# Digital Health Assignment 2

Maksimov, Dmitrii
dmitrii.maksimov@fau.de

January 28, 2022

# Exercise 7

1. Authorize your project, get your client ID and secret and configure the OAuth consentscreen. Explain in your own words how the authorization process works. What role does the OAuth token play?

   When application requests data the server on Google's side request credentials for access. These credentials are represented as a token, which is included in the http request address. After this token is decoded and credentials are checked, access to data is either granted or not.

2. Provide the request body to add a coffee at 10 AM on the 10.12.2015 using tht NutritionSource data type.

Listing 1: the request body to add a coffee at 10 AM on the 10.12.2015

```
{
    "minStartTimeNs": 1449738000000000000,
    "maxEndTimeNs": 1449738000000000000,
    "dataSourceId": "raw:com.google.nutrition:407408718192:NutritionSource",
    "point": [
        {
            "startTimeNanos": 1449738000000000000,
            "endTimeNanos": 1449738000000000000,
            "dataTypeName": "com.google.nutrition",
            "value": [
            {
                "mapVal": [
                    {
                        "key": "fat.total",
                        "value": {
                            "fpVal": 0.4
                        }
                    }
                ]
            },
            {
                "intVal": 1
            },
            {
                "strVal": "coffee"
            }
        ]
    }
]
}
```

3. Add points to the data resource.

   a) Explain your workflow in bullet points.

      1) create data resource via Post request
      2) swap "maxEndTimeNs" and "minStartTimeNs" in json file
      3) create url where datasetId is a "minStartTimeNs - maxEndTimeNs"
      4) create body request from file
      5) send PATCH request

   b) Provide a screenshot of the successful HTTP request statement, showing your added data.

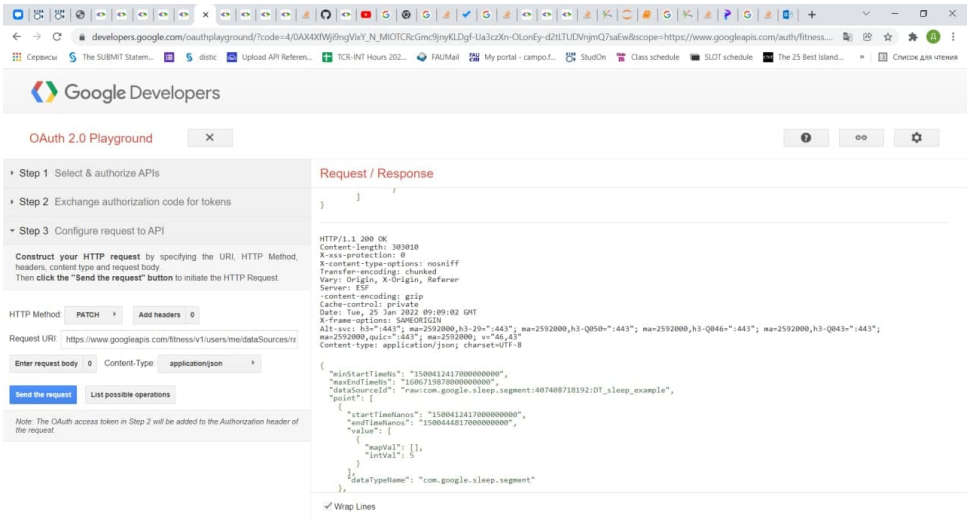


Figure 1: the successful HTTP request

4. Provide a screenshot of the successful HTTP request statement, showing your retrieved data.


(a) Request


(b) Response

Figure 2: Getting the previously posted data as aggregated data

5. Find and write down the language specific requirements that need to be implemented to run a web server from python.
    - Python 2.6 or greater
    - The pip package management tool
    - The Google APIs Client Library for Python
    - The google-auth, google-auth-oauthlib, and google-auth-httplib2 for user authorization
    - The Flask Python web application framework
    - The requests HTTP library

