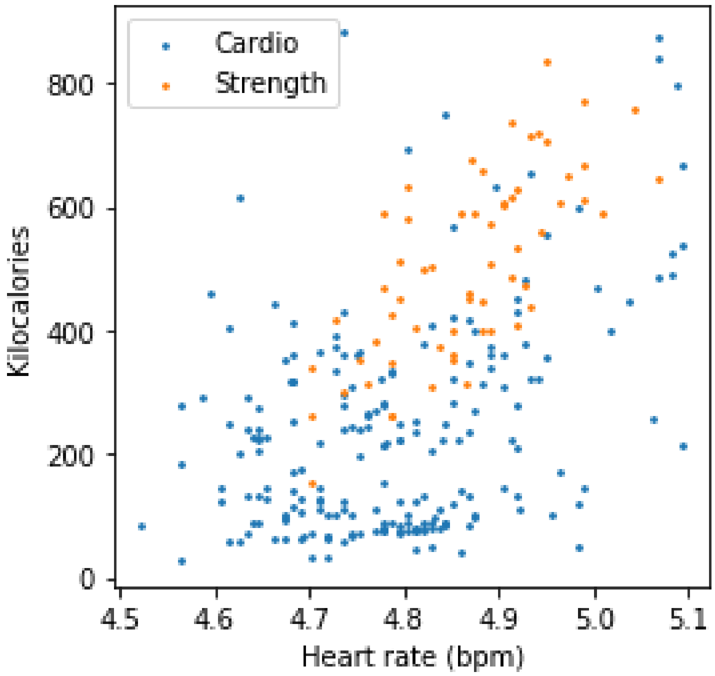


9.1.4 Relationship with target variable

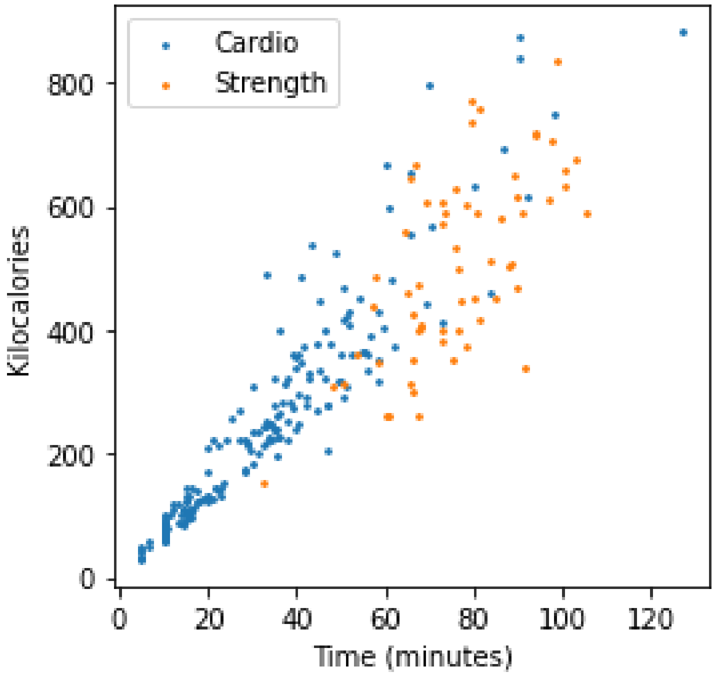
[58]:
```
activity = [0, 1]
activity_type = ['Cardio', 'Strength']
```

[59]:
```
for val,name in zip(activity,activity_type):
    tmp = reg_df[reg_df['activityType'] == val]
    plt.scatter(tmp['heartRateQ99'],
                tmp['kiloCalories'],
                s=3,
                label=name)

plt.xlabel('Heart rate (bpm)')
plt.ylabel('Kilocalories')
plt.legend(loc='best')
plt.savefig('./img/time_vs_kilocalories_scatter.png')
```

