

linear_regression_XYZ_MET-walking

January 12, 2021

```
[1]: from helpers import pandas_helper as pdh
from helpers import math_helper as mth
from sensors.activpal import *
from utils import read_functions

from scipy.stats import linregress
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import RFE
import seaborn as sns

import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import datetime

activpal = Activpal()

activity_focus = 'lopen'
respondents = ['BMR002', 'BMR011', 'BMR012', 'BMR014', 'BMR018', 'BMR030',
↳ 'BMR031', 'BMR032', 'BMR033', 'BMR034', 'BMR036', 'BMR040', 'BMR041',
↳ 'BMR042', 'BMR043', 'BMR044', 'BMR052', 'BMR053', 'BMR055', 'BMR058',
↳ 'BMR064', 'BMR098']
test_respondents = ['BMR004', 'BMR008', 'BMR097']
```

1 Defining functions

1.0.1 Method to retrieve DataFrame containing all information for regression

```
[2]: def get_regression_df(respondent, activity):
    start, stop = get_timestamps(respondent, activity)

    # read in all dataframes necessary
```

```

respondents_df = pdh.read_csv_respondents()
res_number = get_respondent_number(respondent)
vyntus_df, min_index, max_index = get_vyntus_df(respondent,
→respondents_df['gewicht'][res_number], start, stop)
raw_df = get_raw_df(respondent, min_index, max_index)

# add met and mag acc to new dataframe
new_df = pd.DataFrame(index=raw_df.index)
new_df['mean_met'] = vyntus_df['met']
new_df['sum_mag_acc'] = raw_df['sum_mag_acc']

# add features to new dataframe
new_df['length_cm'] = respondents_df['lengte'][res_number]
new_df['weight_kg'] = respondents_df['gewicht'][res_number]
new_df['mean_speed'] = raw_df['mean_speed']
new_df['bmi'] = mth.calculate_bmi(new_df['weight_kg'], new_df['length_cm'])
new_df['gender'] = int(respondents_df['geslacht'][res_number]).
→replace('vrouw',str(0)).replace('man', str(1))
new_df['age_category'] = respondents_df['leeftijdscategorie'][res_number]
convert_age_to_number(new_df, "age_category")
new_df['is_sporter'] = respondents_df['sporter'][res_number]
new_df['meets_activity_guidelines'] = int(respondents_df['voldoet aan
→beweegerichtlijn 2nee17'][res_number].replace('ja', '1').replace('nee', '0'))
new_df['does_muscle_bone_exercises'] = int(respondents_df['voldoet aan
→richtlijn bot en spierversterkende activiteiten'][res_number].replace('ja',
→'1').replace('nee', '0'))
new_df['meets_balance_guidelines'] = int(respondents_df['voldoet aan
→richtlijn balansoefeningen'][res_number].replace('ja', '1').replace('nee',
→'0'))

return new_df

def get_regression_dfs(respondents, activity):
    all_df = pd.DataFrame(index=pd.to_datetime([]))

    for cor in respondents:
        df = get_regression_df(cor, activity)
        all_df = pd.concat([all_df, df])

    all_df.sort_index(inplace=True)
    return all_df

```

1.0.2 Helper method, returning start and stop timestamps for an activity

```
[3]: def get_timestamps(respondent, activity):
    activities_df = read_functions.read_activities(respondent)
    start = activities_df.loc[activity].start
    stop = activities_df.loc[activity].stop

    return (start, stop)
```

1.0.3 Helper method, returning the number of respondent code

```
[4]: def get_respondent_number(respondent):
    if ('BMRO' in respondent):
        return int(respondent.replace('BMRO', ''))

    if ('BMR' in respondent):
        return int(respondent.replace('BMR', ''))
```

