

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la

def plot_points(x,y,xlabel,ylabel):
    plt.figure(figsize=(15, 10))
    plt.plot(x,y,'ob', markersize = 5, markerfacecolor = 'b')
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)

In [2]: def lsquares(x,y,stepen):
    n=len(x)
    A=np.zeros([n,stepen+1])

    for i in range(n):
        for j in range(stepen+1):
            A[i,j]=x[i]**j

    p=la.solve(np.matmul(A.T, A), np.matmul(A.T, y.T))
    p=np.flip(p)

    return p
```

## Linearna regresija

Linearna regresija predstavlja alternativan način za pronalaženje trenda u podacima u odnosu na interpolaciju.

Videli smo da jedan interpolacioni polinom nije dobro rešenje kada imamo veliku količinu podataka. Na današnjem predavanju pokazaćemo da ni splajnovi nisu dobro rešenje u tom slučaju.

Uvodna napomena oko terminologije:

Termin "linearna" ne znači da je rezultat regresije prava odnosno linearna funkcija, već da je rezultat regresije linearna kombinacija koeficijenata i nekih funkcija. Funkcije se zadaju, a koeficijenti određuju.

Na primer, prava je linearna kombinacija koeficijenata  $k$  i  $n$  i funkcija  $f(x) = x$  i  $f(x) = 1$ , dok je polinom drugog stepena linearna kombinacija koeficijenata  $a, b$  i  $c$  i funkcija  $f(x) = x^2, f(x) = x$  i  $f(x) = 1$

## Motivacioni primer

Kao jedan od primera upotrebe regresije uzećemo predikciju ishoda utakmica u Premier ligi.

Tim koji stoji iza sajta <http://www.football-data.co.uk/>, želio je da napravi model za "fer" kvote za utakmice. "Fer" znači da nemaju cilj da zarde od tudeg kladjenja (kao kladionice).

Krenuli su sa jednostavnim pristupom, da je dobar indikator kvaliteta tima razlika u golovima (RG)= broj datih golova - broj primljenih golova na prethodno odigranim utakmicama.

Odabrali su da posmatraju 6 prethodno odigranih utakmica.

Na taj način kreirali su indikator koji su nazvali rejting utakmice (match rating, MR).

$MR = RG_{domaći\_tim} - RG_{gostujući\_tim}$

Npr. Ako igraju Manchester-Liverpool; Man. ima  $RG=+5$ , a Liv.  $RG=+2$ ,  $MR=5-2=3$ .

Da bi kreirali svoj model posmatrali su mečeve odigrane u periodu 1993-2001. Za svaki meč zabeležili su MR i ishod meča.

Nakon toga su podatke organizovali po rejtingu i za svaki rejting izračunali su %pobeda\_domaćina (%PD), %pobeda\_gosta (%PG), %izjednačeno (%I).

U slećem redu učitavamo njihove podatke iz csv (comma separated values) fajla.

Kolone su redom: MR, Broj pobeda domaćeg, Broj pobeda gostujućeg, Broj izjednačenih, %pobeda\_domaćina (%PD), %pobeda\_gosta (%PG), %izjednačeno (%I)

```
In [3]: import pandas as pd

data=pd.read_csv('fudbal.csv', sep=',')
data['%_of_home_wins']=data['%_of_home_wins'].apply(lambda x: float(x.split('%')[0]))
data['%_of_home_draws']=data['%_of_home_draws'].apply(lambda x: float(x.split('%')[0]))
data['%_of_away_wins']=data['%_of_away_wins'].apply(lambda x: float(x.split('%')[0]))

data['Match_rating']=data['Match_rating'].astype(float)

data.head()
```

	Match_rating	Number_of_home_wins	Number_of_draws	Number_of_away_wins	%_of_home_wins	%_of_home_draws	%_of_away_wins
0	-26.0	0	1	1	0.0	50.0	50.0
1	-23.0	0	0	2	0.0	0.0	100.0
2	-22.0	0	0	3	0.0	0.0	100.0
3	-21.0	0	2	4	0.0	33.3	66.7
4	-20.0	2	2	7	18.2	18.2	63.6

Nakon toga kreirali su regresione funkcije (modele) za svaki od %, koristeći rejting meča kao x.

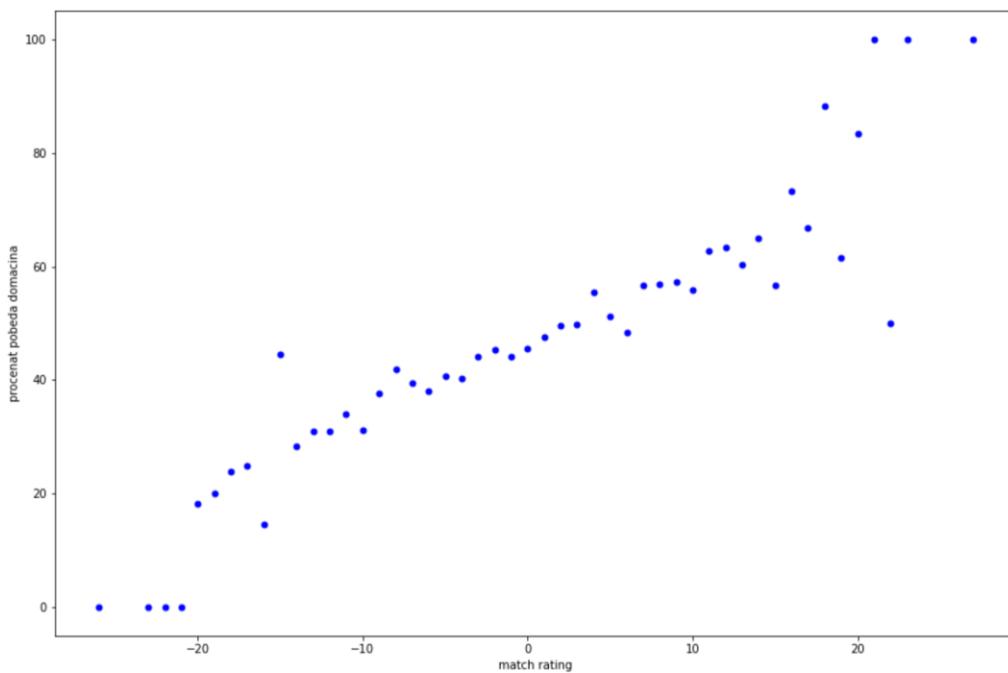
Sa ciljem da kad je poznat MR imaju funkciju na osnovu koje će da predvide %PD, %PG i %I.

Kada su imali procente jednostavno su 100 podelili sa svakim od njih i tako dobili kvote za PD, PG i I.

Npr. ako bi procenat pobede domaćina bio 46.7%, kvota bi bila  $100/46.7=2.15$

U nastavku ponavljamo postupak kreiranja regresionih funkcija, a nakon toga objašnjavamo se potrebne teorijske koncepte.

```
In [4]: match_rating = data['Match_rating']
procenat_pobeda_domacina = data['%_of_home_wins']
plot_points(match_rating,procenat_pobeda_domacina,'match rating','procenat pobeda domacina')
```

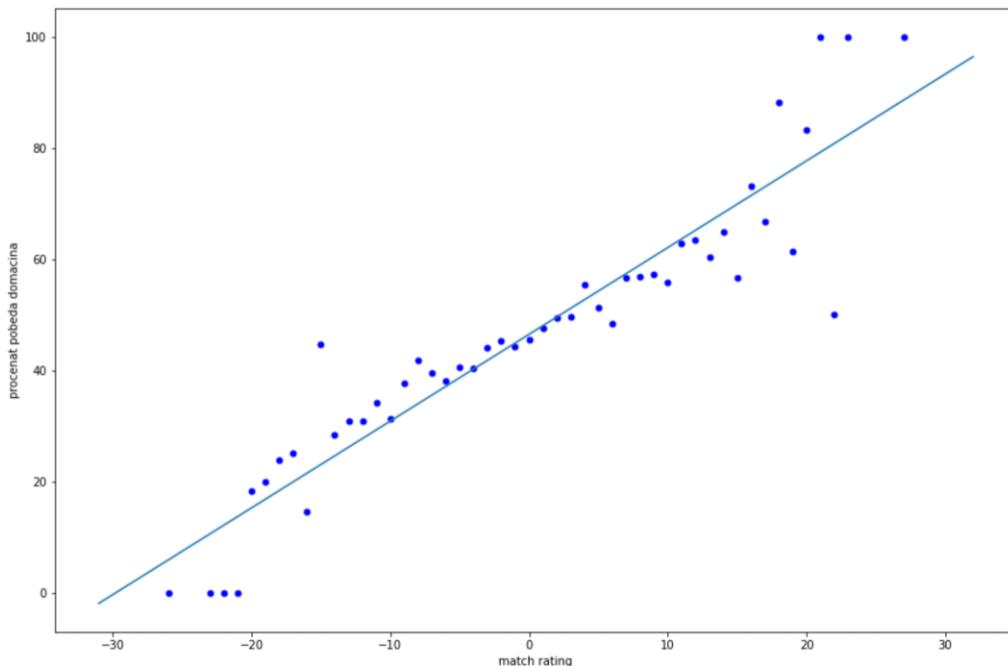


```
In [5]: x=match_rating
y=procenat_pobeda_domacina
p=lsquares(x, y, 1)
print(p)
```

```
[ 1.56148751 46.46609209]
```

```
In [6]: plot_points(x,y,'match rating','procenat pobeda domacina')
xp=np.linspace(np.min(x)-5,np.max(x)+5,100);
plt.plot(xp,np.polyval(p,xp))
```