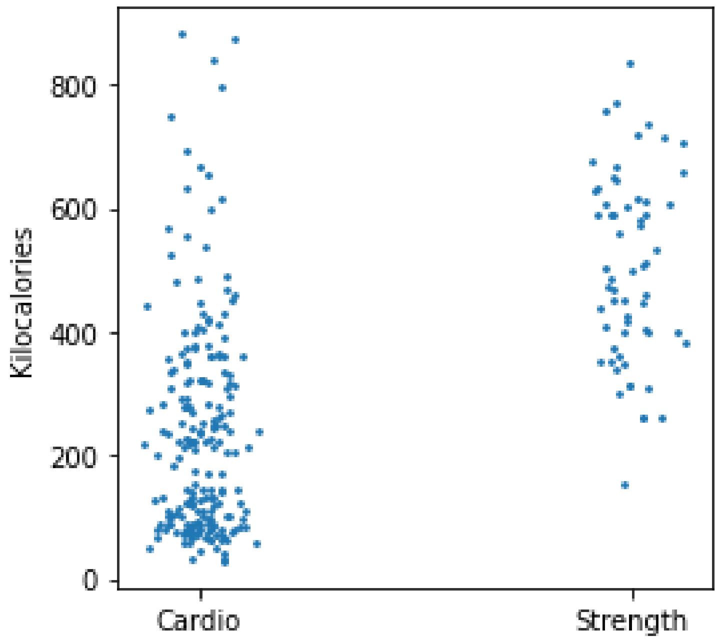
```
plt.show()
```



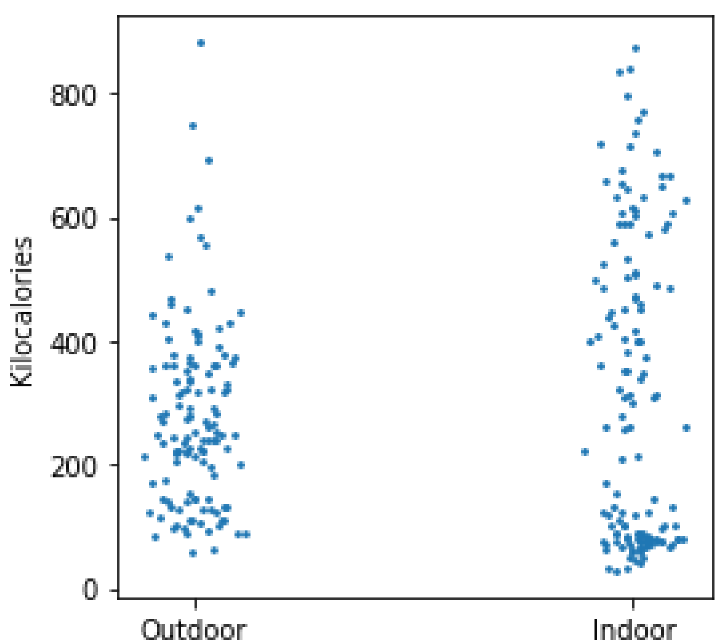
[60]:
```
for val,name in zip(activity,activity_type):
    tmp = reg_df[reg_df['activityType'] == val]
    plt.scatter(tmp['totalTime'],
                tmp['kiloCalories'],
                s=3,
                label=name)

plt.xlabel('Time (minutes)')
plt.ylabel('Kilocalories')
plt.legend(loc='best')
plt.savefig('./img/time_vs_kilocalories_scatter.png')
plt.show()
```



[61]:
```
plt.scatter(reg_df['activityType'] + np.random.normal(scale=1/20,␣
↪size=len(reg_df)),
            reg_df['kiloCalories'], s=3)
plt.ylabel('Kilocalories')
plt.xticks(ticks=[0, 1], labels=['Cardio', 'Strength'])
plt.savefig('./img/activity_type_vs_kilocalories_scatter.png')
plt.show()
```



[62]:
```
plt.scatter(reg_df['indoors'] + np.random.normal(scale=1/20, size=len(reg_df)),
            reg_df['kiloCalories'], s=3)
plt.ylabel('Kilocalories')
plt.xticks(ticks=[0, 1], labels=['Outdoor', 'Indoor'])
plt.savefig('./img/indoors_type_vs_kilocalories_scatter.png')
plt.show()
```



9.2 Correlation

We inspect the correlation matrix to check for multicollinearity. It should be noted that the correlation between `kiloCalories` and `totalTime` is quite high and this to be expected.