1.0.4 Helper method, returning Vyntus (lab data)

The DataFrame contains MET and other information necessary for regression

The DataFrame is resampled to minutes by mean

```
[5]: def get_vyntus_df(respondent, weight, start, stop):
    vyntus_df = pdh.read_csv_vyntus(respondent)
    mask = (vyntus_df.index >= start) & (vyntus_df.index < stop)
    vyntus_df = vyntus_df.loc[mask]

    min_index = vyntus_df.index.min()
    max_index = vyntus_df.index.max()

    vyntus_df['vyn_V02'] = [float(vo2.replace(',', '.')) if type(vo2) == str
    ↪ else vo2 for vo2 in vyntus_df['vyn_V02']]
    vyntus_df['met'] = mth.calculate_met(vyntus_df['vyn_V02'], weight)

    vyntus_df = vyntus_df.resample('60s').mean()[:-1]

    return (vyntus_df, min_index, max_index)
```

1.0.5 Helper method, returning Activpal20

The DataFrame contains sumation of magnitude of acceleration

The DataFrame is resampled to minutes by sumation

```
[6]: def get_raw_df(respondent, start, stop):
    df = activpal.read_data(respondent, start, stop)

    mask = (df.index >= start) & (df.index < stop)
    df = df.loc[mask]

    df = df[['pal_accX', 'pal_accY', 'pal_accZ']].apply(mth.convert_value_to_g)
    df['sum_mag_acc'] = mth.to_mag_acceleration(df['pal_accX'], df['pal_accY'],
    ↪df['pal_accZ'])
    df['mean_speed'] = get_speed(df['pal_accX'], df['pal_accY'], df['pal_accZ'])

    df = df.resample('60s').agg({'sum_mag_acc': 'sum', 'mean_speed': 'mean'})[:-1]

    return df
```

1.0.6 Helper method, from Colin

```
[7]: def convert_age_to_number(dataframe, age_category):
    age_conversion = {
        age_category: {
            "15-19": 0, "20-24": 1, "25-29": 2, "30-34": 3,
            "35-39": 4, "40-44": 5, "45-49": 6, "50-54": 7,
            "55-59": 8, "60-64": 9, "65-69": 10, "70-74": 11, "75-79": 12
        }
    }
    return dataframe.replace(age_conversion, inplace=True)
```

1.0.7 Helper method, returns speed, from Adnan

```
[8]: import scipy.integrate as it

def get_speed(x_acc, y_acc, z_acc):
    activpal_time = 0.05

    x_vel = np.array([sum(x_acc[:i]) * activpal_time for i in
    ↪range(len(x_acc))])
    y_vel = np.array([sum(y_acc[:i]) * activpal_time for i in
    ↪range(len(y_acc))])
    z_vel = np.array([sum(z_acc[:i]) * activpal_time for i in
    ↪range(len(z_acc))])

    return np.sqrt(x_vel ** 2 + y_vel ** 2 + z_vel ** 2)
```

1.0.8 Helper method, returns best features out of all features

```
[9]: def get_best_features(df, feature_columns, n_features_to_select):
    x_train, x_valid, y_train, y_valid = train_test_split(df[feature_columns],
    ↪df['mean_met'], test_size=0.2, random_state=0)

    estimator = LinearRegression()
    selector = RFE(estimator, n_features_to_select=n_features_to_select)
    selector = selector.fit(x_train, y_train)

    return x_train.columns[selector.support_]
```