# Exercise 8

1. exchange the access SCOPE, API SERVICE NAME and API VERSION

Listing 2: SCOPE API SERVICE NAME and API VERSION

```
SCOPES = [
    f'https://www.googleapis.com/auth/fitness.{type}.{access}'
    for type in [
    'activity', 'blood_glucose', 'blood_pressure', 'body', 'body_temperature',
    'heart_rate', 'location', 'nutrition', 'oxygen_saturation', 'reproductive_health', 'sleep'
    ]
    for access in ['write', 'read']
]
API_SERVICE_NAME = 'fitness'
API_VERSION = 'v1'
```

2. Provide the lines of code for the GET method of requesting a specific data source dataset in Python programming.

Listing 3: GET method of requesting a specific data source dataset

```
fit.users().dataSources().datasets()\
    .get(userId='me', body=data_source_id, data_set_id).execute()
```

3. Download the coffee data example DT.json file from studon and add the data to your dataSources via the create method

   a) Provide the lines of code for the POST and PATCH method used in Python

Listing 4: lines of code for the POST and PATCH methodt

```
#POST
fit.users().dataSources().create(userId='me', body=body).execute()
#PATCH
fit.users().dataSources().datasets()\
      .patch(userId='me', dataSourceId=data_source_id,
             datasetId='1500407287000000000-1606687074000000000',
             body=body).execute()
```

   b) Provide the request body for the aggregated coffee data from the 01.04.2018-01.06.2018.

Listing 5: the request body for the aggregated coffee data from the 01.04.2018-01.06.2018

```
data_source_id = 'raw:com.google.nutrition:635848177559:CoffeeTestSource'
dt_begin = datetime(2018, 4, 1, 0, 0)
dt_end = datetime(2018, 6, 1, 23, 59)
body = {
    "aggregateBy": [
        {
            "dataSourceId": data_source_id,
        }],
    "startTimeMillis": int(dt_begin.timestamp() * 1e3),
    "endTimeMillis": int(dt_end.timestamp() * 1e3)
}
```

# Exercise 9

1. Load the sleep and coffee data into Python

   a) Provide the code snipped to load in multiple .json files.

Listing 6: load in multiple .json files

```
def get_dict_from_json(file):
    with open(file) as json_file:
        return json.load(json_file)

coffee_data = get_dict_from_json('coffee_DT_lin_reg.json')
sleep_data = get_dict_from_json('sleep_DT_lin_reg.json')
```

   b) Provide the modified code to convert the unix timestamp into a datetime object

Listing 7: convert the unix timestamp into a datetime object

```
datetime.fromtimestamp(time_nanos / 1e9)
```

   c) When you are using datetime.utcfromtimestamp, what does this imply for you data?

   Return the UTC datetime corresponding to the POSIX timestamp.

2. Once you have visualized the imported data. What relationship would you assume?

   Not quite a linear relationship. Starting at some point, the relationship becomes more linear.

3. Apply a linear regression model to it according to the steps described above.

   a) Provide a screenshot of the plot output with x- and y- axis being hours since midnight.