[63]:
```
C = reg_df.corr(method='pearson')
C = C.style.background_gradient(cmap='coolwarm')
C = C.set_precision(2)
C = C.set_table_attributes('style="font-size: 15px"')
C
```

[63]:
```
<pandas.io.formats.style.Styler at 0x7f1570ee2ca0>
```

9.3 Multicollinearity

We inspect the respect variance inflation factors and are happy to see that all are below 10.

[64]:
```
tmp = reg_df.drop(['kiloCalories'], axis=1)

vifs = []
for i in range(tmp.shape[1]):
    vif = variance_inflation_factor(tmp.to_numpy(), i)
    vifs.append(round(vif, 2))

vifs = pd.DataFrame(vifs, index=tmp.columns, columns=['VIF'])
vifs = vifs.sort_values('VIF', ascending=False)
vifs = vifs.style.background_gradient(cmap='coolwarm')
vifs = vifs.set_precision(2)
vifs = vifs.set_table_attributes('style="font-size: 15px"')

vifs
```

[64]:
```
<pandas.io.formats.style.Styler at 0x7f1570f28280>
```

9.4 Model

The model we shall fit is:

$c_i = \beta_1 t_i + \beta_2 \ln h_i + \beta_3 a_i + \beta_4 d_i + \varepsilon_i$

Where: * c_i - The i-th `kiloCalories` value. * t_i - The i-th `totalTime` value. * h_i - The i-th `heartRateQ99` value. * a_i - The i-th `activityType` value. * d_i - The i-th `indoors` value.

And of course ε_i is a iid normal error.

We create the `X` and `y` dataframes.

[65]:
```python
y = reg_df[['kiloCalories']]
X = reg_df.drop(['kiloCalories'], axis=1)
```

[66]:
```python
X.head()
```

[66]:

	totalTime	heartRateQ99	activityType	indoors
0	33.33	4.74	0	0
1	46.52	4.88	0	0
2	45.15	4.79	0	0
3	47.65	4.82	0	0
4	39.67	4.95	0	0

[67]:
```python
y.head()
```

[67]:

	kiloCalories
0	245.00
1	401.00
2	336.00
3	380.00
4	358.00

We create the model using all of the prepared variables:

[68]:
```python
mdl = sm.OLS(y, X)
res = mdl.fit()

residuals = res.resid

print(res.summary())
```

```
                            OLS Regression Results
================================================================================
=======
Dep. Variable:           kiloCalories   R-squared (uncentered):
0.959
Model:                            OLS   Adj. R-squared (uncentered):
0.958
Method:                 Least Squares   F-statistic:
1528.
Date:                Wed, 30 Sep 2020   Prob (F-statistic):
1.02e-179
Time:                        17:20:55   Log-Likelihood:
-1513.0
No. Observations:                 265   AIC:
3034.
Df Residuals:                     261   BIC:
3048.
Df Model:                           4
```

```
Covariance Type:            nonrobust
=================================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
totalTime       7.9121      0.252     31.374      0.000       7.415       8.409
heartRateQ99   -5.4471      2.482     -2.195      0.029     -10.334      -0.560
activityType -121.9051     19.138     -6.370      0.000    -159.589     -84.221
indoors        48.1236     11.888      4.048      0.000      24.715      71.532
==============================================================================
Omnibus:                       33.706   Durbin-Watson:                   1.795
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               76.750
Skew:                           0.615   Prob(JB):                     2.16e-17
Kurtosis:                       5.332   Cond. No.                         232.
==============================================================================

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

[69]:
```python
with open('mdl_results.txt', 'w') as f:
    text = res.summary().as_text()
    f.write(text)
```

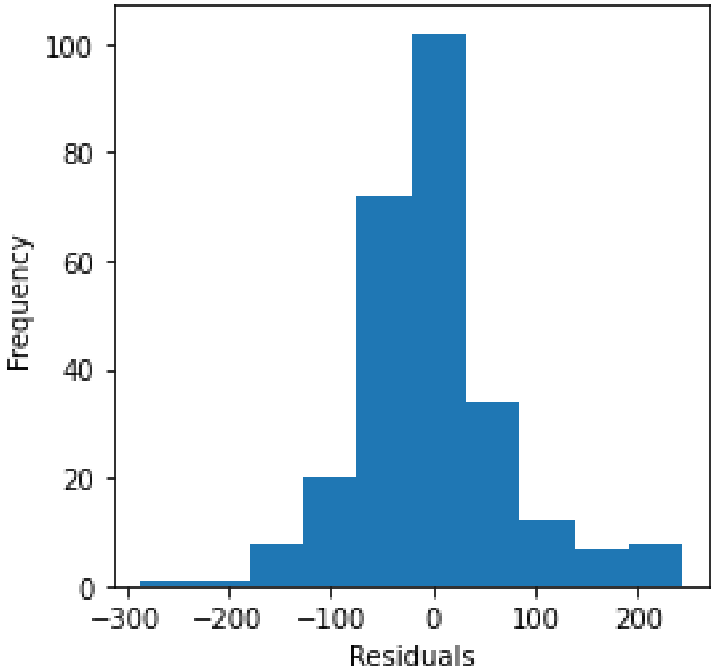
9.5 Model evaluation

It should be noted that the model achieved `R^2 = 0.96` and that the Durbin-Watson test statistic is slightly less than 2.

9.5.1 Visual inspection

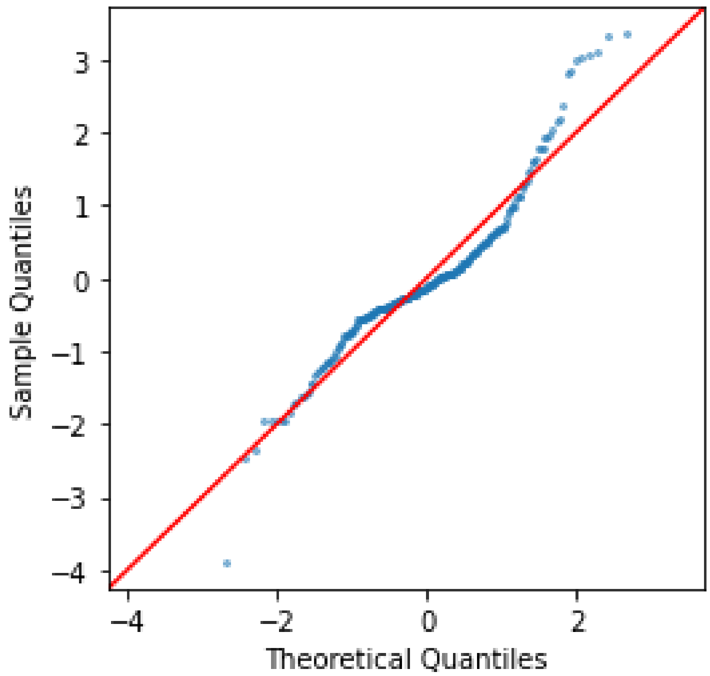
We proceed to inspect the residuals of the model. First we view the histogram of the residuals.

[70]:
```python
plt.hist(residuals)
plt.ylabel('Frequency')
plt.xlabel('Residuals')
plt.savefig('./img/mdl_residuals.png')
plt.show()
```



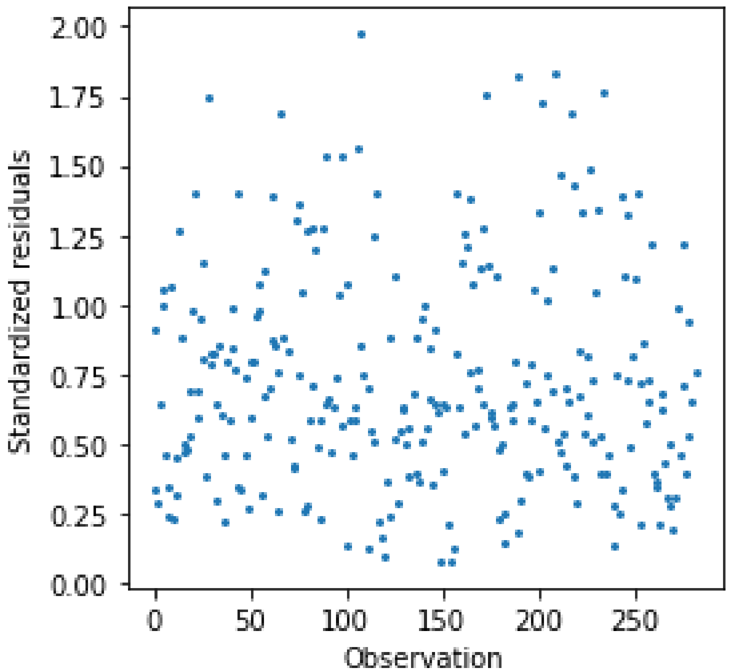
The next plot is a qqplot created to visually inspect the normality of the residuals.

[71]:
```
plt.figure()
ax = plt.gca()
qqplot(res.resid, ax=ax, color='#1f77b4', markersize=2, line='45', fit=True,␣
 ↪alpha=1/2)
plt.savefig('./img/mdl_qq.png')
plt.show()
```



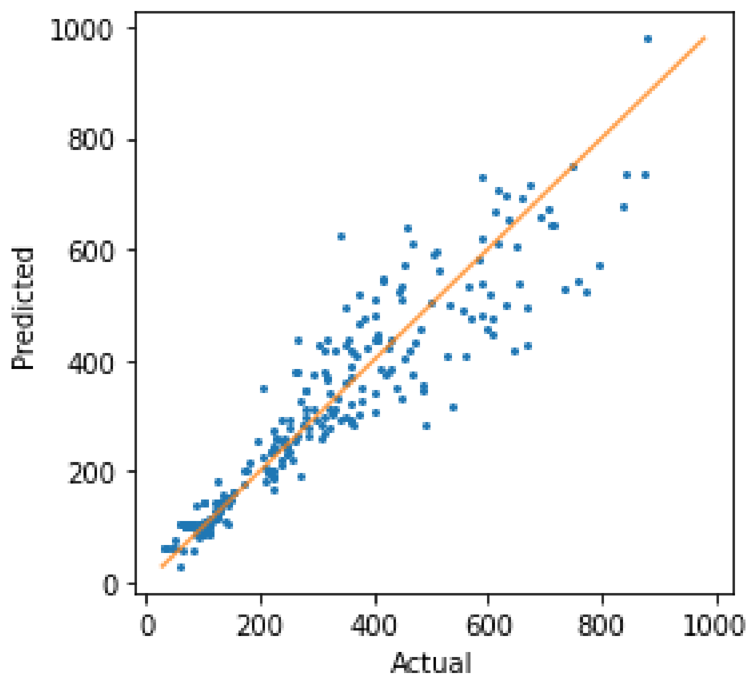
The third plot we make is a plot of the standardized residuals to check for homoskedasticity.

[72]:
```
residuals_std = np.sqrt(np.abs((residuals - np.mean(residuals)) / np.
 ↪std(residuals)))
plt.plot(residuals_std, 'o', markersize=2)
plt.xlabel('Observation')
plt.ylabel('Standardized residuals')
plt.savefig('./img/mdl_residuals_std.png')
plt.show()
```



Finally we compare the predicted `kiloCalories` with the actual values.

[73]:
```
y_pred = res.predict(X)
y_pred = y_pred.to_numpy().reshape(len(y_pred))
y_true = y.to_numpy().reshape(len(y),)