```
Out[6]: [
```



```
In [7]: #ako je mr 15 kolike su sanse da pobedi domaci tim.
```

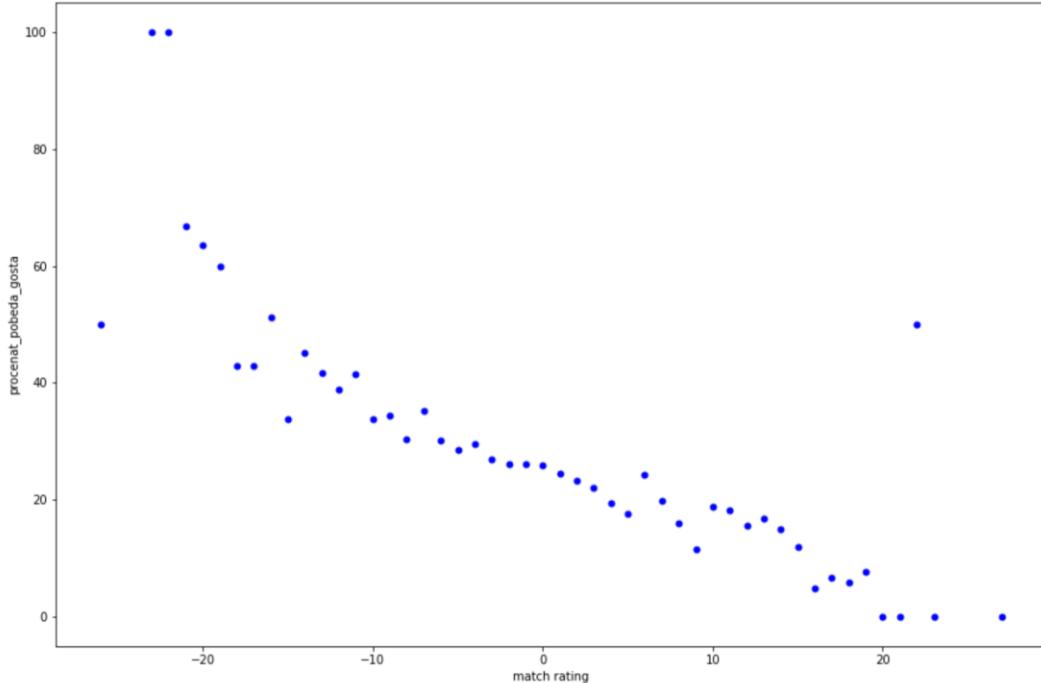
```
print(np.polyval(p,15))
```

```
69.88840469674946
```

```
In [8]: match_rating = data['Match_rating']
```

```
procenat_pobeda_gosta = data['%_of_away_wins']
```

```
plot_points(match_rating,procenat_pobeda_gosta,'match rating','procenat_pobeda_gosta')
```



```
In [9]: x=match_rating  
y=procenat_pobeda_gosta
```

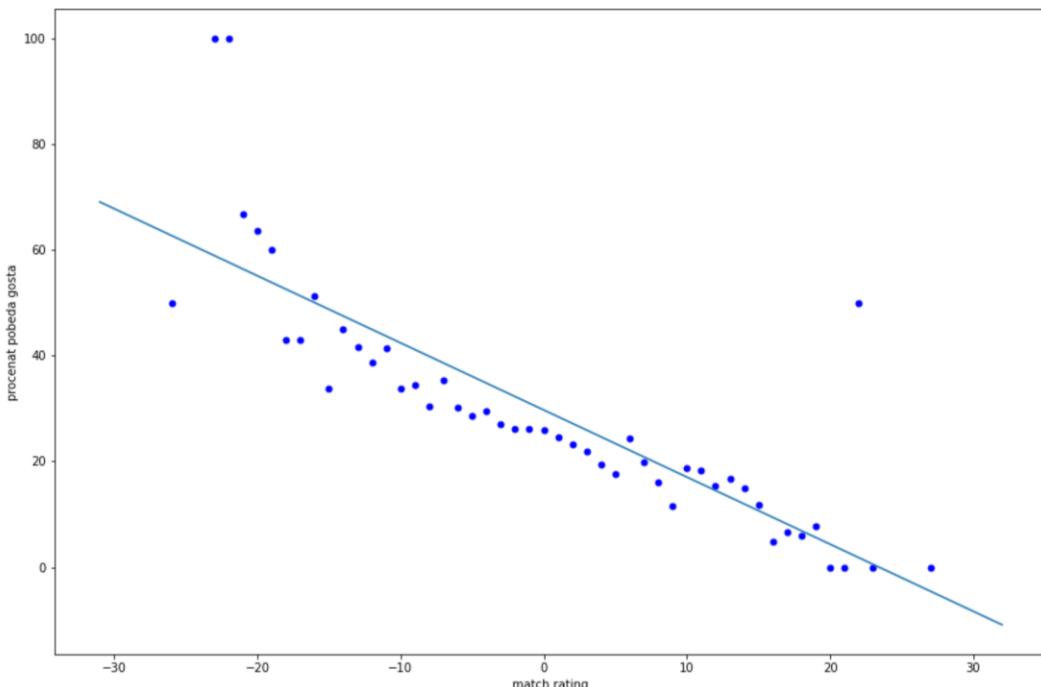
```
p=lsquares(x, y, 1)  
print(p)
```

```
[-1.26909739 29.73610403]
```

```
In [10]: plot_points(x,y,'match rating','procenat pobeda gosta')
```

```
xp=np.linspace(np.min(x)-5,np.max(x)+5,100);  
plt.plot(xp,np.polyval(p,xp))
```

```
Out[10]: [
```



```
In [11]: #ako je mr 15 kolike su sanse da pobedi gostujuci tim.  
print(np.polyval(p,15))
```

```
10.699643115250627
```

```
In [12]: #koristimo kvadratni polinom
```

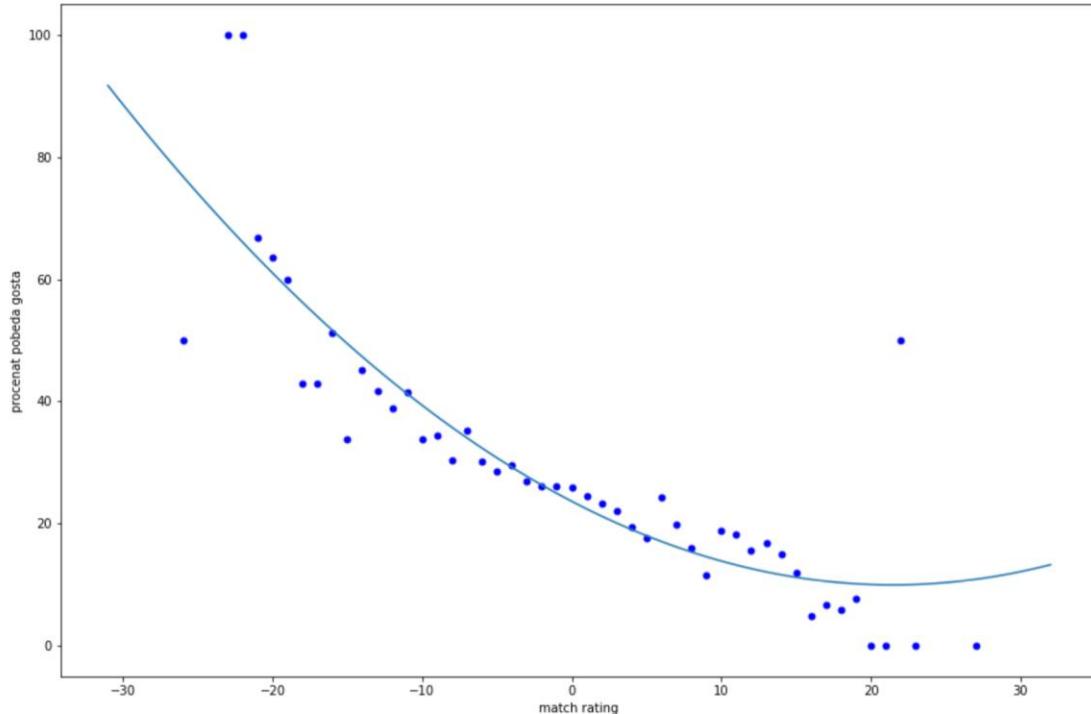
```
x=match_rating
y=procenat_pobeda_gosta;

p=lsquares(x, y, 2)
print(p)

plot_points(x,y,'match rating','procenat pobeda gosta');
xp=np.linspace(np.min(x)-5,np.max(x)+5,100);
plt.plot(xp,np.polyval(p,xp))

[ 0.02969177 -1.27471452 23.6445588 ]
```

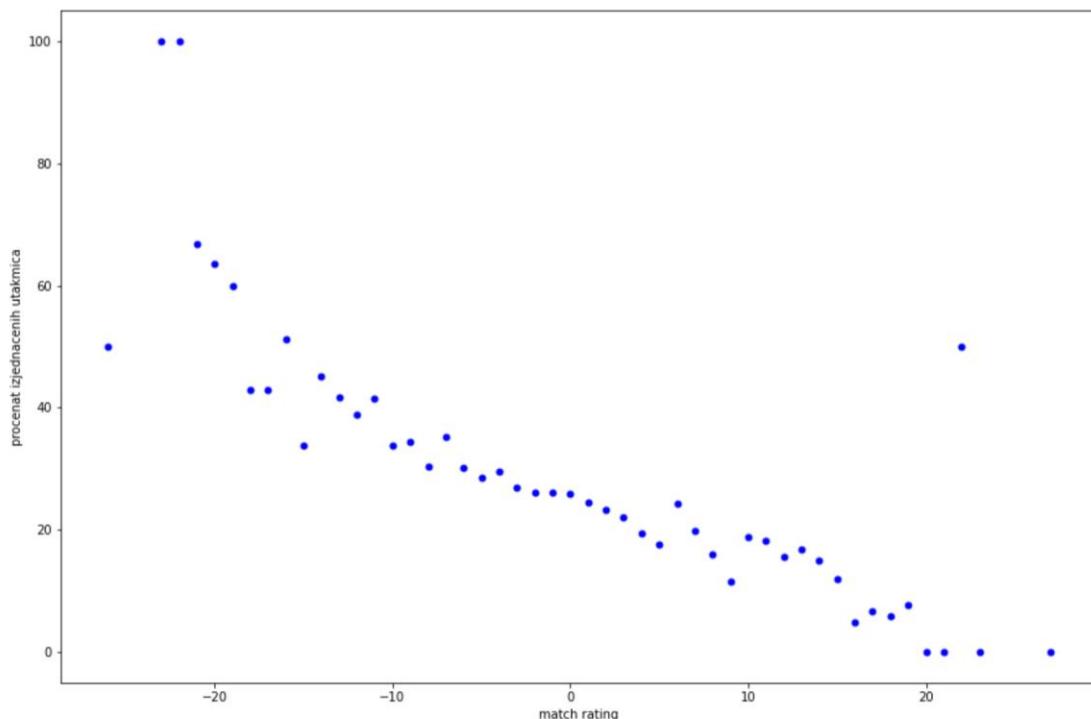
```
Out[12]: [<matplotlib.lines.Line2D at 0x1b912f518d0>]
```



```
In [13]: #ako je mr 15 kolike su sanse da pobedi gostujuci tim.
print(np.polyval(p,15))
```