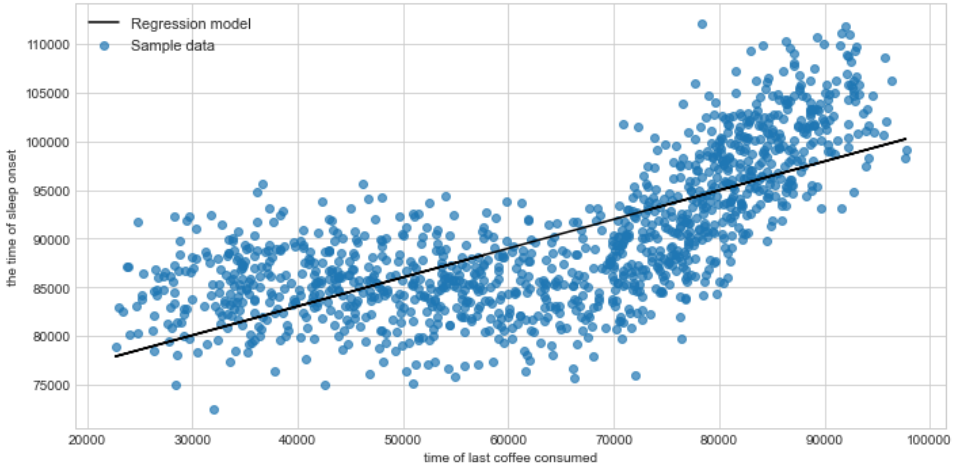


Figure 3: Result of regression

   b) What is the estimated sleep onset, if you have your last coffee at 20:29:06

   Next day at 01:51:34.

# Exercise 10

1. Apply a polynomial regression model to the sleep and coffee dataset.
   a) Provide the screenshot of your model output graph and the polynomial function.

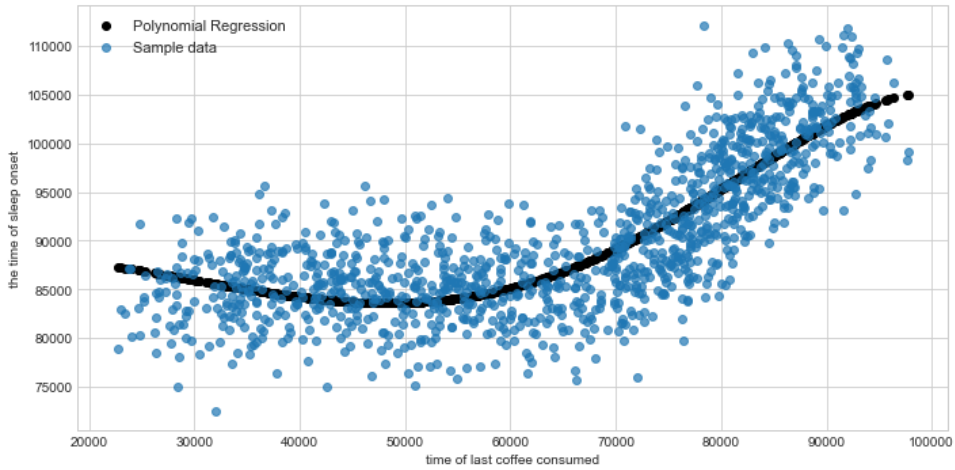


Figure 4: Result of polynomial regression

   b) Provide the predicted sleep onset time when having a coffee at 20:29:06 and 15:03:45.

      coffee: 20:29:06 - sleep onset: 01:21:12; coffee: 15:03:45 - sleep onset: 23:19:13

   c) Compare your results with the results of the linear regression model. What kind of error estimation can be performed and what are the limitations of the models used?

      It's obvious that results of the polynomial regression outperforms the linear regression. This is because the relationship between dependent and independent variables is not linear. MSE can be chosen as a metric. Despite the fact that polynomial regression is well suited for this task, it has some limitations:

      - very sensitive to the outliers
      - prone to overfitting
      - variables may not be highly correlated

2. Download the DT MC example 1.py file provided on StudOn and fill in the missing parts
   a) Use the MC method to approximate the actual data distribution. Provide the predicted parameter error after 10000 runs.