m = np.min(np.hstack([y_true, y_pred]))
M = np.max(np.hstack([y_true, y_pred]))

x = np.linspace(m, M, len(y))
plt.plot(y_true, y_pred, 'o', markersize=2)
plt.plot(x,x, alpha=3/4)
plt.ylabel('Predicted')
plt.xlabel('Actual')
plt.savefig('./img/mdl_predicted_vs_actual.png')
plt.show()
```



The next step is to take a look at the datapoints with the biggest error.

[74]:
```
X['totalTime'] = reg_df['totalTime']
X['heartRateQ99'] = reg_df['heartRateQ99']
X['y_true'] = y
X['y_pred'] = y_pred
X['e'] = X['y_true'] - X['y_pred']
X = X.sort_values('e', ascending=False)
```

[75]:
```
X.head()
```

[75]:

	totalTime	heartRateQ99	activityType	indoors	y_true	y_pred	e
209	79.25	4.99	1	1	770.00	526.07	243.93
190	66.65	4.99	1	1	667.00	426.37	240.63
234	65.75	5.07	1	1	644.00	418.83	225.17
173	69.65	5.09	0	1	795.00	571.49	223.51
28	43.33	5.09	0	0	536.00	315.08	220.92

9.5.2 Statistical tests

We now move on to performing statistical tests for normality and homoskedasticity.

[76]:
```
def hypothesis_decision(x, alpha=0.05):
    if x < alpha:
        return 'Reject null hypothesis'
```

```python
    else:
        return 'Fail to reject null hypothesis'
```

First we carry out the Shapiro-Wilks test for normality. The hypotheses are:

H_0 : Data comes from a normal distribution.

H_1 : Data does not come from a normal distribution.

[77]:
```python
_,shapiro_pval = shapiro(residuals)
```

The second test we perform is the Breusch-Pagan for homoscedasticity. The hypotheses are:

H_0 : Homoscedasticity.

H_1 : Lack of homoscedasticity / Heteroskedasticity.

[78]:
```python
_,_,_,breusch_pval = het_breuschpagan(residuals, X)
```

We summarize the outcomes of the tests in a nice table.

[79]:
```python
pvalues = [shapiro_pval, breusch_pval]
name = ['Shapiro-Wilks', 'Breusch-Pagan',]
null_hyp = ['Normality', 'Heteroskedasticity']

tests = pd.DataFrame([name, null_hyp, pvalues],
                     index=['Name', 'Null Hypothesis', 'P-value'])

tests = tests.transpose()
tests['Decision'] = tests['P-value'].apply(hypothesis_decision)
tests
```

[79]:

	Name	Null Hypothesis	P-value	Decision
0	Shapiro-Wilks	Normality	0.00	Reject null hypothesis
1	Breusch-Pagan	Heteroskedasticity	0.00	Reject null hypothesis

10 Estimated model

The estimated model is:

$$c = 7.91t - 5.44 \ln h - 121.90a + 48.12d \tag{1}$$

Where: * c denotes `kiloCalories`. * t denotes`totalTime`. * h denotes `heartRateQ99`. * a denotes `activityType`. * d denotes `indoors`.

11 Summary

- I downloaded data generated by my Polar watch that tracks `heart rate` and estimates burned `kilocalories` during workouts.

- The data came in the form of `.json` files which were read, transformed and cleaned with `pandas`.
- The clean dataset contains `283` workouts over a nearly one year period.
- After further transforming the data, I find that the `duration` of a workout and `kilocalorie`s burned have a `0.92` correlation.
- A linear regression model was built using `statsmodels` to predict the total `kilocalorie` expenditure.
- Variables in the model are the `duration` of the session, the `99 quantile of heart rate`, the `activity` type and whether the workout is `indoors`.
- The model achieves an adjusted uncentered `R^2` of `0.958`.
- The estimated formula is: `calories = 7.91 * time - 5.44 * log(heart_rate) - 121.90 * activity_type + 48.12 * indoors`.
- All of the variables are statistically significant.
- The model has a tendency to underpredict long workouts.
- The residuals fail statisticall tests for normality and homoskedasticity.