Mój kod

In [1]:
```
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model,preprocessing
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import time
```

In [2]:
```
data = pd.read_csv("outfile.csv")
data.head(10)
```

	X_Val	Y_Val
0	180	99.074074
1	184	99.074074
2	188	99.074074
3	192	99.074074
4	196	99.074074
5	200	99.074074
6	204	99.074074
7	208	99.074074
8	212	99.074074
9	216	99.074074

In [3]:
```
X_Val = data.iloc [:, 0]
Y_Val = data.iloc[:,1]
X_Val =X_Val.as_matrix()
Y_Val =Y_Val.as_matrix()
```

```
E:\Users\sticz\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: FutureWarning: Method .as_matrix will be
version. Use .values instead.
  This is separate from the ipykernel package so we can avoid doing imports until
E:\Users\sticz\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: FutureWarning: Method .as_matrix will be
version. Use .values instead.
  after removing the cwd from sys.path.
```

In [4]:
```
start_time = time.time()

X_Val = data.iloc [:, 0]
X_Val =X_Val.as_matrix()
model = Pipeline([('poly', PolynomialFeatures(degree=18)),
                  ('linear', LinearRegression(fit_intercept=True,normalize= Tru
model = model.fit(X_Val.reshape(-1, 1), Y_Val.reshape(-1, 1))

elapsed_time_sk = time.time() - start_time

elapsed_time_sk
```

```
E:\Users\sticz\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: FutureWarning: Method .as_matrix will be
version. Use .values instead.
  after removing the cwd from sys.path.


0.013889551162719727
```

In [5]:
```
model.named_steps['linear'].coef_   #Współczynniki wielomianu
```

```
array([[ 0.00000000e+00, -2.07438708e+01,  1.41300383e-01,
        -5.39264394e-04,  1.31448642e-06, -2.20692276e-09,
         2.67830261e-12, -2.42283563e-15,  1.66529922e-18,
        -8.79638579e-22,  3.59110429e-25, -1.13392794e-28,
         2.75710114e-32, -5.10696434e-36,  7.06724590e-40,
        -7.06828626e-44,  4.82254212e-48, -2.00728286e-52,
         3.84342654e-57]])
```

In [6]:
```
Xz = np.arange(1,5000)
Predykcje = model.predict(Xz.reshape(-1, 1))
Predykcje
```

```
array([[1348.24411015],
       [1327.92038527],
       [1307.87285531],
       ...,
       [ 126.5464373 ],
       [ 126.70464042],
       [ 126.86382011]])
```

In [7]:

```
1 Predykcje_df = pd.DataFrame(Predykcje)
2 Predykcje_df.head(50)
```

	0
0	1348.244110
1	1327.920385
2	1307.872855
3	1288.098363
4	1268.593781
5	1249.356013
6	1230.381993
7	1211.668686
8	1193.213084
9	1175.012211
10	1157.063120
11	1139.362891
12	1121.908635
13	1104.697491
14	1087.726625
15	1070.993233
16	1054.494537
17	1038.227787
18	1022.190261
19	1006.379264
20	990.792127
21	975.426209
22	960.278893
23	945.347591
24	930.629740
25	916.122801
26	901.824262
27	887.731637
28	873.842464
29	860.154306
30	846.664751
31	833.371410
32	820.271920
33	807.363942
34	794.645158
35	782.113279
36	769.766033
37	757.601175
38	745.616484
39	733.809757

In [8]:
```
Xz = pd.DataFrame(Xz)
Xz
```

	0
0	1
1	2
2	3
3	4
4	5
...	...
4994	4995
4995	4996
4996	4997
4997	4998
4998	4999

4999 rows × 1 columns

In [9]:
```
#model.score(Predykcje, Y_Val, sample_weight=None)
```

In [10]:
```
result = pd.concat([Xz, Predykcje_df], axis=1, join='inner')

result.to_csv('porazka.csv',index=False,header=False)
```

In [11]:
```
result
```

	0	0
0	1	1348.244110
1	2	1327.920385
2	3	1307.872855
3	4	1288.098363
4	5	1268.593781
...	...	...
4994	4995	126.230031
4995	4996	126.386281
4996	4997	126.546437
4997	4998	126.704640
4998	4999	126.863820