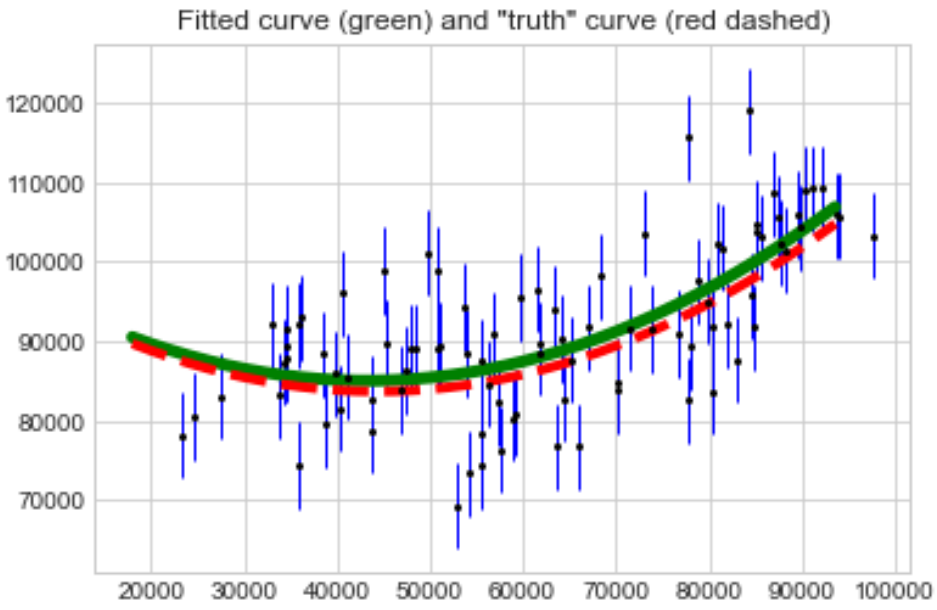


Figure 5: Result of MC algorithm

From the image above we can conclude what actual and predicted coefficients are similar, but in order to quantify the result, we calculate MAE of coefficients except for intercept: -0.06

b) Use a histogram to plot the result and explain it.

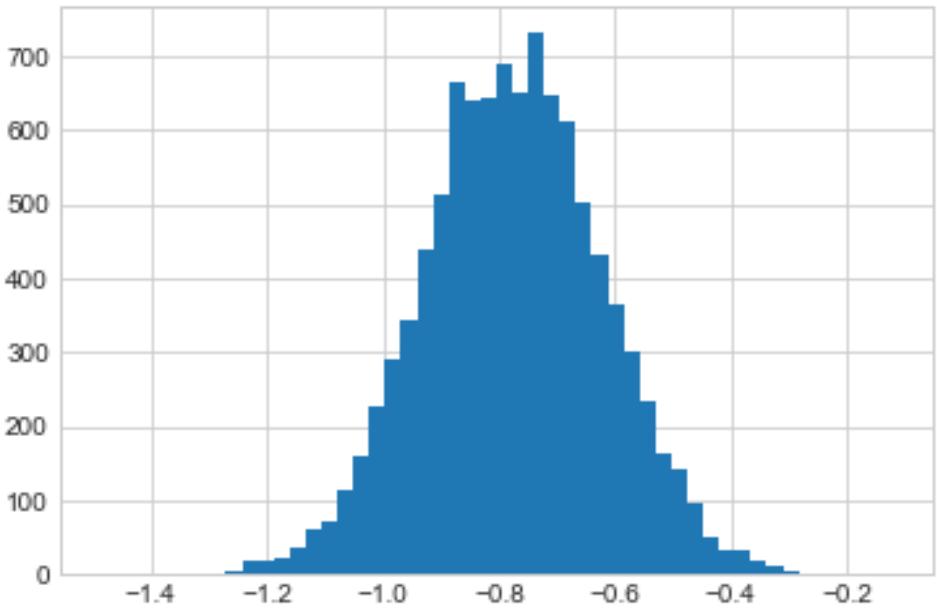


Figure 6: Histogram for "b" coefficient

To be honest I can't draw any conclusion from this histogram other than that distribution is normal. Hence, we can conclude that result of simulations is adequate.