```
11.204488504774536
```

```
In [14]: match_rating = data['Match_rating']
procenat_izjednacenih = data['%_of_home_draws']
plot_points(match_rating,procenat_pobeda_gosta,'match rating','procenat izjednacenih utakmica')
```

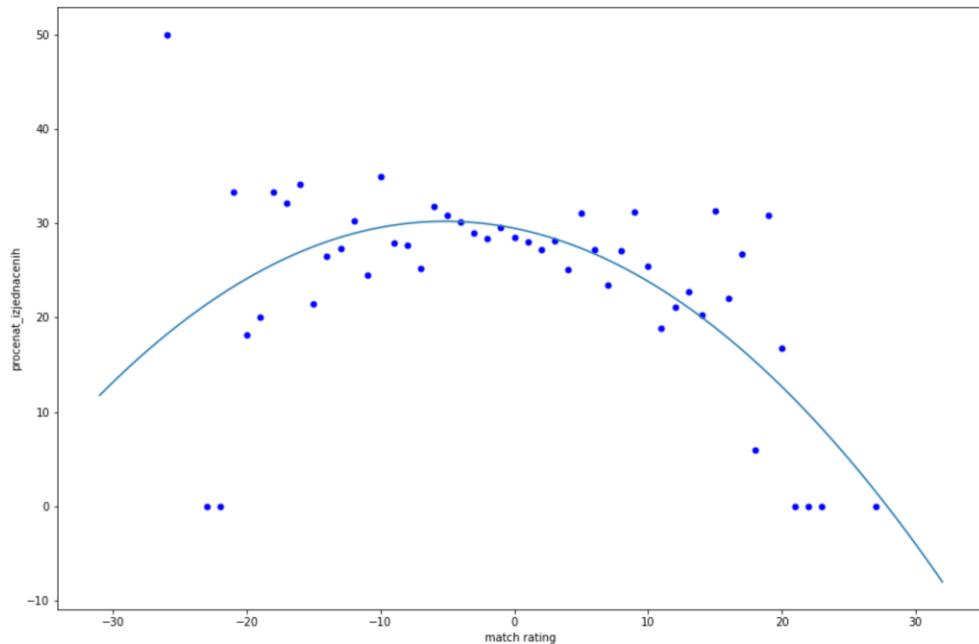


```
In [15]: x=match_rating
y=procenat_izjednacenih
p=lsquares(x, y, 2)
print(p)
plot_points(x,y,'match rating','procenat_izjednacenih')

xp=np.linspace(np.min(x)-5,np.max(x)+5,100);
plt.plot(xp,np.polyval(p,xp))

[-2.76800515e-02 -2.86197809e-01  2.94684440e+01]
```

Out[15]: [<matplotlib.lines.Line2D at 0x1b91303f978>]



```
In [16]: #ako je mr 15 kolike su sanse da bude izjednaceno.
np.polyval(p,15)
```

Out[16]: 18.947465266341055

## Algoritam linearne regresije

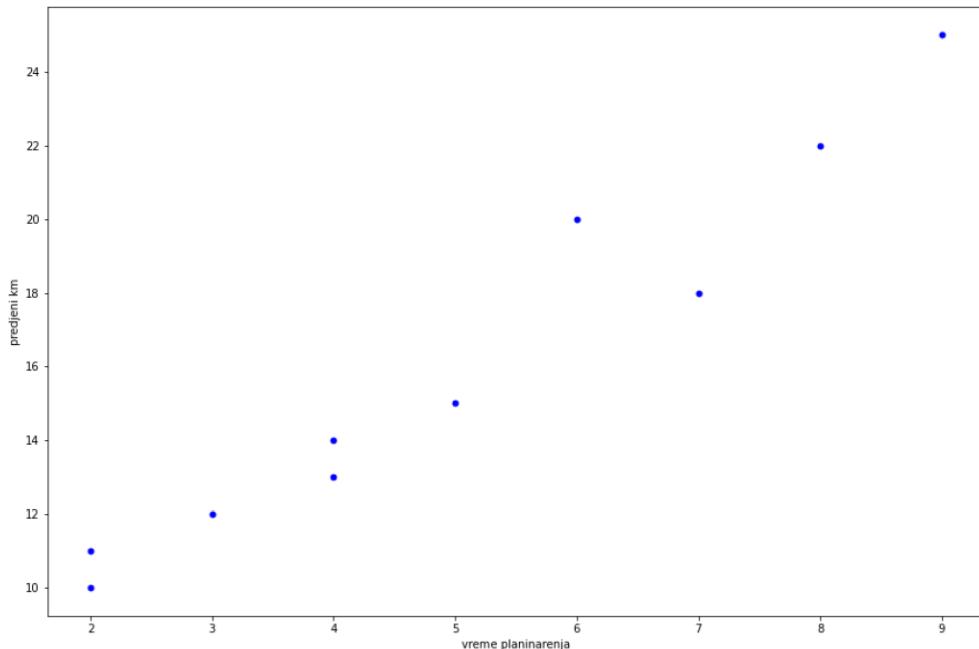
Iako je primer sa fudbalom zanimljiv, sam postupak linearne regresije objasnićemo na jednostavnijem primeru.

Dati su nam podaci o 10 planinara, o tome koliko su vremena planinarili u satima, i koliki su put prešli u kilometrima.

Naš zadatak je da, pomoću linearne regresije, proverimo da li u podacima postoji neki trend, odnosno da li se na osnovu baš ovih podataka može pronaći eventualna zavisnost između prednjeg puta i vremena planinarenja.

```
In [17]: vreme_planinarenja = np.array([2,2,3,4,4,5,6,7,8,9]).astype(float)
predjeni_km = np.array([10,11,12,13,14,15,20,18,22,25]).astype(float)
```

```
In [18]: plot_points(vreme_planinarenja,predjeni_km,'vreme planinarenja','predjeni km')
```



Krenućemo od najednostavnijeg oblika linearne regresije, uklapanje prave u podatke.

Dakle, cilj nam je da odredimo pravu na osnovu datih podataka.

Koliko koeficijenata ima prava? Koliko mi podataka imamo?

Očigledno je da određivanje 2 koeficijenta na osnovu 10 tačaka daje preodređen sistem jednačina.

Ako su date tačke  $(x_1, y_1), (x_2, y_2) \dots (x_{10}, y_{10})$ , a prava oblika  $y = kx + n$  onda sistem ima sledeći oblik:

$$\begin{aligned}kx_1 + n &= y_1 \\kx_2 + n &= y_2 \\kx_3 + n &= y_3 \\kx_4 + n &= y_4 \\kx_5 + n &= y_5 \\kx_6 + n &= y_6 \\kx_7 + n &= y_7 \\kx_8 + n &= y_8 \\kx_9 + n &= y_9 \\kx_{10} + n &= y_{10}\end{aligned}$$

Dakle, ne možemo da odredimo pravu liniju koja će proći kroz svih 10 tačaka jer prava nije dovoljno kompleksan model da može da obuhavati sve tačke (osim ako baš tačke nisu generisane pomoću prave).

U ovom slučaju dovoljno kompleksan model koji bi prošao kroz sve tačke je polinom devetog stepena. Hajde da nacrtamo taj polinom.

```
In [19]: def linterp(x,y):
    n = len(x)
    p = 0
    for i in range(n):
        L = 1
        for j in range(n):
            if i != j:
                L = np.convolve(np.array([1, -x[j]]), (x[i]-x[j]), 'same')
        p = p + y[i]*L
    return p

x=vreme_planinarenja
y=predjeni_km
p=linterp(x, y)

print(p)

[nan nan nan nan nan nan nan nan nan]
```

Zbog čega smo dobili grešku deljenja nulom u pozivu linterp?

Zato što  $x_1$  i  $x_2$  imaju istu vrednost (vrednost 2), odnosno za jedno  $x$  imamo dva različita  $y$ . Isti slučaj je i sa  $x_4$  i  $x_5$ .

Vec sada možemo videti zašto je interpolacija problematična ako imamo jako puno podataka. Vro rečko čete u velikoj količini podataka imati garanciju da čete za jedno  $x$  imate samo jedno  $y$ .

U našem konkretnom primeru dva planinara su hodala isto vreme, a prešla zraličit put što je normalna pojava na koju interpolacija nije spremna.

Izbacićemo sada problematične tačke i pogledati kako izgleda interpolacioni polinom.

```
In [20]: x=vreme_planinarenja
y=predjeni_km

x=np.delete(x, [0,3])
y=np.delete(y, [0,3])

print(x)
print(y)

p=linterp(x, y)

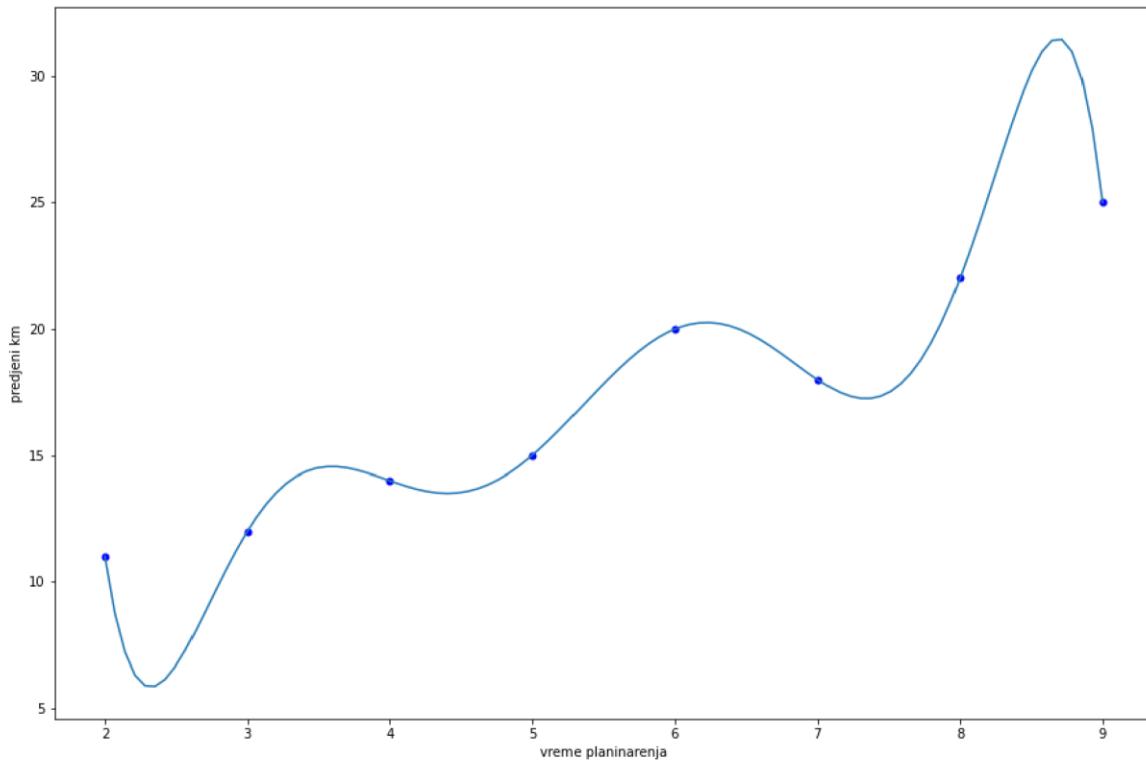
print(p)

plot_points(x,y, 'vreme planinarenja', 'predjeni km');

xp=np.linspace(np.min(x),np.max(x),100);
plt.plot(xp,np.polyval(p,xp))