1.0.9 Helper methods to plot various graphs

```
[10]: def plot_met(met, title = 'MET while walking'):
    plt.ylabel('MET')
    plt.plot(met, marker = 'o')
    plt.title(title)
    plt.xticks(rotation=70)

def plot_mag_acc(mag_acc, title = 'sum_mag_acc while walking'):
    plt.ylabel('Sum magnitude acceleration, g')
    plt.plot(mag_acc, marker = 'o')
    plt.title(title)
    plt.xticks(rotation=70)
    plt.yticks(rotation=60)

def plot_mean_speed(mean_speed, title = 'mean_speed while walking'):
    plt.ylabel('Mean speed, g')
    plt.plot(mean_speed, marker = 'o')
    plt.title(title)
    plt.xticks(rotation=70)
    plt.yticks(rotation=60)

def plot_lin_reg(x, y, xlabel = 'acceleration, g', ylabel = 'MET'):
    linreg = linregress(x, y)
    fx = np.array([x.min(), x.max()])
    fy = linreg.intercept + linreg.slope * fx

    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.plot(x, y, 'o')
    plt.plot(fx, fy, '-')

def plot_heatmap(df):
```

```

plt.figure(figsize=(15,10))
sns.heatmap(df.corr(), annot=True, cmap=plt.cm.Reds)
plt.show()

def plot_pred_truth(pred_y, valid_y, title):
    plt.plot(range(len(pred_y)), pred_y, label='prediction', marker='o')
    plt.plot(range(len(valid_y)), valid_y, label='ground_truth', marker='o')

    plt.title(title)
    plt.ylabel('met')
    plt.xlabel('Prediction Number')
    plt.legend()

def plot_ground_truth_vs_prediction(pred_y, valid_y, title='Predictions on_
↪ validation dataset'):
    plt.figure(figsize=(20,10))
    bar_width = 0.35

    pred_index = np.arange(len(pred_y))
    y_index = np.arange(len(valid_y)) + bar_width
    plt.bar(pred_index, pred_y, bar_width, label='Prediction')
    plt.bar(y_index, valid_y, bar_width, label='Ground Truth')

    plt.xlabel('Prediction Number')
    plt.ylabel("MET")
    plt.title(title)
    plt.xticks(pred_index + 0.15, pred_index)
    plt.legend()
    plt.grid()
    plt.show()