3. Estimation of sleep onset probability

a) Provide the predicted probability of you going to sleep after 12pm or later

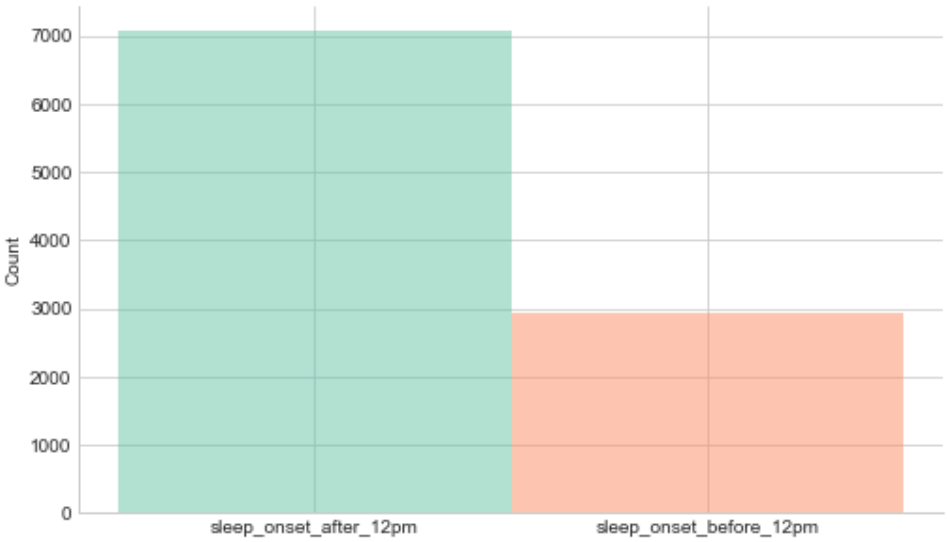


Figure 7: predicted probabilities

b) Provide the modified code snippet.

Listing 8: DT MC example 2

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

#count data entries according to time
#count_early coffee
x_early = len(sec_since_mid_coffee_modi[sec_since_mid_coffee_modi < 12 * 60 * 60])# before oder at 12am
#value of x

#count_early coffee
x_mid = len(sec_since_mid_coffee_modi[(sec_since_mid_coffee_modi >= 12 * 60 * 60) & (sec_since_mid_coffee_modi < 21 * 60 * 60)])#
#value of x

#count_late coffee
x_late = len(sec_since_mid_coffee_modi[sec_since_mid_coffee_modi >= 21 * 60 * 60])

#calculate probability
p_early_coffee=x_early/len(sec_since_mid_coffee_modi)
p_mid_coffee=x_mid/len(sec_since_mid_coffee_modi)
p_late_coffee=x_late/len(sec_since_mid_coffee_modi)



def DT_sleep_model(consuming_hours, coffee_threshold, p_early_coffee, p_mid_coffee, p_late_coffee):
    ''' The model, i.e., how the different inputs interact to generate the output
    Model constants (not picked at random)
    :param consuming_hours: number of hours in a day you are consuming coffee (e.g. 5-19 Uhr)
    :param coffee_threshold: amount of coffee you need that will for sure result in a sleep_onset_after 12pm
    Model inputs (picked at random)
    :param p_early_coffee: probability of a last early_coffee
    :param p_mid_coffee: probability of a last mid_coffee
    :param p_late_coffee: probability of a last late_coffee
    :return: The result of the model is either sleep_onset_before_12pm or sleep_onset_after_12pm.'''
    # taking 1 sample from the bernoulli distribution with a given probability of success.
    # a binomial distribution with just 1 trial is equivalent to the bernoulli distribution. that's done by setting size = 1
    if np.random.binomial(1, p_late_coffee, 1)[0]:
        # you're having a late coffee (after 9pm) it will result in a late sleep anyways
        return False
      # starting your day, and drink coffee
    day_start = True
    mid_coffee = np.random.binomial(1, p_mid_coffee, 1)[0]
    early_coffee = np.random.binomial(1, p_early_coffee, 1)[0]
    if mid_coffee:
        early_coffee=False
    while consuming_hours > 0 and coffee_threshold > 0:
        if early_coffee and day_start:
            coffee_threshold -= 2 #extra cup
            day_start=False
            consuming_hours-=4 #as 6 start --> until 12am when total is 10h consuming time