4999 rows × 2 columns

Własna implementacja

In [12]:
```
X = np.arange(1,5000)
```

In [13]:
```
Wielomianki = np.polyfit(X_Val,Y_Val,18)
Wielomianki  #wspolczynniki wielomianow
```

```
E:\Users\sticz\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RankWarning: Polyfit may be poorly condi
  """Entry point for launching an IPython kernel.


array([ 4.40560841e-58, -1.93714767e-53,  3.72364500e-49, -4.01317313e-45,
        2.49337414e-41, -6.58703646e-38, -2.76996220e-34,  3.74065277e-30,
       -1.97467120e-26,  6.55903341e-23, -1.49134491e-19,  2.35315966e-16,
       -2.52395554e-13,  1.73937158e-10, -6.76156722e-08,  8.81668753e-06,
        2.83949183e-03, -1.04141348e+00,  1.88806722e+02])
```

In [14]:
```
def Mein(x):
    power = 0 #potegi
    suma = 0
    for w in reversed(Wielomianki):
        if(power ==0):
            suma += w
        else:
            res = w*x**power
            suma +=res
        power +=1
    return suma
```

In [15]:
```
Mein(4000)
```

```
168.22003739327192
```

In [16]:
```
def do_listy(X):
    lista = []
    for a in range(X.size):
        wyniczek = Mein(a)
        lista.append(wyniczek)
    return lista
```

In [17]:
```
start_time = time.time()
lista = do_listy(X)
elapsed_time_moje = time.time() - start_time

elapsed_time_moje
```

```
0.4627692699432373
```

In [18]:
```
moje_wyniki_df = pd.DataFrame(lista)
```

In [19]:
```
1 moje_wyniki_df
```

	0
0	188.806722
1	187.768157
2	186.735323
3	185.708270
4	184.687047
...	...
4994	124.418402
4995	124.570158
4996	124.721839
4997	124.873432
4998	125.024772

4999 rows × 1 columns

In [20]:
```
1 moj_result = pd.concat([Xz, moje_wyniki_df], axis=1, join='inner')
2 moj_result
3 moj_result.to_csv('mojaporazka.csv',index=False,header=False)
```

In [21]:
```
1 moj_result
```

	0	0
0	1	188.806722
1	2	187.768157
2	3	186.735323
3	4	185.708270
4	5	184.687047
...	...	...
4994	4995	124.418402
4995	4996	124.570158
4996	4997	124.721839
4997	4998	124.873432
4998	4999	125.024772

4999 rows × 2 columns

Viva la fiesta i espania

In [22]:

```
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model,preprocessing
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
```

In [23]:

```
data = pd.read_csv("ElectionData.csv")
data.head(10)
```

	TimeElapsed	time	territoryName	totalMandates	availableMandates	numParishes	numParishesAp
0	0	2019-10-06 20:10:02	Território Nacional	0	226	3092	1081
1	0	2019-10-06 20:10:02	Território Nacional	0	226	3092	1081
2	0	2019-10-06 20:10:02	Território Nacional	0	226	3092	1081
3	0	2019-10-06 20:10:02	Território Nacional	0	226	3092	1081
4	0	2019-10-06 20:10:02	Território Nacional	0	226	3092	1081
5	0	2019-10-06 20:10:02	Território Nacional	0	226	3092	1081
6	0	2019-10-06 20:10:02	Território Nacional	0	226	3092	1081
7	0	2019-10-06 20:10:02	Território Nacional	0	226	3092	1081
8	0	2019-10-06 20:10:02	Território Nacional	0	226	3092	1081
9	0	2019-10-06 20:10:02	Território Nacional	0	226	3092	1081

10 rows × 28 columns

In [24]:
```
data_teritorio = data.query('territoryName == "Território Nacional"')
data_teritorio = data_teritorio.loc[:, ["TimeElapsed","Party","Mandates","FinalM
data_teritorio
```

	TimeElapsed	Party	Mandates	FinalMandates
0	0	PS	0	106
1	0	PPD/PSD	0	77
2	0	B.E.	0	19
3	0	CDS-PP	0	5
4	0	PCP-PEV	0	12
...	...	...	...	...
21256	265	PURP	0	0
21257	265	PDR	0	0
21258	265	PPM	0	0
21259	265	PTP	0	0
21260	265	MAS	0	0