```

2 Univariate linear regression

2.0.1 One respondent

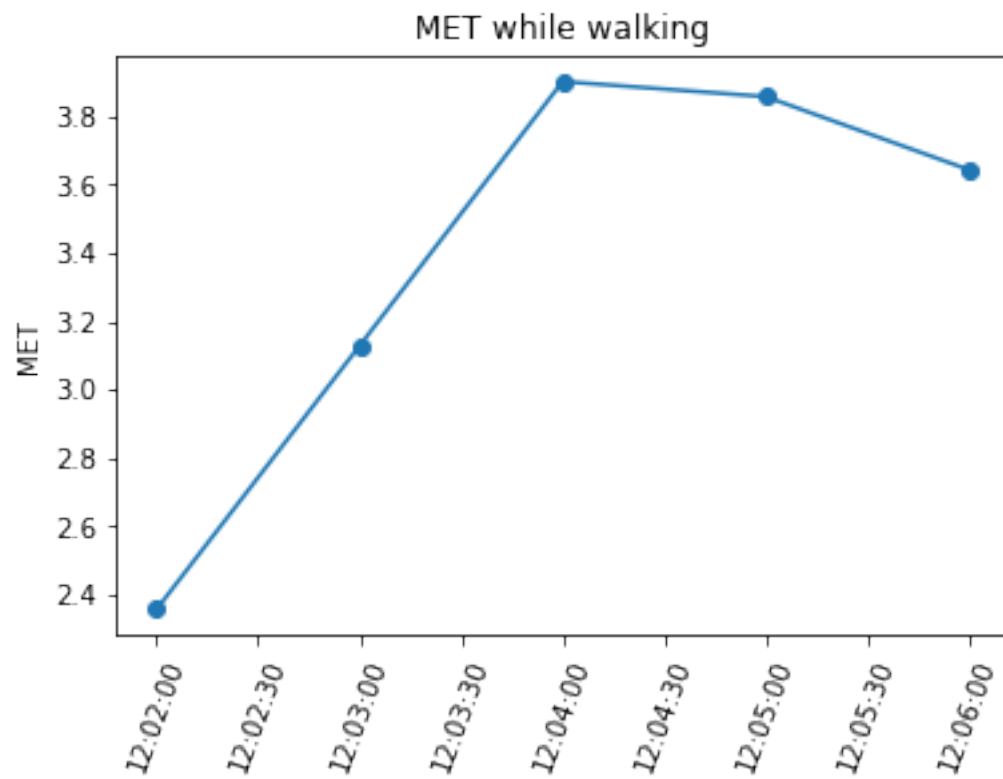
```

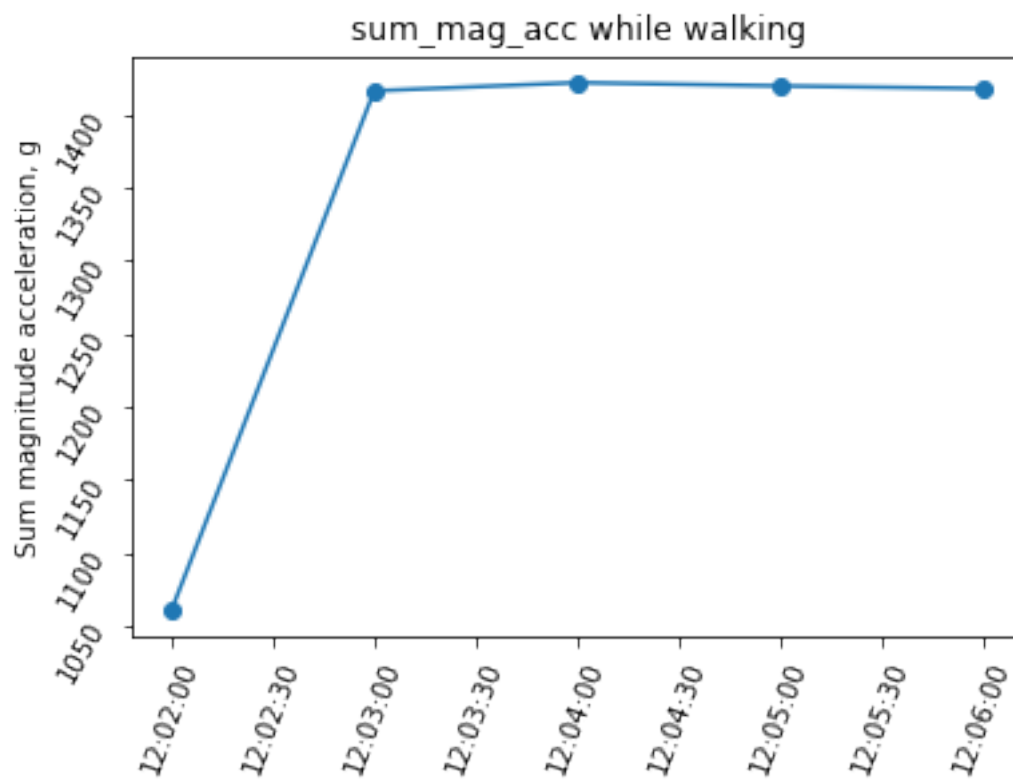
[11]: res30 = 'BMR011'
df30 = get_regression_df(res30, activity_focus)

plot_met(df30['mean_met'])
plt.show()
plot_mag_acc(df30['sum_mag_acc'])
plt.show()
plot_mean_speed(df30['mean_speed'])
plt.show()

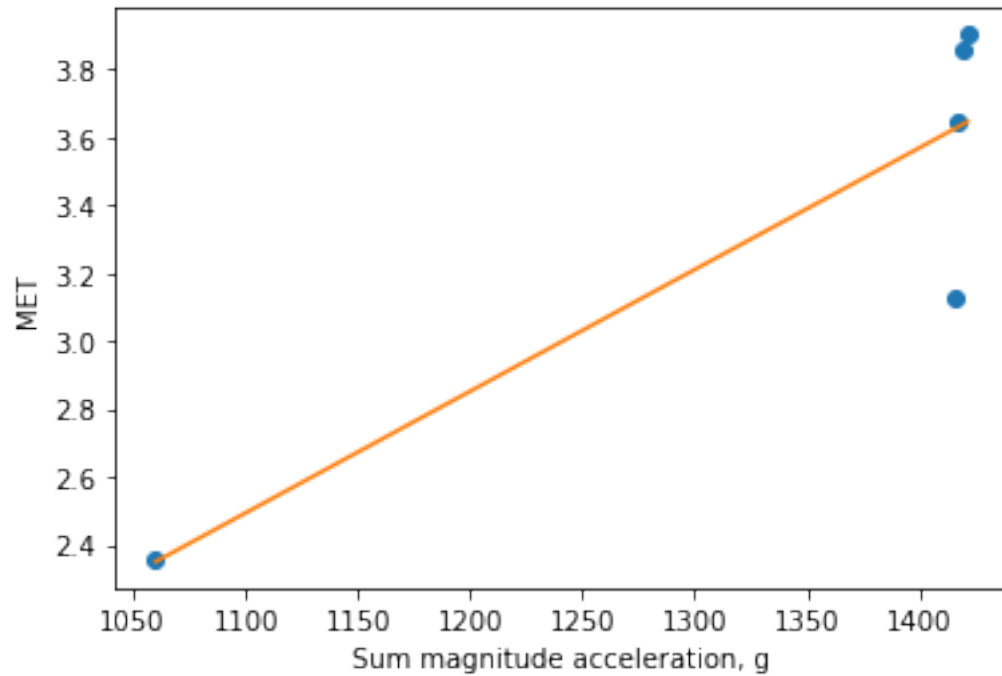
```

```
print('r = ' + str(df30['sum_mag_acc'].corr(df30['mean_met'])))  
plot_lin_reg(df30['sum_mag_acc'], df30['mean_met'], xlabel='Sum magnitude_  
↪ acceleration, g')
```





$r = 0.8859527619029444$

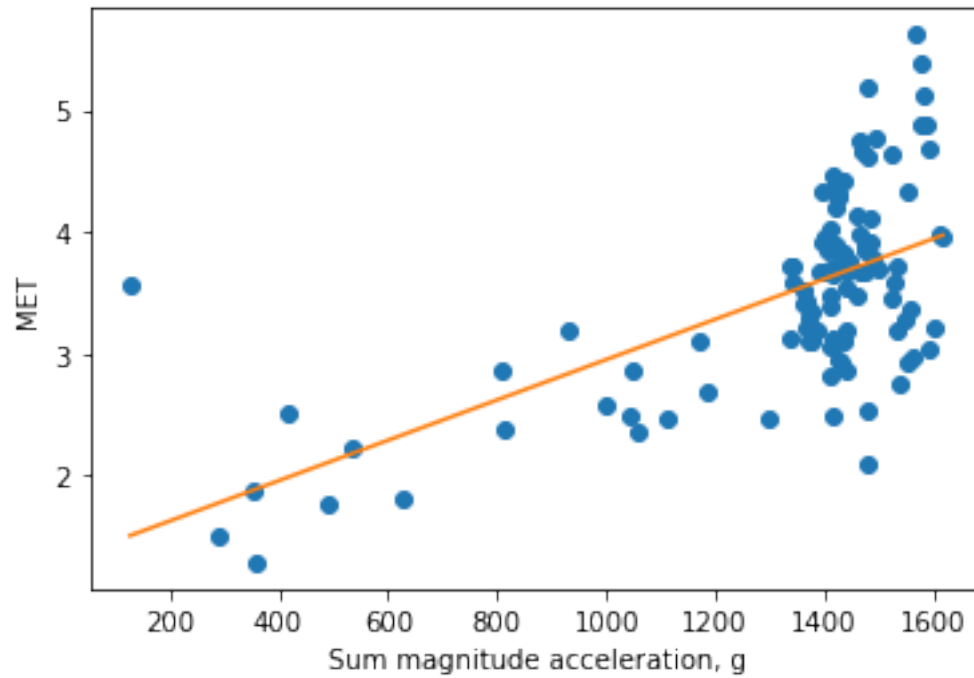


2.0.2 All respondents

```
[12]: all_df = get_regression_dfs(respondents, activity_focus)
test_df = get_regression_dfs(test_respondents, activity_focus)

plot_lin_reg(all_df['sum_mag_acc'], all_df['mean_met'], xlabel='Sum magnitude_
↪acceleration, g')
all_df.shape
```

[12]: (106, 12)

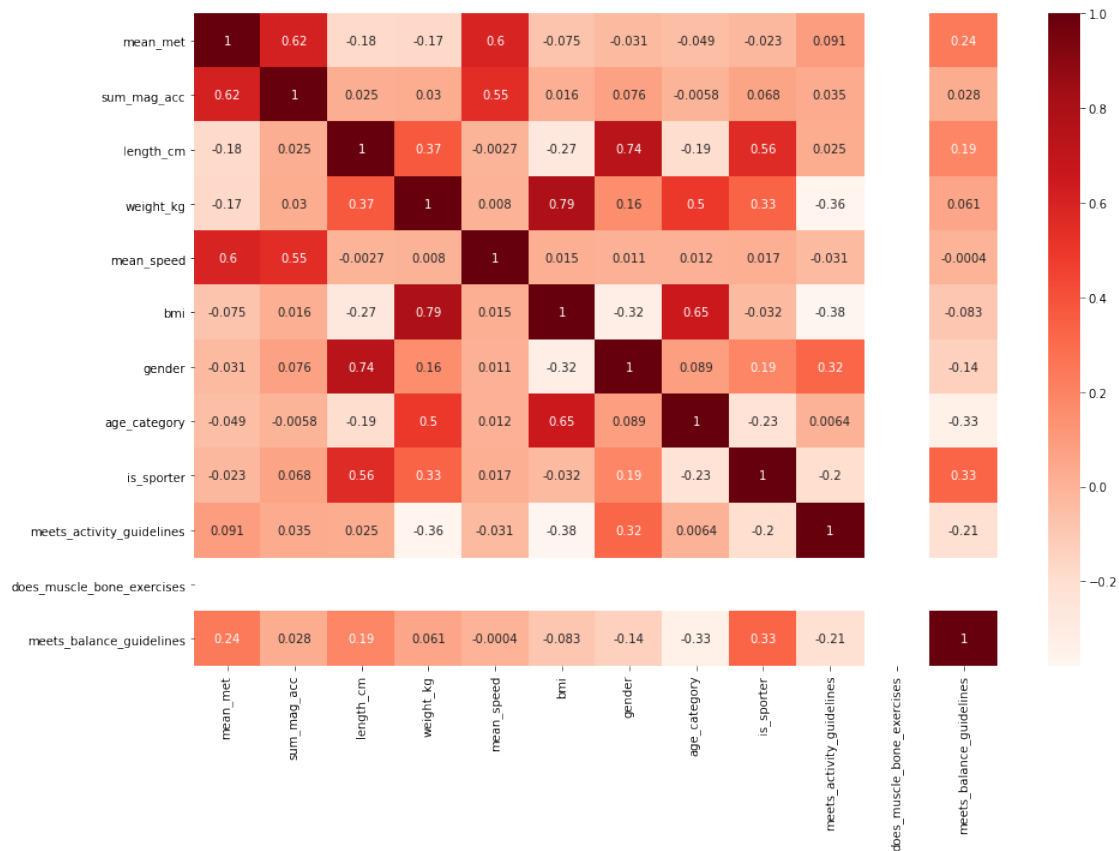


3 Multivariate Linear regression model training

3.1 Dimensionality reduction

3.1.1 Pearson Correlation confusion matrix

```
[13]: plot_heatmap(all_df)
```



3.1.2 Drop missing values

```
[14]: all_df = all_df.dropna(how='any')
test_df = test_df.dropna(how='any')
```

3.1.3 Drop variables with zero variation (unary)

```
[15]: print(all_df.var())

if 'does_muscle_bone_exercises' in all_df.columns:
    all_df = all_df.drop('does_muscle_bone_exercises', axis=1)

if 'does_muscle_bone_exercises' in test_df.columns:
    test_df = test_df.drop('does_muscle_bone_exercises', axis=1)

all_df.columns
```

```

mean_met          0.691199
sum_mag_acc       96081.026127
length_cm         103.643523
weight_kg         171.186772
mean_speed        7365.581017
bmi               16.249296
gender            0.252381
age_category      14.832345
is_sporter        0.200629
meets_activity_guidelines 0.086253
does_muscle_bone_exercises 0.000000
meets_balance_guidelines 0.176819
dtype: float64