        if mid_coffee and consuming_hours % 2 == 0:
          # if you're having a mid_coffee, you get a cup of coffee every couple of hours
          coffee_threshold -= 2

        consuming_hours -= 1 # deducting hours
    return coffee_threshold == 0
```

```
def run_DT_simulations(runs=10000, consuming_hours=0, coffee_threshold=0, p_early_coffee=0.7, p_mid_coffee=0.35, p_late_coffee=0.0
    ''' Running 10,000 simulations (default) and plotting the results
    Only highlighting here the parameters not mentioned in the model function
    :param runs: number of times the model will run
    :return:
    '''
    results = []
    sim_count = 0
    while sim_count < runs:
        results.append(DT_sleep_model(consuming_hours, coffee_threshold, p_early_coffee, p_mid_coffee, p_late_coffee))
        sim_count += 1
  # making the output easier to interpret
    results = list(map(lambda x: 'sleep_onset_before_12pm' if x else 'sleep_onset_after_12pm', results))
    results_pd = pd.DataFrame(data=results)
    results_pd.columns = ['type']
    p_sleep_onset_before_12pm = round((results_pd[results_pd['type'] == 'sleep_onset_before_12pm'].count().values[0])/float(runs),
    p_sleep_onset_after_12pm = round((results_pd[results_pd['type'] == 'sleep_onset_after_12pm'].count().values[0]) / float(runs),
    print('Probability of having a sleep_onset_before_12pm = ' + str(p_sleep_onset_before_12pm) + ' || Probability of a sleep_onset
    # plotting the output of all model runs
    fig, ax = plt.subplots(figsize=(7, 4), constrained_layout=True)
    # removing to and right border
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    # setting the color palette
    sns.set_palette("Set2")
    sns.histplot(data=results_pd, x='type', hue='type', legend=False, linewidth=0)
    ax.xaxis.label.set_visible(False)
    plt.show()
    return(results)



results=run_DT_simulations(consuming_hours=10, coffee_threshold=6, p_early_coffee=p_early_coffee,
                           p_mid_coffee=p_mid_coffee, p_late_coffee=p_late_coffee)
```

# Exercise 11

1. Provide the modified code snippet.

Listing 9: DT Bayes Linear Regression

```
def linregFitBayes(X, y, maxIter):
    #This learn a Bayes regression model by optimizing log marginal likelihood.

    #we can start with some initial values for alpha and beta and then change them iteratively based on data
    alpha = 0.01
    beta = 1

    log_ml_old = -float('inf') #initialize log marginal likelihood
    Lhist = np.empty((maxIter, 1))

    #implement the log marginal likelihood logp(Y|X) and for a fixed number of iteration optimize it
    #at each loop iteration compare new (L) and old (L_old) log marginal likelihood
    #if abs(L - L_old) < 1e-2 stop the optimization

    #Initialize some variables --> they stay the same over all iterations, therfore out of the loop
    N, M = X.shape # matrix dimension
    XX = np.dot(np.transpose(X), X) # covariance of X np.dot --> see https://numpy.org/doc/stable/reference/generated/numpy.dot.html
    Xy = np.dot(np.transpose(X), y) # covariance of y

    for i in range(maxIter): #iterativly estimate gradient decent
        # needed formulars
        # identity matrix A needed for log maginal likelihood and prediction later p. 3
        A = alpha*np.identity(M) + beta * XX # formula learning parameters p. 3 formula np.identity --> see https://numpy.org/doc/stable/
        # you will need the inverse of matrix A, therefore one way of computing it is using the Cholesky decomposition
        cholA = np.transpose(np.linalg.cholesky(A)) # Cholesky decomposition formula to get the inverse of A see --> https://numpy.org/doc
        Ui = np.linalg.inv(cholA) # formula is needed to calculate inverse of A
        A_inv = np.dot(Ui, np.transpose(Ui)) # formula for A inverted p.3

        mn = beta * np.dot(A_inv, Xy) # formula p.3 fill in missing parameter

        #intermediate variable to use to compute the Expectation Em
        t1 = sum((y - np.dot(X, mn)) * (y - np.dot(X, mn)))# inner term of formular p.3
        t2 = np.dot(np.transpose(mn), mn) # second term of formular p. 3 --> all the variables are devided by 2 in the formular --> devide

        Emn = beta * t1 - alpha * t2 # merge the two intermediate variables to one formular see p.3
        # log marginal likelihood
        log_det = sum(np.log(np.diag(A)))# norm of the matrix p.3

        #maximizing initial formula p.3
        log_ml = float(M) * np.log(alpha) + float(N) * np.log(beta) - log_det \
                 - Emn - float(N) * np.log(2 * np.pi)
        log_ml = log_ml / 2 #--> as stated earlier here we divide everything by 2 insead of every term seperately

        # update paramerters p.4
        gamma = M - alpha * np.trace(A_inv)# formula p.4
        alpha = gamma/ t2
        beta = (N - gamma) / t1

        Lhist[i] = log_ml# keep track of log_ml to see convergence
        if abs(log_ml_old - log_ml) < 1e-2: # run until convergence
            break
        else:
            log_ml_old = log_ml

    # dict of our model
    model = {'wN': mn, 'VN': A_inv, 'beta': beta, 'alpha': alpha, 'gamma': gamma}

    return model, log_ml_old, Lhist
```

2. Provide the output graph and describe it.

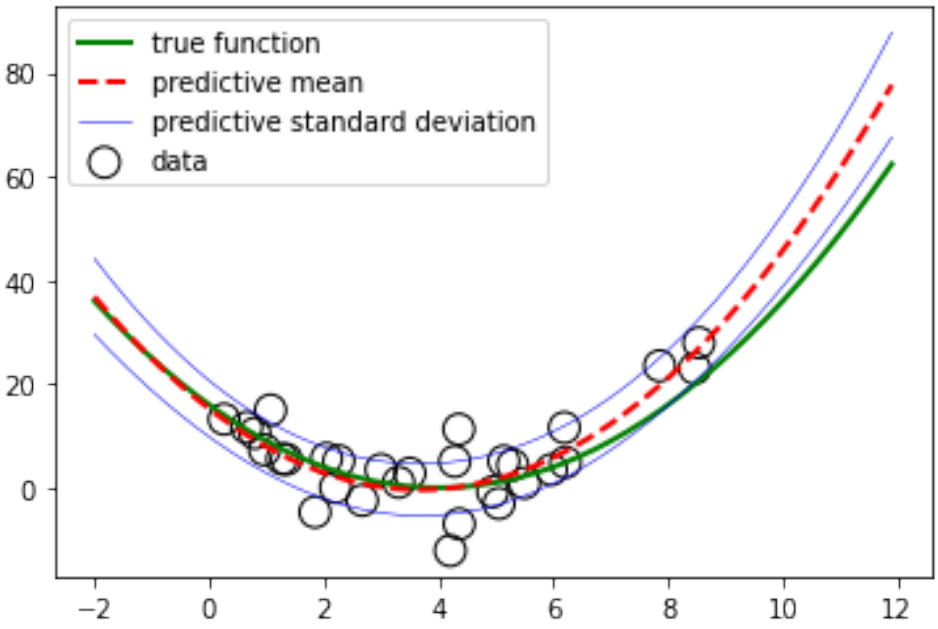


Figure 8: the output graph

From the illustration above we can infere that Bayesian linear regression aproximates our original function well despite the additional noise. The function being fitted went outside the standard deviation at the right side due to the greater effect of noise on the data.

3. How would you communicate the uncertainty of the model?

   The more predict uncertainty estimate value the more uncertainty by far. Hence, if this value quit a big we can communicate the uncertainty of the model.