1134 rows × 4 columns

In [25]:
```
data_teritorio.Party.unique()
```

```
array(['PS', 'PPD/PSD', 'B.E.', 'CDS-PP', 'PCP-PEV', 'PAN', 'CH',
       'R.I.R.', 'PCTP/MRPP', 'A', 'L', 'IL', 'JPP', 'NC', 'PDR', 'PNR',
       'PURP', 'PPM', 'MPT', 'PTP', 'MAS'], dtype=object)
```

In [26]:
```
PS = data_teritorio.query('Party == "PS"')
PPD_PSD = data_teritorio.query('Party == "PPD/PSD"')
BE = data_teritorio.query('Party == "B.E."')
CDS_PP = data_teritorio.query('Party == "CDS-PP"')
PCP_PEV = data_teritorio.query('Party == "PCP-PEV"')
PAN = data_teritorio.query('Party == "PAN"')
CH = data_teritorio.query('Party == "CH"')
RIR = data_teritorio.query('Party == "R.I.R."')
PCTP_MRPP = data_teritorio.query('Party == "PCTP/MRPP"')
A = data_teritorio.query('Party == "A"')
L = data_teritorio.query('Party == "L"')
IL= data_teritorio.query('Party == "IL"')
JPP = data_teritorio.query('Party == "JPP"')
NC = data_teritorio.query('Party == "NC"')
PDR = data_teritorio.query('Party == "PDR"')
PNR = data_teritorio.query('Party == "PNR"')
PURP = data_teritorio.query('Party == "PURP"')
PPM = data_teritorio.query('Party == "PPM"')
MPT = data_teritorio.query('Party == "MPT"')
PTP = data_teritorio.query('Party == "PTP"')
MAS = data_teritorio.query('Party == "MAS"')
```

In [27]:
```
lista_partii = [PS,PPD_PSD,BE,CDS_PP,PCP_PEV,PAN,CH,RIR,
                    PCTP_MRPP,A,L,IL,JPP,NC,PDR,PNR,PURP,PPM,MPT,PTP,MAS]
```

In [28]:
```
1 #lista_partii[19]
2 PS
```

	TimeElapsed	Party	Mandates	FinalMandates
0	0	PS	0	106
386	5	PS	0	106
772	10	PS	0	106
1158	15	PS	0	106
1544	20	PS	0	106
1930	25	PS	0	106
2316	30	PS	0	106
2702	35	PS	0	106
3105	40	PS	6	106
3508	45	PS	8	106
3911	50	PS	9	106
4314	55	PS	14	106
4717	60	PS	16	106
5120	65	PS	16	106
5523	70	PS	21	106
5926	75	PS	21	106
6329	80	PS	22	106
6732	85	PS	29	106
7135	90	PS	34	106
7538	95	PS	36	106
7941	100	PS	39	106
8344	105	PS	39	106
8747	110	PS	47	106
9150	115	PS	49	106
9553	120	PS	53	106
9956	125	PS	58	106
10359	130	PS	65	106
10762	135	PS	67	106
11165	140	PS	68	106
11568	145	PS	73	106
11971	150	PS	75	106
12374	155	PS	80	106
12777	160	PS	84	106
13180	165	PS	86	106
13583	170	PS	87	106
13986	175	PS	87	106
14389	180	PS	90	106
14792	185	PS	91	106
15195	190	PS	92	106
15598	195	PS	92	106

In [ ]:
```
1
```

In [29]:

```
def partie_funkcja():
    lista_partii = [PS,PPD_PSD,BE,CDS_PP,PCP_PEV,PAN,CH,RIR,
                    PCTP_MRPP,A,L,IL,JPP,NC,PDR,PNR,PURP,PPM,MPT,PTP,MAS]
    lista_wspolczynnikow = []
    lista_predykcji = []
    lista_X_Val = []
    lista_Y_Val = []
    lista_results = []

    for a in range (0,21):
        print(a)
        time = lista_partii[a].iloc[:,0]
        X_Val = time.as_matrix()

        mandates = lista_partii[a].iloc[:, 2]
        Y_Val = mandates.as_matrix()

        model = Pipeline([('poly', PolynomialFeatures(degree=18)),
                          ('linear', LinearRegression(fit_intercept=True,normalize=
        model = model.fit(X_Val.reshape(-1, 1), Y_Val.reshape(-1, 1))


        wspolczynniki = model.named_steps['linear'].coef_
        predykcje = model.predict(X_Val.reshape(-1, 1))

        lista_wspolczynnikow.append(wspolczynniki)
        lista_Y_Val.append(Y_Val)


        predykcje_df = pd.DataFrame(predykcje).astype(int)
        lista_predykcji.append(predykcje)

        X_Val_pd = pd.DataFrame(X_Val)
        lista_X_Val.append(X_Val_pd)


        lista_Y_Val.append(Y_Val)
        Y_Val_df = pd.DataFrame(Y_Val)

        result = pd.concat([X_Val_pd, predykcje_df,Y_Val_df], axis=1, join='inne
        lista_results.append(result)

        a+=1
```