```

```

[15]: Index(['mean_met', 'sum_mag_acc', 'length_cm', 'weight_kg', 'mean_speed',
            'bmi', 'gender', 'age_category', 'is_sporter',
            'meets_activity_guidelines', 'meets_balance_guidelines'],
          dtype='object')

```

3.1.4 Drop very low correlation variables

```

[16]: if 'gender' in all_df.columns:
        all_df = all_df.drop('gender', axis=1)

    if 'gender' in test_df.columns:
        test_df = test_df.drop('gender', axis=1)

    if 'bmi' in all_df.columns:
        all_df = all_df.drop('bmi', axis=1)

    if 'bmi' in test_df.columns:
        test_df = test_df.drop('bmi', axis=1)

    if 'is_sporter' in all_df.columns:
        all_df = all_df.drop('is_sporter', axis=1)

    if 'is_sporter' in test_df.columns:
        test_df = test_df.drop('is_sporter', axis=1)

    if 'age_category' in all_df.columns:
        all_df = all_df.drop('age_category', axis=1)

    if 'age_category' in test_df.columns:
        test_df = test_df.drop('age_category', axis=1)

    if 'meets_activity_guidelines' in all_df.columns:

```

```
all_df = all_df.drop('meets_activity_guidelines', axis=1)

if 'meets_activity_guidelines' in test_df.columns:
    test_df = test_df.drop('meets_activity_guidelines', axis=1)
```

3.1.5 Define remaining features

```
[17]: remaining_features = ['sum_mag_acc', 'mean_speed', 'length_cm', 'weight_kg',
    ↪ 'meets_balance_guidelines']
```

3.1.6 Recursive Feature Elimination (RFE)

```
[18]: best_features = get_best_features(all_df, remaining_features, 5)
print(best_features)
```

```
Index(['sum_mag_acc', 'mean_speed', 'length_cm', 'weight_kg',
      'meets_balance_guidelines'],
      dtype='object')
```

3.2 Multivariate regression best features

```
[19]: cleaned_df = all_df.dropna(how='any')

x = all_df[best_features]
y = all_df['mean_met']

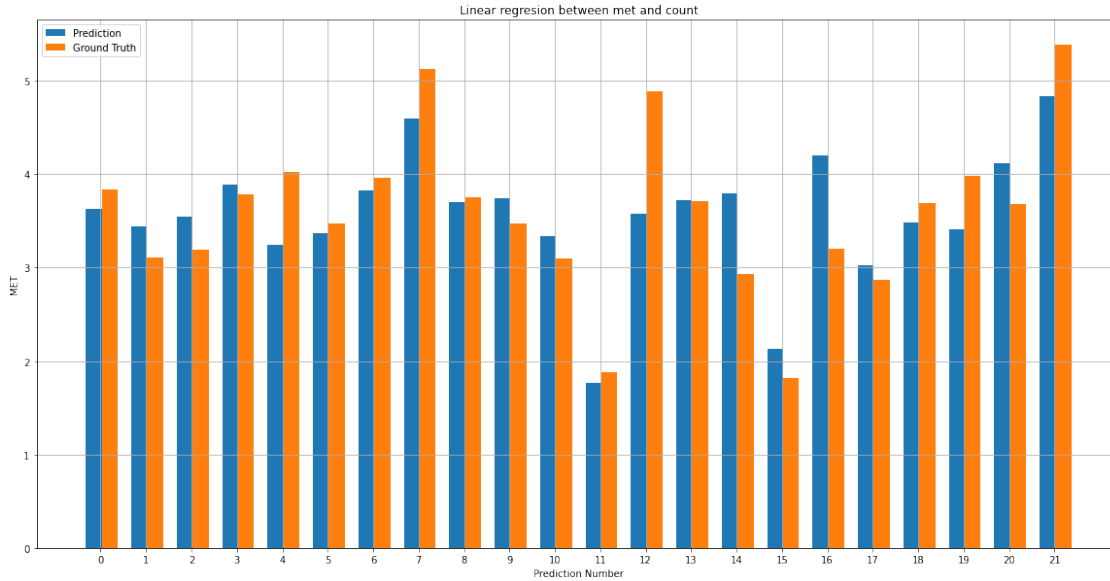
train_x, valid_x, train_y, valid_y = train_test_split(x,y, test_size=0.2,
    ↪ random_state=0)

mlr = LinearRegression()
mlr.fit(train_x,train_y)

pred_y = mlr.predict(valid_x)

plot_ground_truth_vs_prediction(pred_y, valid_y, 'Linear regression between met_
    ↪ and count')

print('mean squared error = ' + str(mean_squared_error(valid_y, pred_y)))
print('r squared = ' + str(mlr.score(train_x,train_y)))
```



mean squared error = 0.26400930717542187
r squared = 0.5886420285687723

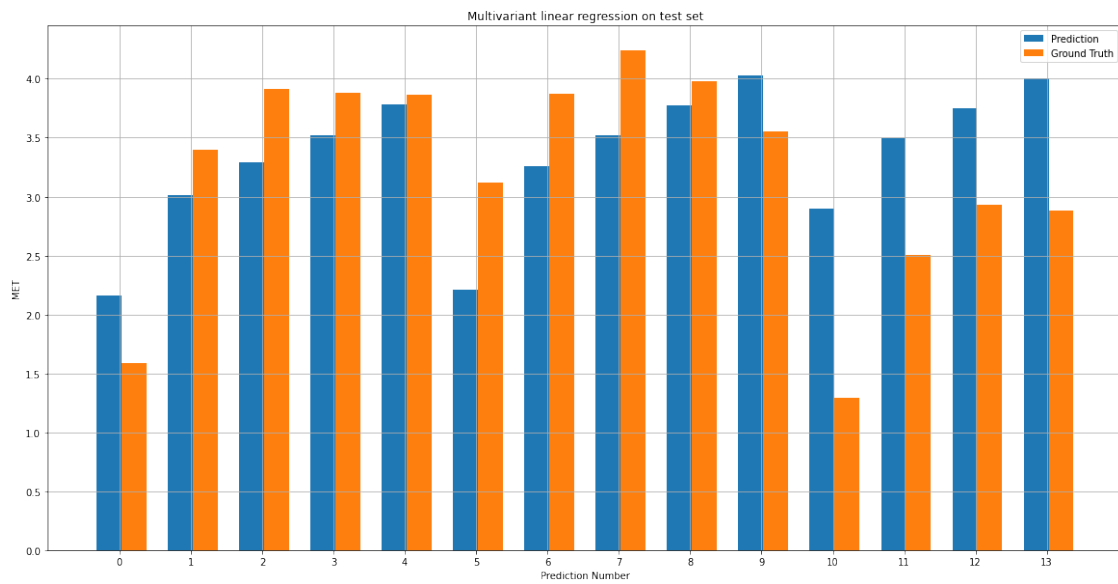
3.3 Hyperparameter tuning

[]:

3.4 Multivariate Linear Regression model on Test set

```
[20]: test_x = test_df[best_features]
test_y = test_df['mean_met']
pred_y = mlr.predict(test_x)

plot_ground_truth_vs_prediction(pred_y, test_y, 'Multivariant linear regression_
↳ on test set')
print('mean squared error = ' + str(mean_squared_error(test_y, pred_y)))
print('r squared = ' + str(mlr.score(test_x, test_y)))
```



```
mean squared error = 0.6065909444449543
r squared = 0.20094743223133726
```

```
[ ]:
```