[2. 3. 4. 5. 6. 7. 8. 9.]
[11. 12. 14. 15. 20. 18. 22. 25.]
[-2.9166667e-02 1.1083333e+00 -1.7458333e+01 1.4733333e+02
-7.17179167e+02 2.00705833e+03 -2.9778333e+03 1.8100000e+03]
```

Out[20]: [<matplotlib.lines.Line2D at 0xb9133ea748>]



Vidimo da interpolacioni polinom ima jako velike oscilacije i sigurno ne bi bio dobar predikcion model za broj pređenih kilometara na osnovu vremena hodanja.

Može da pokušamo i splajnovima.

```
In [21]: from scipy.interpolate import CubicSpline

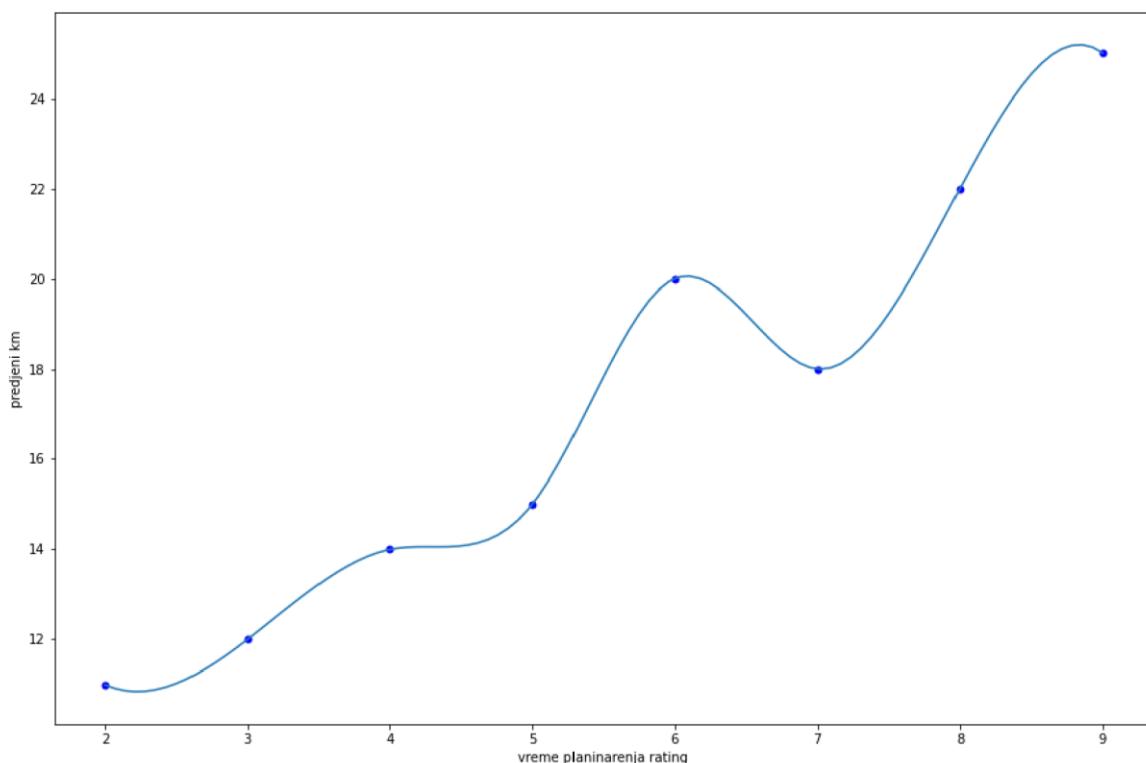
xp=np.linspace(1,np.max(x),100)

cs = CubicSpline(x, y) #'not-a-knot' (default)

plot_points(x,y,'vreme planinarenja rating','predjeni km')

xp=np.linspace(np.min(x),np.max(x),100)
plt.plot(xp,cs(xp))
```

Out[21]: [`<matplotlib.lines.Line2D at 0x1b9133484e0>`]



Rezultat je malo bolji, ali još uvek imamo oscilacije. Male promene vremena hodanja daju velike promene pređenog puta, što nije realno. Cilj nam je da imamo prediktivni model koji što više moguće oslikava realnost, odnosno koji će davati predikcije koje imaju smisla u odnosu na domen u kome radimo.

Iz svih do sada navednih razloga odustajemo od upotrebe interpolacije već koristimo drugačiji pristup koji se naziva regresija.

Vraćamo se našem sistemu od dve nepoznate i deset jednačina. Probaćemo da ga rešimo tako da rezultujuća prava ne prolazi kroz svih 10 tačaka već da se u njih uklopi nabolje što može.

Najbolje moguće uklapanje definisemo tako što ćemo tražiti da prava koju određujemo za date tačke ima najmanji moguć zbir kvadrata grešaka. Prvo ćemo definisati šta je greška za tačku.

Greška  $e_i$  za svaku datu tačku  $x_i$  predstavlja razliku između vrednosti koju dobijemo kada  $x_i$  zamenimo u pravu (koju određujemo) i vrednosti  $y_i$  iz podataka (koja odgovara tački  $x_i$ ).

Ako sa  $f(x) = kx + n$  označimo pravu koju određujemo, greška je definisana:

$$e_i = y_i - f(x_i)$$

Na sledećem grafiku, greška je vertikalno rastojanje između zelenih tačaka ( $y_i$ ) i crvenih tačaka ( $f(x_i)$ ).

Prava na grafiku je prava  $y = 2x + 6$  koja je određena pomoću regresije tako da minimizuje zbir kvadrata grešaka  $e_i$ .

```
In [22]: x=vreme_planinarenja
y=predjeni_km
p=lsquares(x, y, 1)

print(p)

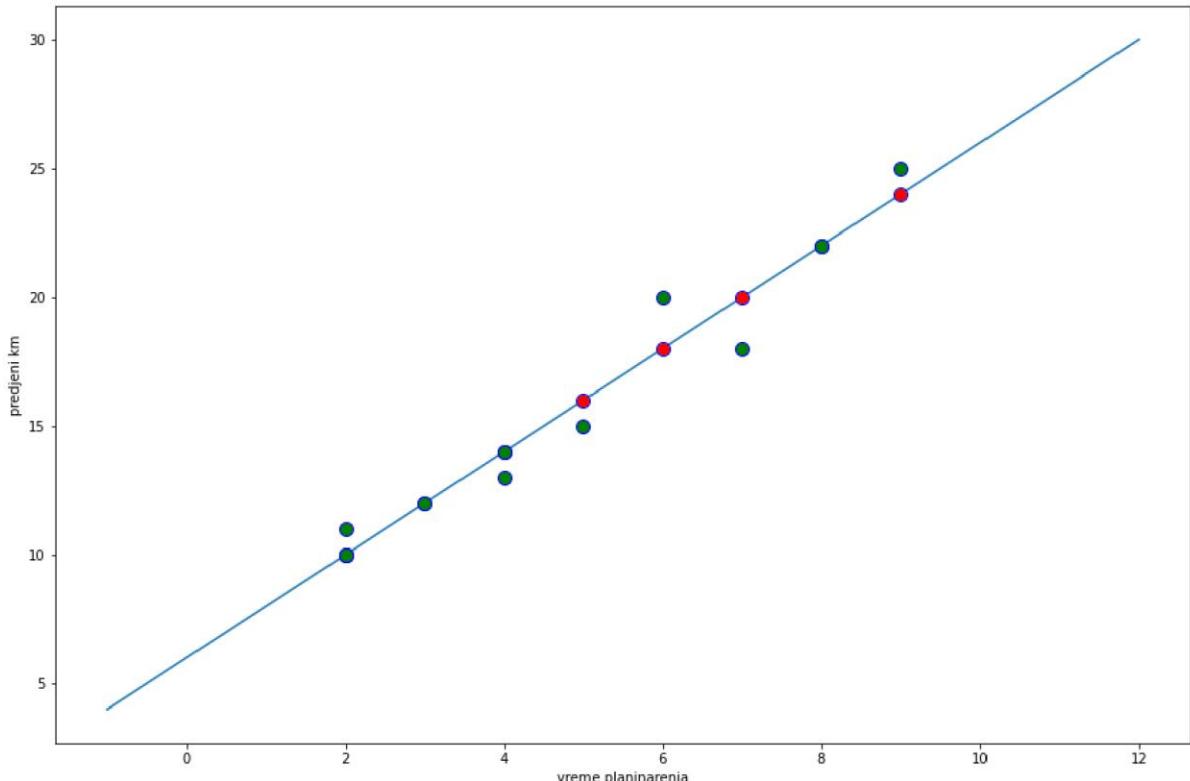
plot_points(x,y,'vreme planinarenja','predjeni km')

xp=np.linspace(np.min(x)-3,np.max(x)+3,100)
plt.plot(xp,np.polyval(p,xp))

plt.plot(x,np.polyval(p,x), 'ob', markersize=10, markerfacecolor='r')
plt.plot(x,y, 'ob', markersize=10, markerfacecolor='g')
```

[2. 6.]

Out[22]: [`<matplotlib.lines.Line2D at 0x1b912c2f320>`]



Dali smo definiciju greške za tačku, sada ćemo definisati ukupnu grešku.

Ukupna greška definiše se kao zbir kvadrata grešaka za tačke:

$$SSE = \sum_{i=1}^n (y_i - f(x_i))^2$$

gde je  $n$  ukupan broj datih tačaka, odnosno podataka (u našem slučaju 10).

Kvadrat greške koristimo zato što će nam olakšati posao prilikom traženja minimuma greške, kao što ćete videti u nastavku.

Šta mislite da li upotreba kvadrata greške umesto same greške može biti problematična? Kao pomoć, razmislite o tome kako izgledaju greške tačaka koje su atipične, tj. mnogo odstupaju od ostalih tačaka (npr. iskusan planinar u skupu sa "običnim" ljudima).

Da ponovimo sada naš cilj, tražimo pravu, tj. koeficijente  $k$  i  $n$  takve da je vrednost SSE minimalna.

To je problem traženja minimuma funkcije u 3d.

```
In [23]: import plotly.express as px
import sys

tmp_xx=np.linspace(-10,15,100)

xx, yy = np.meshgrid(tmp_xx, tmp_xx, sparse=False, indexing='ij')

SSE = np.zeros([len(xx),len(xx)])
SSE_as_list = []

for i in range(len(xx)):
    for j in range(len(xx)):
        pred = xx[i,j]*x + yy[i,j]
        tmp_list = [xx[i,j],yy[i,j],np.sum((pred-y)**2)]
        SSE_as_list.append(tmp_list)

SSE_as_matrix = np.matrix(SSE_as_list)

tmp_df=pd.DataFrame(data=SSE_as_matrix,columns=['x','y','SSE'])

fig = px.scatter_3d(tmp_df, x='x', y='y', z='SSE')

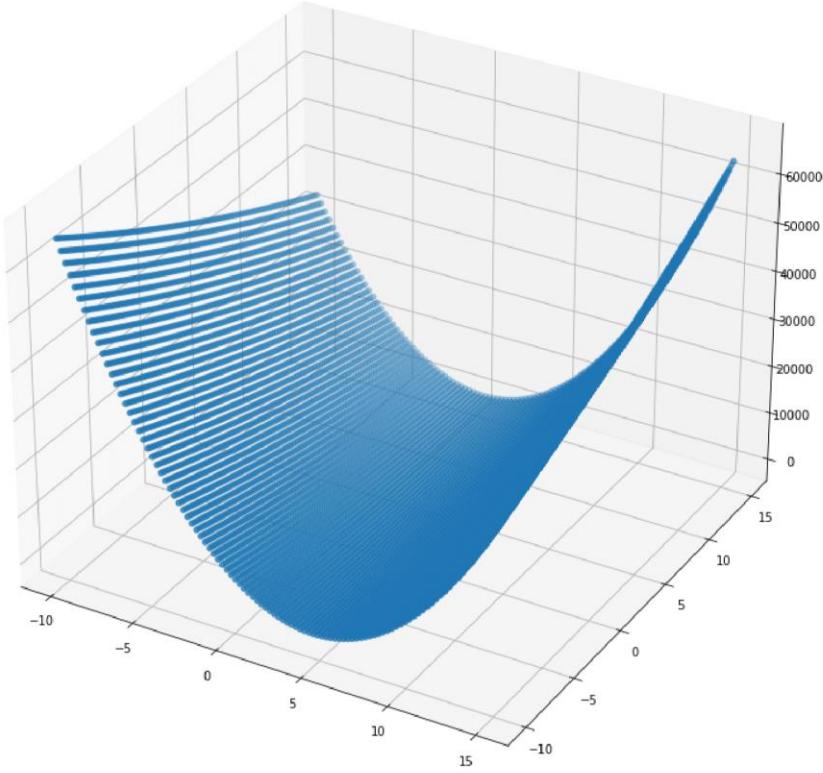
fig.show()
```

```
In [49]: from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(15, 10))
ax = Axes3D(fig, auto_add_to_figure=False)
fig.add_axes(ax)

ax.scatter(tmp_df['x'], tmp_df['y'], tmp_df['SSE'])
```

```
Out[49]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x1b925e3a390>
```



Na osnovu rezultata prethodnog koda vidimo da je minimum za  $k = 2$  i  $n = 6$ , a minimalna vrednost SSE je 12.

Minimum funkcije  $SSE(k, n)$  tražimo analitički pomoću parcijalnih izvoda.

Oređujemo parcijalne izvode  $SSE(k, n)$  po  $k$  i po  $n$  i izjednačavamo ih sa 0.

$$\begin{aligned}\frac{\delta SSE(k, n)}{\delta k} &= 0 \\ \frac{\delta SSE(k, n)}{\delta n} &= 0\end{aligned}$$

$$\begin{aligned}SSE &= \sum_{i=1}^n (y_i - f(x_i))^2 \\ SSE &= \sum_{i=1}^n (y_i - (kx_i + n))^2 \\ SSE &= \sum_{i=1}^n (y_i - kx_i - n)^2\end{aligned}$$

$$\frac{\delta SSE(k, n)}{\delta k} = 2 \sum_{i=1}^n (y_i - kx_i - n)(-x_i) = 0$$

$$\frac{\delta SSE(k, n)}{\delta n} = 2 \sum_{i=1}^n (y_i - kx_i - n)(-1) = 0$$

Sredićemo sada malo poslednje dve jednačine:

$$\sum_{i=1}^n x_i^2 \cdot k + \sum_{i=1}^n x_i \cdot n = \sum_{i=1}^n y_i x_i$$

$$\sum_{i=1}^n x_i \cdot k + \sum_{i=1}^n 1 \cdot n = \sum_{i=1}^n y_i$$

Dobili smo sistem od dve jednačine sa dve nepoznate gde sve vrednosti suma možemo da odredimo iz podataka.

Rešavamo sistem za naš primer sa planinarima.

```
In [25]: print(x)
print(y)
sum_xi_2=np.sum(x**2)
sum_xi=np.sum(x)
sum_xi_yi=np.sum(x*y)
sum_1=np.sum(np.ones([1,len(x)]))
sum_yxi=np.sum(y)

print([sum_xi_2, sum_xi, sum_xi_yi, sum_1, sum_yxi])

[2. 2. 3. 4. 4. 5. 6. 7. 8. 9.]
[10. 11. 12. 13. 14. 15. 20. 18. 22. 25.]
[304.0, 50.0, 908.0, 10.0, 160.0]
```

Dobili smo sledeći sistem:

$$\begin{aligned} 304 \cdot k + 50 \cdot n &= 908 \\ 50 \cdot k + 10 \cdot n &= 160 \end{aligned}$$

Rešavamo sistem:

```
In [26]: A=np.array([[304.,50.],[50.,10.]])
b=np.array([908.,160.])
x=la.solve(A,b)

print(x)
```

[2. 6.]

Rešenje je  $k=2$  i  $n=6$ . To znači da se u naše podatke najbolje uklapa prava:

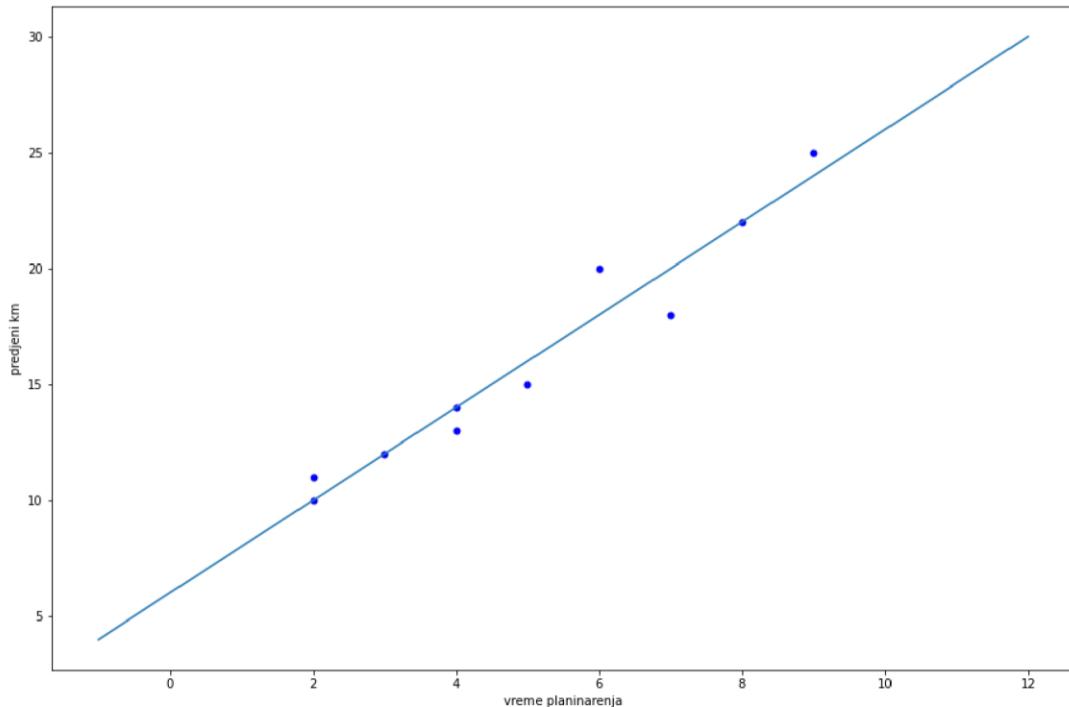
$$f(x) = 2x + 6$$

Crtamo podatke i pravu.

```
In [27]: x=vreme_planinarenja
y=predjeni_km
plot_points(x,y,'vreme planinarenja','predjeni km');

xp=np.linspace(np.min(x)-3,np.max(x)+3,100)
plt.plot(xp,2*xp+6)
```

Out[27]: [<matplotlib.lines.Line2D at 0x1b9244a3cc0>]



```
In [28]: SSE = np.sum( (2*x+6-y)**2)
print(SSE)
```

12.0

Vidimo da je ukupna kvadratna greška 12. Greška se meri u kvadratima jedinice podatka. Dakle, naša greška je  $12 \text{ km}^2$ .

Da li je greška koju smo dobili mala, odnosno da li naš model (prava) radi dobro?

Svaki odgovor na ovo pitanje bio bi subjektivan.

Da bi dobili objektivan odgovor moramo da uporedimo pravu sa nekim drugim modelom na istim podacima.

Takvi modeli se tipično nazivaju osnovni modeli (*baseline models*).

Osnovni model u našem slučaju odabraćemo tako što ćemo se pretvarati da nemamo  $x$  odnosno da nemamo informaciju o vremenu planinarenja, već samo o pređenom putu.

Ako nas neko pita da damo predikciju o tome koliko kilometara je prešao neki planinar, a mi pitamo koliko vremena je hodao, i dobijemo odgovor "ne znam", sve što bi mi mogli da kažemo je: "tipičan planinar po našem skupu podataka pređe 16km".

Odakle nam 16km? To je prosečna vrednost za  $y$  koju ćemo označiti sa  $\bar{y}$ .

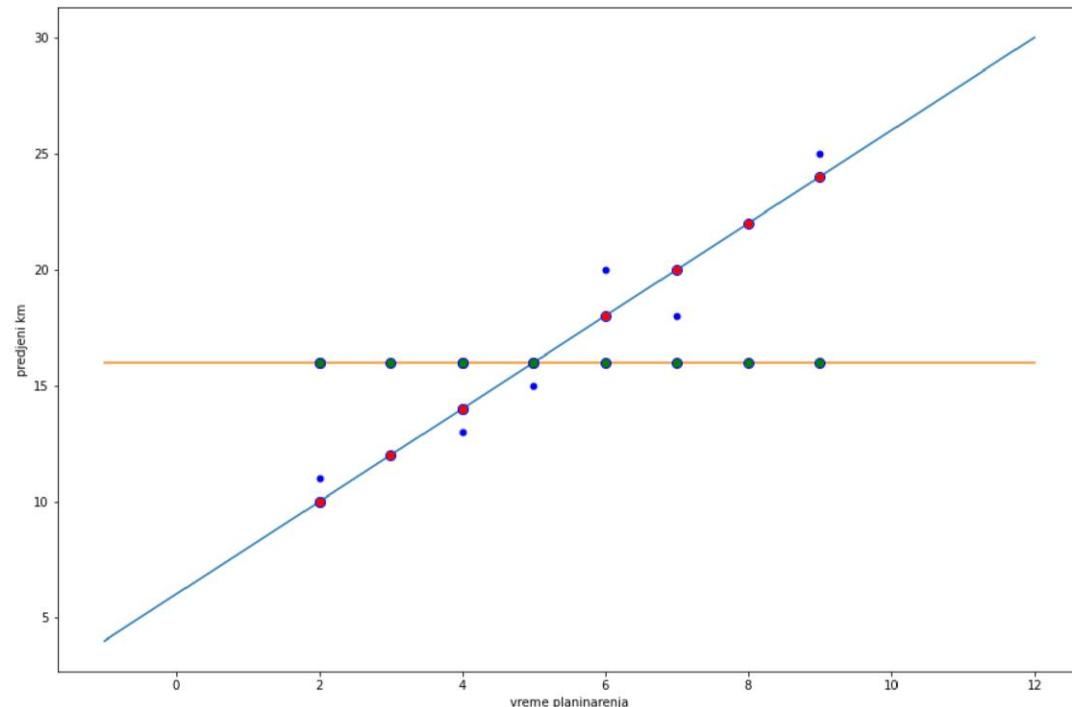
In [29]: `print(np.mean(y))`

16.0

Poredimo sada predikcije naše prave sa prosekom odnosno pravom  $y = 16$  koja za svako  $x$  predviđa uvek vrednost 16km.

In [30]: `x=vreme_planinarenja  
y=predjeni_km  
plot_points(x,y,'vreme planinarenja','predjeni km')  
  
xp=np.linspace(np.min(x)-3,np.max(x)+3,100);  
plt.plot(xp,2*xp+6)  
  
plt.plot(xp,np.ones(len(xp))*16)  
  
plt.plot(x,y,'ob', markersize=8, markerfacecolor='r')  
plt.plot(x,np.ones(len(x))*16, 'ob', markersize=8, markerfacecolor='g')`

Out[30]: [`<matplotlib.lines.Line2D at 0x1b92471de48>`]



Na prvi pogled naša prava ima bolje predikcije, ali ćemo to ipak morati da formalizujemo.

Definišemo sada dve sume. Prva meri koliko vrednosti  $y$  odstupaju od svog proseka  $\bar{y}$ , odnosno koliko je prosek dobar prediktioni model:

$$SST = \sum_{i=1}^n (y - \bar{y})^2$$

Oznaka  $SST$  je iz literature i označava tolanu sumu kvadrata (*sum of squares total*). U nastavku ćemo reći zašto se koristi termin "totalna".

```
In [31]: SST=np.sum((y-np.mean(y))**2)
print(SST)
228.0
```

Vidimo da za naš primer  $SST$  ima vrednost  $228\text{km}^2$ . Na osnovu toga već možemo da tvrdimo da naš model (prava) radi jer je  $SSE$  mnogo manje od  $SST$ . U nastavku ćemo pokazati način na koji možemo da kvalitet modela izmerimo u procentima i time dobijemo meru kvaliteta modela koja ne zavisi od jedinice merenja  $y$ .

Računamo sada razliku između predikcija naše prave i  $y = \bar{y}$ :

$$SSR = \sum_{i=1}^n (f(x_i) - \bar{y})^2$$

Oznaka  $SSR$  je iz literature i označava sumu kvadrata koja je rezultat regresije (*sum of squares due to regression*), odnosno pokazuje koliko se regresija razlikuje od proseka. Na slici iznad  $SSR$  je vertikalno rastojanje između crevenih i zelenih tačaka.

```
In [32]: SSR=sum((2*x+6-np.mean(y))**2)
print(SSR)
216.0
```

Sada kada smo definisali  $SSR$ , objasnićemo zašto se  $SST$  zove totalna suma. Pod totalnom sumom misli se na ukupnu sumu rastojanja: od proseka do prave + od prave do  $y$  vrednosti iz podataka:

$$SST = SSR + SSE$$

Rastojanje od proseka do vrednosti  $y$  je zbir rastojanja od proseka do prave + od prave do  $y$ .

### Koeficijent determinacije

Kvalitet našeg modela izmerićemo pomoću koeficijenta determinacije ( $r^2$ ) koji poredi koliko je naša prava (dobijena regresijom) doprinela tačnijoj predikciji  $y$  u odnosu na prosek:

$$r^2 = \frac{SSR}{SST}$$

Koeficijent determinacije meri koliki procenat varijabilnosti (rastojanja)  $y$  u odnosu na svoj prosek može da se objasni (predviđi) pomoću regresije za  $x$ .

Za naš konkretni primer to bio procenat varijabilnosti pređenog puta koji je objašnjen na osnovu regresione prave po vremenu hodanja.

Pokazaćemo sada koje su dve ekstremne vrednosti za  $r^2$ .

Ako je predikcija našeg modela uvek prosek, tj. rezultat regresije je prava koja takođe vraća uvek prosečnu vrednost, tada je  $r^2 = 0$  jer:

$$\begin{aligned} f(x_i) &= \bar{y} \\ SSR &= \sum_{i=1}^n (f(x_i) - \bar{y})^2 = \sum_{i=1}^n (\bar{y} - \bar{y})^2 = 0 \\ r^2 &= \frac{SSR}{SST} = \frac{0}{SST} = 0 \end{aligned}$$

Ako je predikcija našeg modela uvek  $y$ , tj. rezultat regresije je prava koja prolazi kroz sve tačke  $y$ , tada je  $r^2 = 1$  jer:

$$\begin{aligned} f(x_i) &= y \\ SSR &= \sum_{i=1}^n (f(x_i) - \bar{y})^2 = \sum_{i=1}^n (y - \bar{y})^2 = SST \\ r^2 &= \frac{SSR}{SST} = \frac{SST}{SST} = 1 \end{aligned}$$

Izračunavamo  $r^2$  za naš primer sa planinarima.

```
In [33]: print(SSE)
print(SSR)
print(SST)
12.0
216.0
228.0
```

```
In [34]: r_2=SSR/SST
print(r_2)
0.9473684210526315
```

Vidimo da je vrednost dosta blizu 1, odnosno da je ~95% varijabilnosti pređenog puta u našem skupu podatka objašneno pomoću regresije za vreme hodanja.

To znači da je naš model jako dobar.

## Linearna regresija za polinom proizvoljnog stepena

Do sada smo pokazali na koji način se pomoću linearne regresije u podatke može uklopiti prava.

Sada ćemo pokzati opšti postupak pomoću koga se u podatke može uklopiti polinom proizvoljnog stepena  $m$ :

$$p(x) = \sum_{j=0}^m a_j x^j$$

Koristimo isti postupak kao i ranije. Tražimo minimum  $SSE$ , ali ovaj put ne za 2 parametra nego za  $m+1$ .

Napomena: u nastavku su istaknuti najvažniji koraci postupka. Kompletan postupak dat je u udžbeniku i spada u informativni deo gradiva.

$$\begin{aligned}\frac{\delta SSE}{\delta a_0} &= 0 \\ \frac{\delta SSE}{\delta a_1} &= 0 \\ &\dots \\ \frac{\delta SSE}{\delta a_m} &= 0\end{aligned}$$

$$\frac{\sum_{i=1}^n (y_i - \sum_{j=0}^m a_j x_i^j)^2}{\delta a_0} = 0$$

$$\frac{\sum_{i=1}^n (y_i - \sum_{j=0}^m a_j x_i^j)^2}{\delta a_1} = 0$$

$$\frac{\sum_{i=1}^n (y_i - \sum_{j=0}^m a_j x_i^j)^2}{\delta a_m} = 0$$

$$2 \sum_{i=1}^n (y_i - \sum_{j=0}^m a_j x_i^j) (-x_i^0) = 0$$

$$2 \sum_{i=1}^n (y_i - \sum_{j=0}^m a_j x_i^j) (-x_i^1) = 0$$

$$2 \sum_{i=1}^n (y_i - \sum_{j=0}^m a_j x_i^j) (-x_i^m) = 0$$

$$\sum_{i=1}^n (\sum_{j=0}^m a_j x_i^j - y_i) x_i^0 = 0$$

$$\sum_{i=1}^n (\sum_{j=0}^m a_j x_i^j - y_i) x_i^1 = 0$$

$$\sum_{i=1}^n (\sum_{j=0}^m a_j x_i^j - y_i) x_i^m = 0$$

$$\sum_{i=1}^n (\sum_{j=0}^m a_j x_i^j) x_i^0 = \sum_{i=1}^n y_i x_i^0$$

$$\begin{aligned}
\sum_{i=1}^n (\sum_{j=0}^m a_j x_i^j) x_i^1 &= \sum_{i=1}^n y_i x_i^1 \\
&\dots \\
\sum_{i=1}^n (\sum_{j=0}^m a_j x_i^j) x_i^m &= \sum_{i=1}^n y_i x_i^m \\
&\dots \\
\sum_{j=0}^m \sum_{i=0}^n x_i^j x_i^0 a_j &= \sum_{i=1}^n y_i x_i^0 \\
\sum_{j=0}^m \sum_{i=0}^n x_i^j x_i^1 a_j &= \sum_{i=1}^n y_i x_i^1 \\
&\dots \\
\sum_{j=0}^m \sum_{i=0}^n x_i^j x_i^m a_j &= \sum_{i=1}^n y_i x_i^m
\end{aligned}$$

Kao malu pomoć u daljem izvođenju prikazaćemo sumu  $\sum_{j=0}^m \sum_{i=0}^n x_i^j x_i^0 a_j$  po sabircima (suma ima  $m$  sabiraka):

$$\sum_{j=0}^m \sum_{i=0}^n x_i^j x_i^0 a_j = \sum_{i=0}^n x_i^0 x_i^0 a_0 + \sum_{i=0}^n x_i^1 x_i^0 a_1 + \dots + \sum_{i=0}^n x_i^m x_i^0 a_m$$

Ako posmatramo na primer  $x^0$  i  $x^1$  vidimo da je suma  $\sum_{i=0}^n x_i^0 x_i^1$  ustvari skalarni proizvod ta dva vektora koji se može dobiti tako što se jedan od vektora uzme kao vrsta, a drugi kao kolona i onda se primeni množenje matrica:

$$\left[ \begin{array}{cccc} x_1^1 & x_2^1 & \dots & x_n^1 \end{array} \right]_{1 \times n} * \begin{bmatrix} x_1^0 \\ x_2^0 \\ \dots \\ x_n^0 \end{bmatrix}_{n \times 1} = \sum_{i=1}^n x_i^0 x_i^1$$

To znači da se sve sume  $\sum_{i=0}^n x_i^j x_i^k$  gde je  $k = 0, 1, 2, \dots, m$  iznad mogu dobiti kao proizvod matrica:

$$A^T * A$$

$$A^T = \left[ \begin{array}{cccc} x_1^0 & x_2^0 & \dots & x_n^0 \\ x_1^1 & x_2^1 & \dots & x_n^1 \\ \dots \\ x_1^m & x_2^m & \dots & x_n^m \end{array} \right]_{m+1 \times n} \quad A = \left[ \begin{array}{cccc} x_1^0 & x_1^1 & \dots & x_1^m \\ x_2^0 & x_2^1 & \dots & x_2^m \\ \dots \\ x_n^0 & x_n^1 & \dots & x_n^m \end{array} \right]_{n \times m+1}$$

Ako sada u proizvod matrica  $A^T A$  ubacimo i vektor nepoznatih koeficijenata polinoma  $a = (a_0, a_1, \dots, a_m)$  sve sume oblike  $\sum_{j=0}^m \sum_{i=0}^n x_i^j x_i^k a_j$  gde je  $k = 0, 1, 2, \dots, m$  mogu se dobiti kao sledeći proizvod matrica:

$$A^T * A * a$$

Dodajemo dimenzionalnosti matrica i vektora u proizvod:

$$A^T = \left[ \begin{array}{cccc} x_1^0 & x_2^0 & \dots & x_n^0 \\ x_1^1 & x_2^1 & \dots & x_n^1 \\ \dots \\ x_1^m & x_2^m & \dots & x_n^m \end{array} \right]_{m+1 \times n} \quad A = \left[ \begin{array}{cccc} x_1^0 & x_1^1 & \dots & x_1^m \\ x_2^0 & x_2^1 & \dots & x_2^m \\ \dots \\ x_n^0 & x_n^1 & \dots & x_n^m \end{array} \right]_{n \times m+1} * \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_m \end{bmatrix}_{m+1 \times 1}$$

Na sličan način, sve sume oblike  $\sum_{i=1}^n y_i x_i^k$  gde je  $k = 0, 1, 2, \dots, m$  mogu se dobiti kao proizvod vektora  $y = (y_1, y_2, \dots, y_n)$  i matrice  $A^T$ :

$$A^T = \left[ \begin{array}{cccc} x_1^0 & x_2^0 & \dots & x_n^0 \\ x_1^1 & x_2^1 & \dots & x_n^1 \\ \dots \\ x_1^m & x_2^m & \dots & x_n^m \end{array} \right]_{m+1 \times n} * \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}_{n \times 1}$$

Iz prethodnog dobijamo da se problem određivanja polinoma:

$$p(x) = \sum_{j=0}^m a_j x^j$$

koji se najbolje ukalpa u podatke  $(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)$  svodi na rešavanje sledećeg sistema:

$$(A^T A)a = A^T y$$

Napisaćemo sada kod koji za date podatke i stepen polinoma m formira matricu A i rešava sistem prikazan u prethodom redu.

```
In [35]: def lsquares(x,y,stepen):
    n=len(x)
    A=np.zeros([n,stepen+1])

    for i in range(n):
        for j in range(stepen+1):
            A[i,j]=x[i]**j

    p=la.solve(np.matmul(A.T, A), np.matmul(A.T, y.T))
    p=np.flip(p)

    return p
```

```
In [36]: print(np.flip([0,1,2,3,4]))
```

```
[4 3 2 1 0]
```

```
In [37]: x=vreme_planinarenja
y=predjeni_km

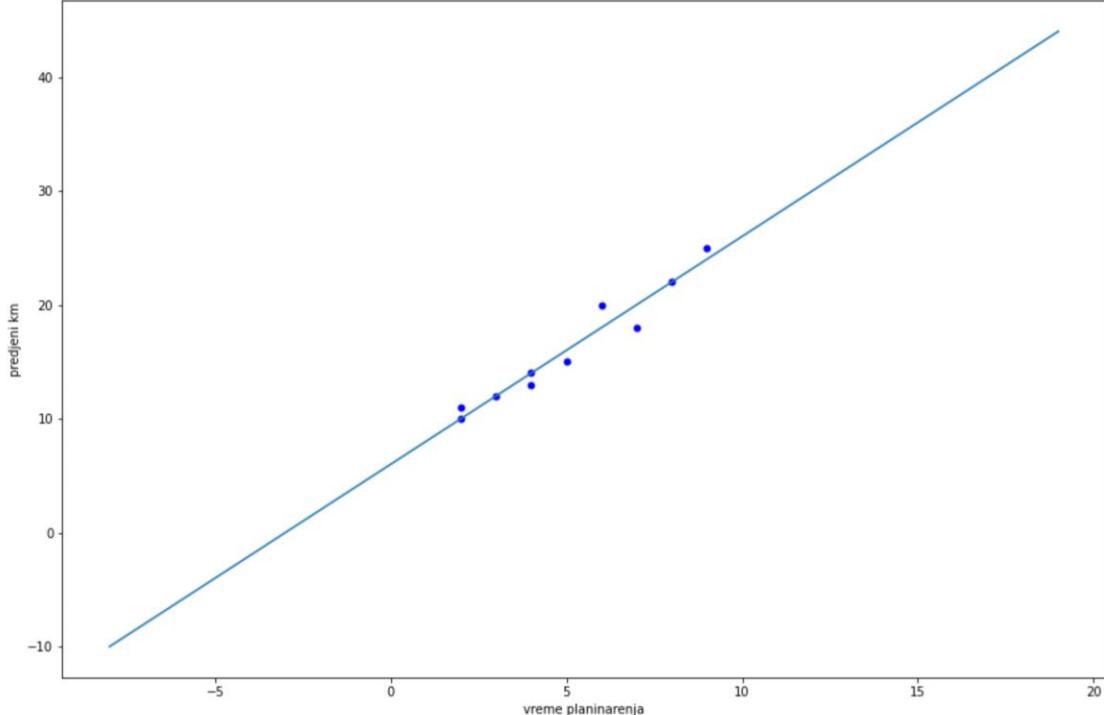
p=lsquares(x,y,1)

print(p)
```

```
[2. 6.]
```

```
In [38]: plot_points(x,y,'vreme planinarenja','predjeni km')
xp=np.linspace(np.min(x)-10,np.max(x)+10,100)
plt.plot(xp,np.polyval(p,xp))
```

```
Out[38]: []
```



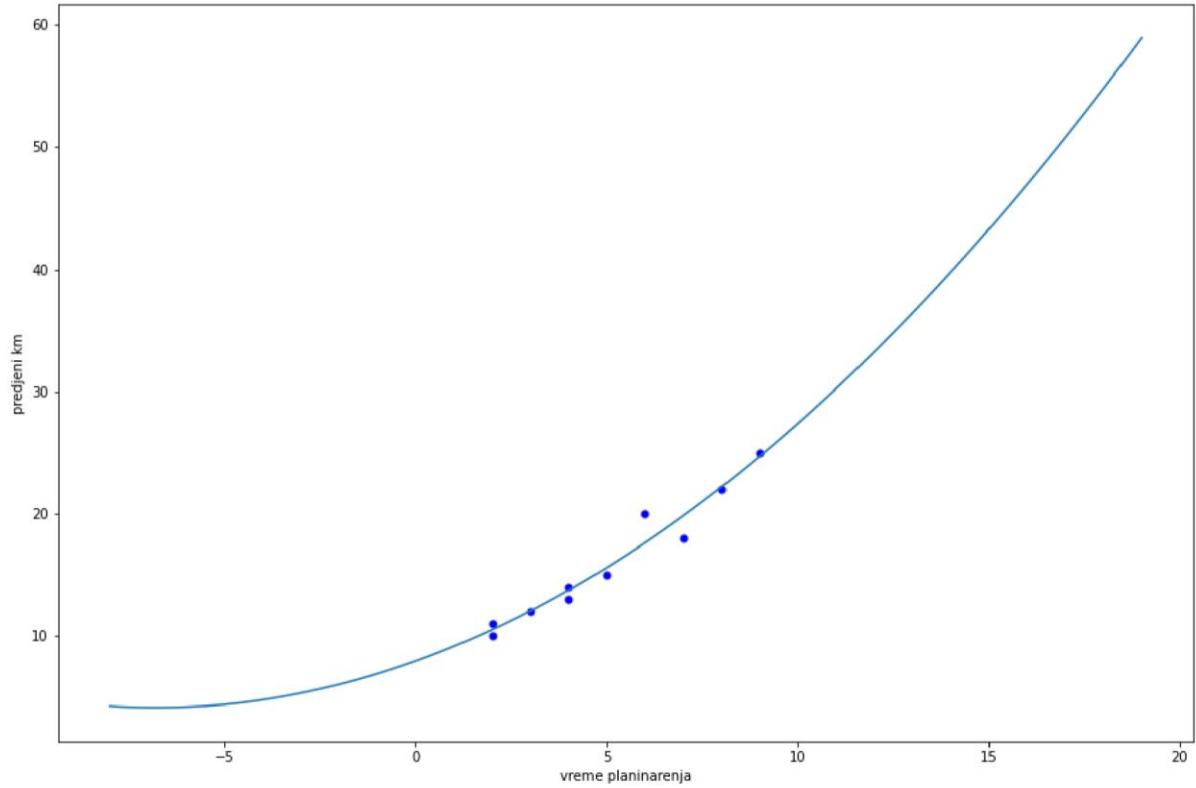
```
In [39]: p=lsquares(x,y,2)
print(p)

plot_points(x,y,'vreme planinarenja','predjeni km')

xp=np.linspace(np.min(x)-10,np.max(x)+10,100)
plt.plot(xp,np.polyval(p,xp))
```

```
[0.08241758 1.12087912 7.89010989]
```

```
Out[39]: []
```



Linearna regresija pomoću polinoma je samo specijalan slučaj linearne regresije za funkcije oblika  $y = x^i$ . Zašto?

Rezultat linearne regresije je linearna kombinacija koeficijenata i funkcija. Funkcije zadajemo, a koeficijente određujemo rešavanjem sistema koji formiramo na način na koji smo pokazali iznad.

Linearna regresija može se raditi za bilo koje funkcije  $f_1(x), f_2(x), \dots, f_m(x)$ , a tada matrica ima oblik:

$$A = \begin{bmatrix} f_0(x_1) & f_1(x_1) & \dots & f_m(x_1) \\ f_0(x_2) & f_1(x_2) & \dots & f_m(x_2) \\ \dots \\ f_0(x_n) & f_1(x_n) & \dots & f_m(x_n) \end{bmatrix}$$

Funkcija koja je data u nastavku deo je informativnog gradiva, a pomoću nje je moguće uraditi linearu regresiju za proizvoljan niz funkcija.

```
In [40]: def lsquares_general(x,y,functions):
    n=len(x)
    m=len(functions)
    A=np.zeros([n,m])
    for i in range(n):
        for j in range(m):
            A[i,j]=functions[j](x[i])

    p=la.solve(np.matmul(A.T, A), np.matmul(A.T, y.T))

    return p
```

```
In [41]: x=vreme_planinarenja
y=predjeni_km

p=lsquares_general(x,y,[lambda x: x,lambda x: 1])

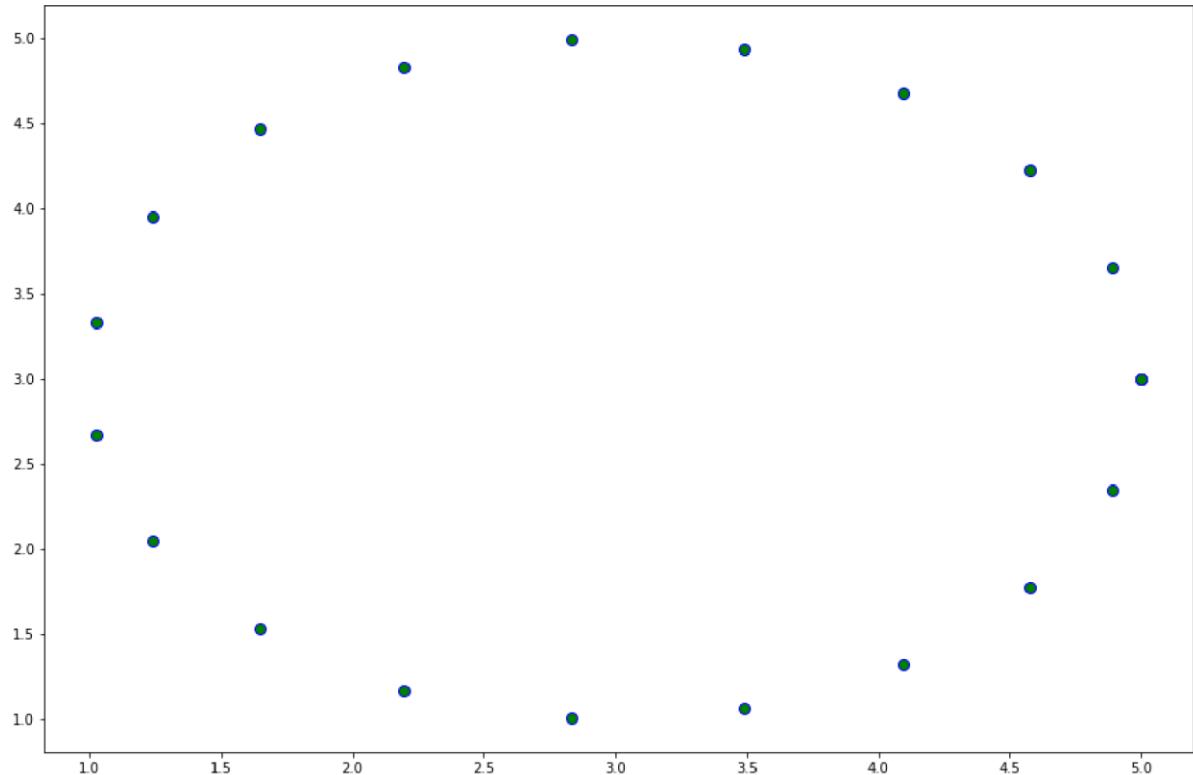
print(p)
[2. 6.]
```

```
In [42]: x_coor=y_coor=3 #centra kruga
r=2 #poluprečnik
t = np.linspace(0,2*np.pi,20).T
circsx = r*np.cos(t) + x_coor
circsy = r*np.sin(t) + y_coor

fig = plt.figure(figsize=(15, 10))
plt.plot(circsx,circsy,'ob', markersize=8, markerfacecolor='g')

#za objašnjenje pogledati definiciju sinusa i konsinus na jediničnom krugu
#za tačku (x,y) na jediničnom krugu koja odgovara ugлу t važi x=sin(t), a y=sin(t)
#pomeramo koordinate za zadati centar i množimo sa r ako ne želimo baš jedinični krug
```

Out[42]: [<matplotlib.lines.Line2D at 0x1b924553080>]



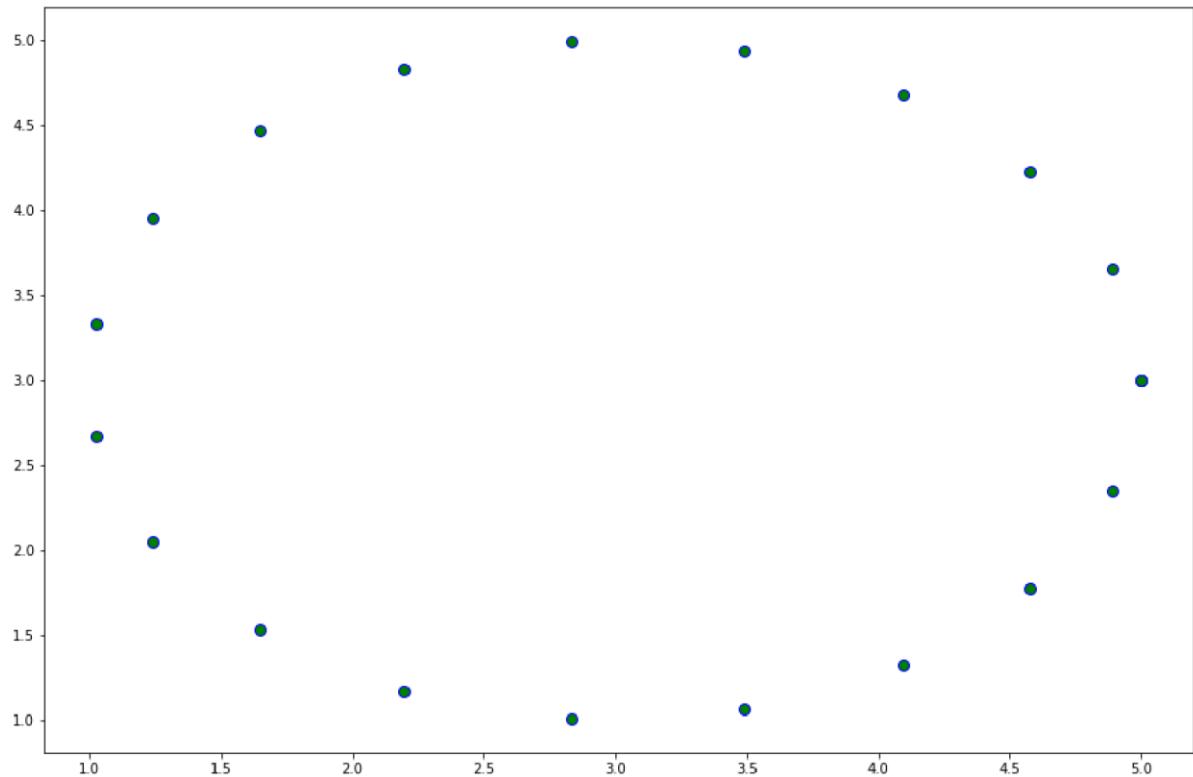
```
In [43]: x=np.linspace(0,2*np.pi,20).T
y=circsy.T
```

```
In [44]: aproks=lsquares_general(x,y,[lambda x: np.sin(x), lambda x: 1]) #u ovom slučaju znamo da je vrednost y
dobijena pomoću funkcije sin(x) pa prosleđujemo tu funkciju kao deo ulaznih parametara
```

```
In [45]: fig = plt.figure(figsize=(15, 10))

plt.plot(circsx,circsy,'ob', markersize=8, markerfacecolor='g')
```

Out[45]: [<matplotlib.lines.Line2D at 0x1b9245c6668>]



```
In [46]: fig = plt.figure(figsize=(15, 10))

xp=np.linspace(np.min(x)-1,np.max(x)+1,100)
circsxp = r*np.cos(xp) + x_coor

tmp = aproks[0]*np.sin(xp)+aproks[1] #koristimo rezultate regresije da dobijemo vrednosti za y, odnosno
#y-koordinate tačaka koje odgovaraju circxp koordinatama

plt.plot(circsxp,tmp, 'ob', markersize=8, markerfacecolor='r')
```

Out[46]: [<matplotlib.lines.Line2D at 0x1b924697748>]