In [30]:
```
lista_wspolczynnikow,lista_predykcji,lista_X_Val,lista_Y_Val,lista_results = par
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

```
E:\Users\sticz\Anaconda3\lib\site-packages\ipykernel_launcher.py:13: FutureWarning: Method .as_matrix will b
```

In [31]:
```
lista_results[2]  #time, mandaty predykcja, mandaty rzeczywistosc
```

	0	0	0
0	0	0	0
1	5	0	0
2	10	0	0
3	15	0	0
4	20	0	0
5	25	0	0
6	30	0	0
7	35	0	0
8	40	0	0
9	45	0	0
10	50	0	0
11	55	0	0
12	60	0	0
13	65	0	0
14	70	0	0
15	75	0	0
16	80	0	0
17	85	1	1
18	90	1	2
19	95	1	2
20	100	1	2
21	105	2	2
22	110	2	2
23	115	2	2
24	120	3	3
25	125	4	6
26	130	5	6
27	135	6	7
28	140	7	7
29	145	9	8
30	150	10	10
31	155	11	12
32	160	12	14
33	165	13	14
34	170	14	14
35	175	14	14
36	180	14	15
37	185	14	15
38	190	14	15
39	195	14	15
40	200	15	15

In [32]:

```
#najwieksze partie - predykcje mandatow
predykcja_partia_PS = lista_results[0].iloc[:, 1].values.tolist()
predykcja_partia_PPD = lista_results[1].iloc[:, 1].values.tolist()
predykcja_partia_BE = lista_results[2].iloc[:, 1].values.tolist()
predykcja_partia_PS
```

```
[0,
 0,
 0,
 0,
 0,
 0,
 0,
 1,
 4,
 7,
 10,
 13,
 15,
 17,
 19,
 21,
 24,
 28,
 31,
 35,
 38,
 42,
 45,
 49,
 53,
 57,
 62,
 66,
 70,
 73,
 76,
 79,
 82,
 84,
 86,
 88,
 90,
 90,
 91,
 92,
 92,
 93,
 95,
 97,
 98,
 99,
 99,
 99,
 99,
 100,
 100,
 101,
 103,
 105]
```

In [33]:
```
def binary_search(item_list,mid):
    first = item_list[0]
    last  = item_list[-1]
    szukany = last
    print(item_list[mid])
    if szukany - item_list[mid] <= 30 :
        print('got it')
        return item_list[mid],mid
    else:
        if szukany < item_list[mid]:
                mid = mid - 1
                return binary_search(item_list,mid)

        else:
                mid = mid + 1
                return binary_search(item_list,mid)

```

In [34]:
```
mid = int(len(predykcja_partia_PS)/2)
a, b = binary_search(predykcja_partia_PS,mid)
wynik_PS = lista_results[0].iloc[b, :]

mid = int(len(predykcja_partia_PPD)/2)
c,d  = binary_search(predykcja_partia_PPD,mid)
wynik_PPD = lista_results[1].iloc[d, :]

mid = int(len(predykcja_partia_BE)/2)
d,e  = binary_search(predykcja_partia_BE,mid)
wynik_BE = lista_results[2].iloc[e, :]
```

```
66
70
73
76
got it
49
got it
6
got it
```

In [35]:
```
wynik_PS #timestamp - 150
```

```
0    150
0     76
0     75
Name: 30, dtype: int64
```

In [36]:
```
wynik_PPD  #timestamp - 135
```

```
0    135
0     49
0     49
Name: 27, dtype: int64
```

In [37]:
```
wynik_BE  #timestamp - 135, bo binary search szuka od srodka,  a mandatow jest <
```

```
0    135
0      6
0      7
Name: 27, dtype: int64
```

Implementacja Sk-learn jest wielokrotnie szybsza, oraz wykazuje większą dokładność na tym zb

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: