Добрый вечер

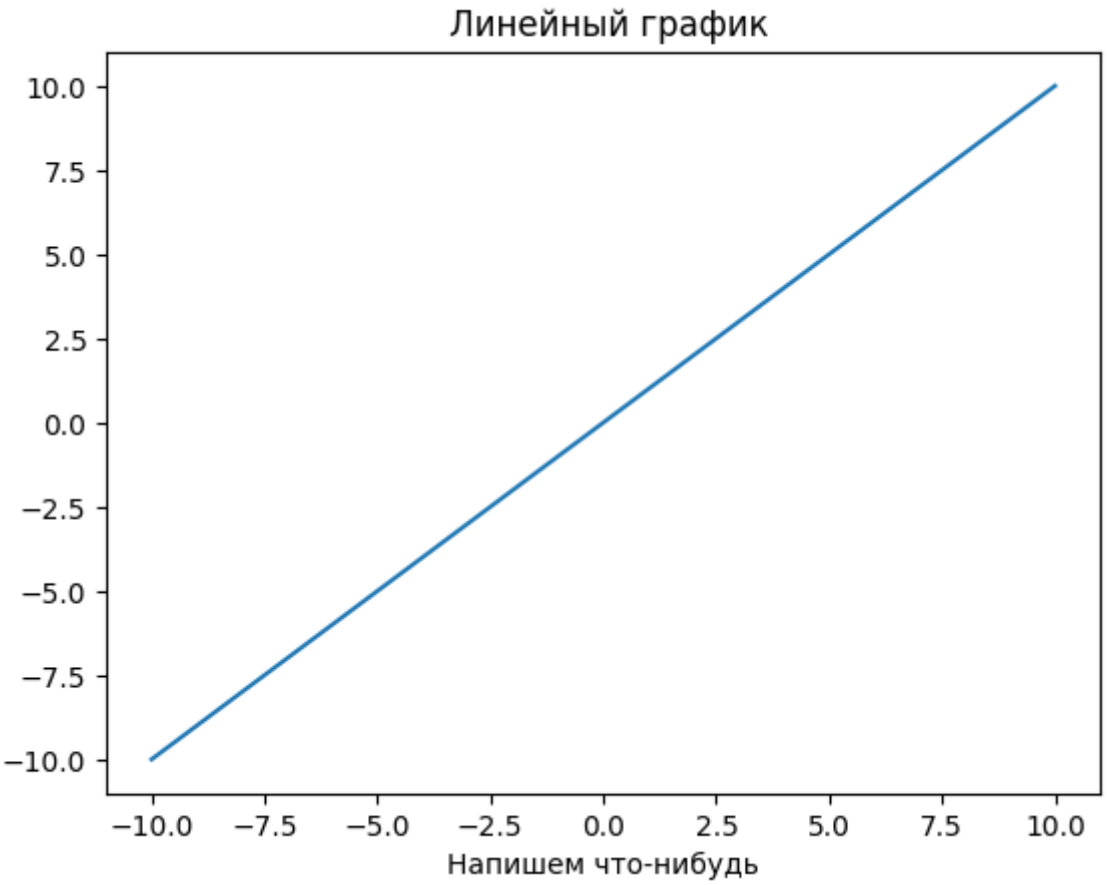
подготовил Воронин Сергей

Сегодня, как уже можно было догадаться, мы переходим на что-то новое. Сложность потихоньку растет, тренды каждый день меняются, и если вчера у нас была линейная регрессия, когда мы довольно понятные зависимости меняли на благородную прямую, это если что называется Линейная регрессия, то сегодня бы уже перейти на что-то более интересное, а именно **Полиномиальная регрессиия**. Задание сегодня будет небольшим, идея в том чтобы быстро и понятно разобрать суть того, что эта регрессия дает, и какой ее смысл. А смысл-то прост - заменить что-то более сложное, когда уже прямая нам не подходит, на степенную функцию. Простым примером степенной функции будет:

$$y = 0.1x + 2.5x^2 + 0.5x^3$$

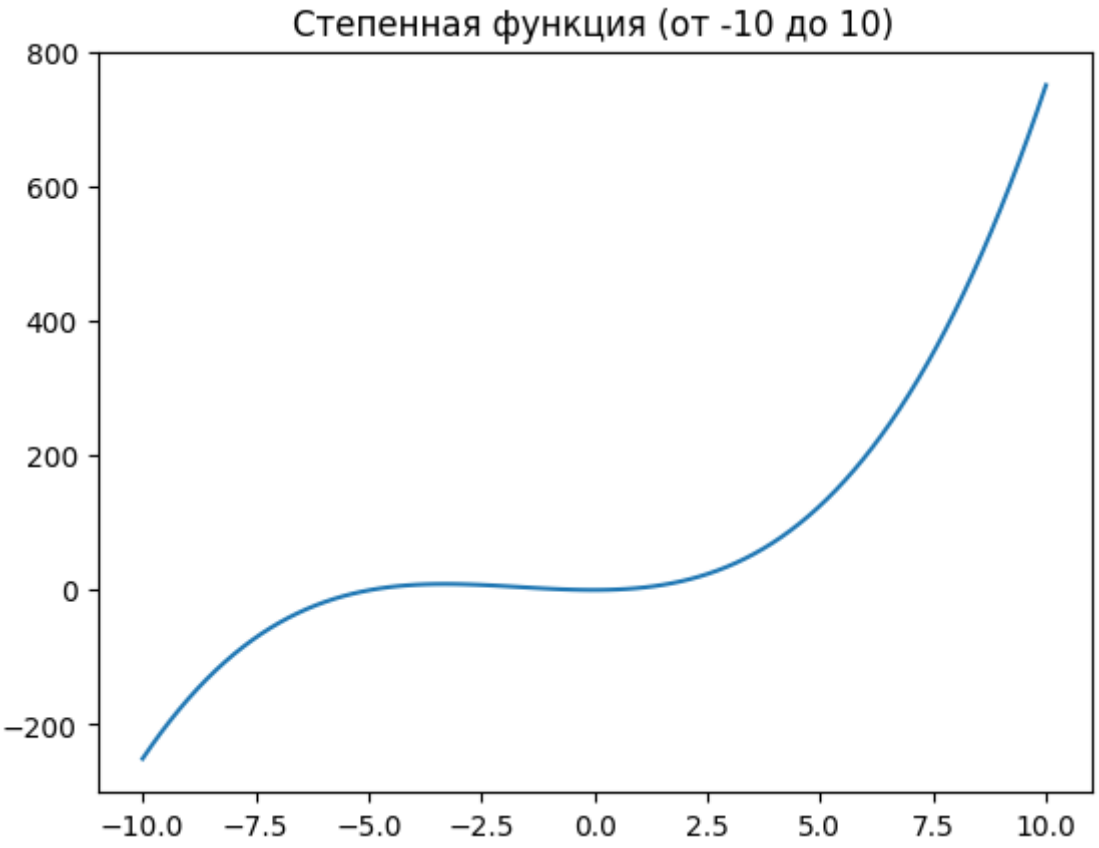
Но к ней мы придем чуть позже, а для начала вообще вспомним, как строить графики.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
'''
%matplotlib inline  указывает, что график необходимо построить все в той
же оболочке Jupyter,
#но теперь он выводится как обычная картинка. Данный способ удобен тем,
что позволяет проводить очень много экспериментов в рамках одного окна
(точнее web-страницы).
'''
x=np.linspace(-10,10,num=100) # генерирует все точки в заданном интервале
y=x
plt.plot(x,y)
plt.xlabel("Напишем что-нибудь")
plt.title(" Линейный график")
plt.show()
```



Ну и функция, о которой мы говорили в начале ($y = 0.1x + 2.5x^2 + 0.5x^3$), выглядит так

```
plt.plot(x,0.1*x+2.5*x**2+0.5*x**3)
plt.title("Степенная функция (от -10 до 10)")
plt.show()
```

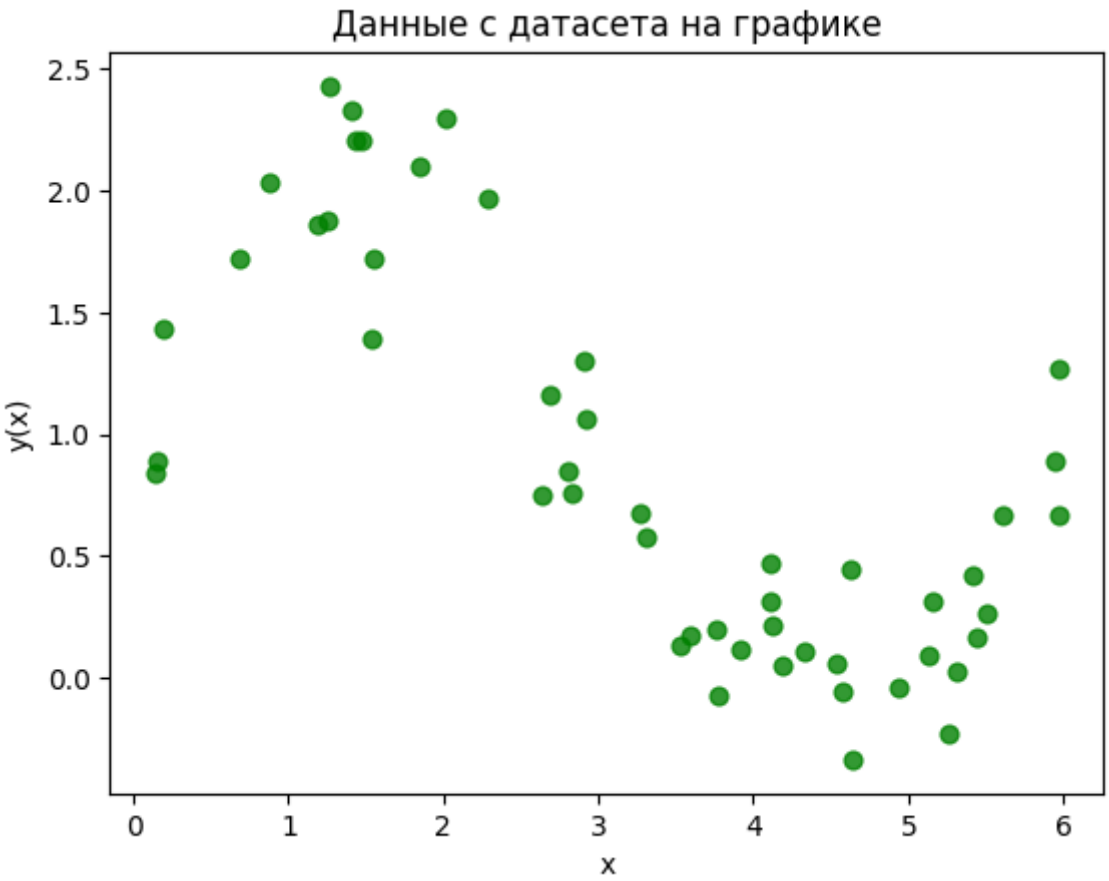


Теперь загрузим датасет с уже готовыми данными (я всё беру из курса от SkillBox, так что вопросы если что к ним), которые построил машинный алгоритм

```
data=pd.read_csv("3.10_non_linear.csv")
print(data.head())
margin=0.3
plt.scatter(data.x_train,data.y_train, 40,'g','o',alpha=0.8, label='data')
plt.title("Данные с датасета на графике")
plt.xlabel("x")
plt.ylabel("y(x)")
```

```
    x_train   y_train
0  0.138368  0.838812
1  0.157237  0.889313
2  0.188684  1.430040
3  0.685553  1.717309
4  0.874237  2.032588
```

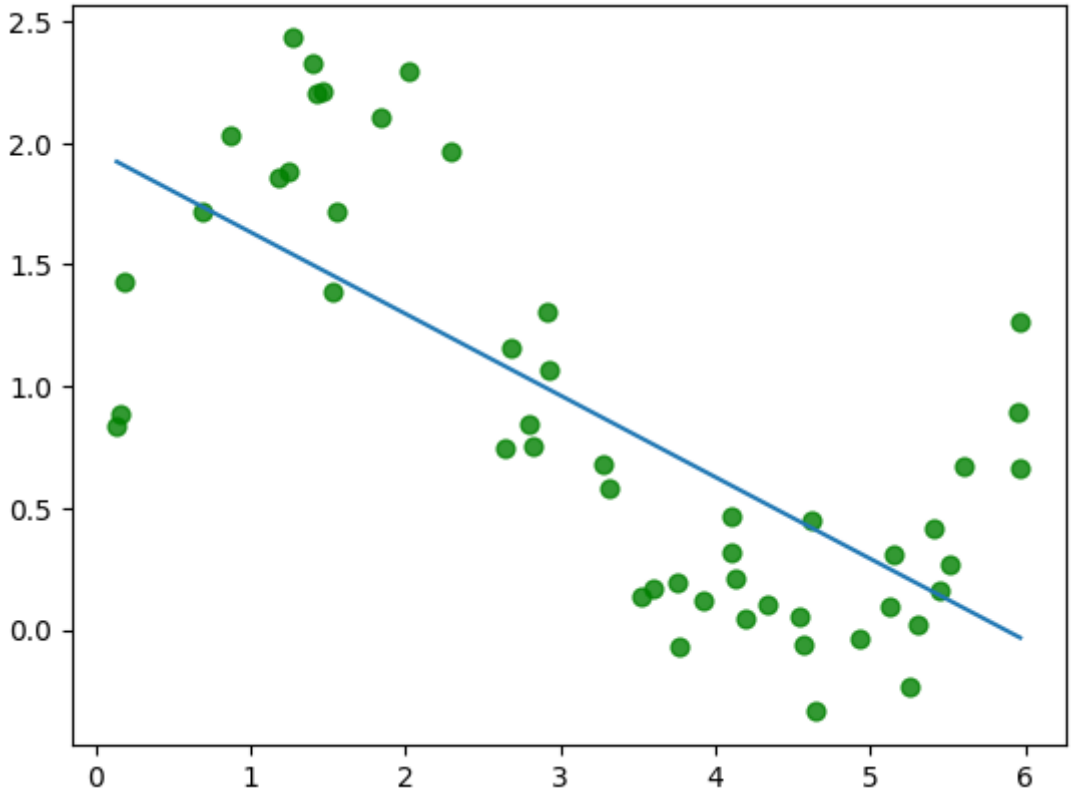
```
Text(0, 0.5, 'y(x)')
```



Теперь попробуем для начала обучить сначала линейную регрессию, а потом и полиномиальную регрессию, и сравним их потом

```
from sklearn.linear_model import LinearRegression
reg=LinearRegression().fit(data[['x_train']],data.y_train)
print(reg)
y_hat=reg.predict(data[['x_train']]) # внутрь можно поставить любые
данные, и они будут спрогнозированы, но можно и те же, которые мы и
полуили из датасета
#print(y_hat)
plt.scatter(data.x_train,data.y_train, 40,'g','o',alpha=0.8, label='data')
plt.plot(data.x_train,y_hat)
plt.show()
```

```
LinearRegression()
```



Внимание!

Дальше пойдет небольшая хитрость. Изначально нам подавались только лишь точки, то есть x и y координаты. Напишем функцию, которая будет из одномерного массива X позвращать многомерный массив из столбцов с иксами, возведенными поочередено в степень номера своего столбца. Выглядеть это будет так:

```
def generate_degrees(source_data:list, degree:int):
    return np.array([source_data**n for n in range(1, degree+1)]).T
degree=3
X_new=generate_degrees(data.x_train, degree)
X_new.shape
```

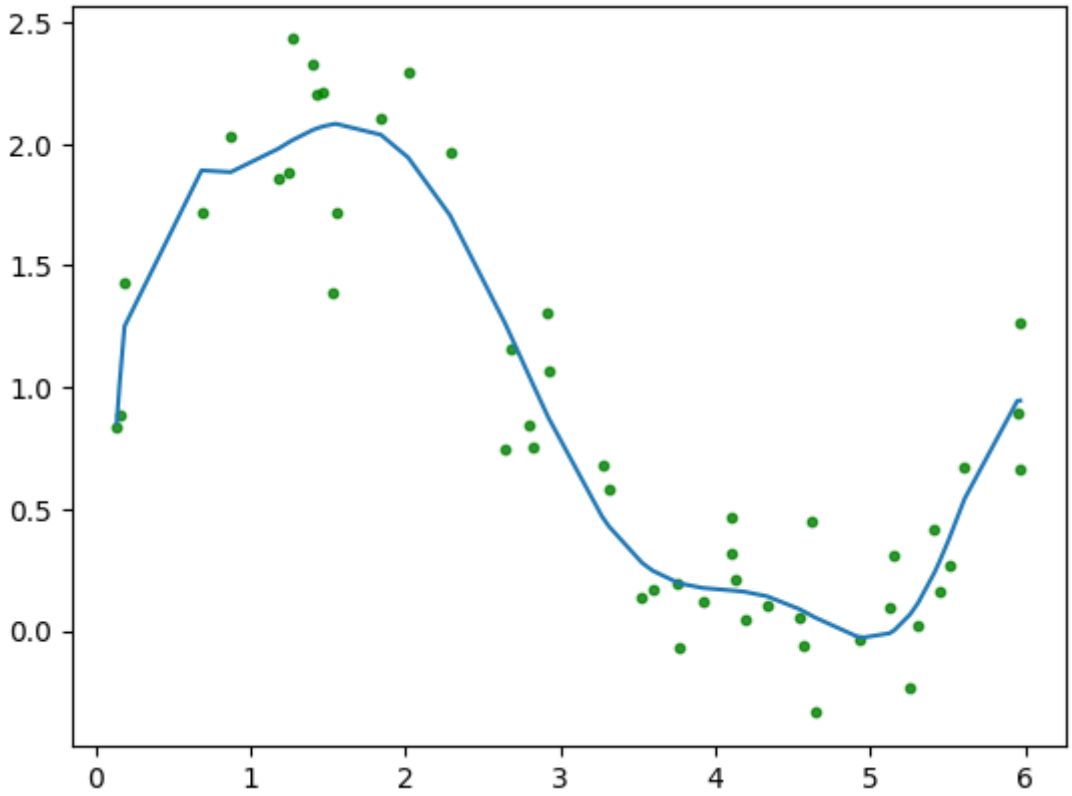
```
(50, 3)
```

Теперь напишем гениальную функцию, которая в зависимости от степень будет строить по примеру Линейной регрессии графики, основанные на таком же принципе как и обычно.
Выводить в этот раз я никаких математических формул не буду потому, что формула для множественной и полиномиальной регрессий не отличнаются! Выводы формул будут абсолютно одинаковыми, а значит, мы можем применить функцию из sklearn и на нашу новую матрицу (поэтому не будем выеживаться, самостоятельно считая эти коэффициенты $\theta $)

```
from sklearn.metrics import mean_squared_error

def train_polynomial(degree, data):
    """Генерим данные, тренируем модель

    дополнительно рисуем график
    """
    best=[]
    for i in range(1,degree+1):
        X = generate_degrees(data['x_train'], i)
        model = LinearRegression().fit(X, data['y_train'])
        y_pred = model.predict(X)
        error = mean_squared_error(data['y_train'], y_pred)
        best.append(error)
        print("Степень полинома %d Ошибка %.3f" % (i, error))

    print(f"Наилучшая степень полинома: {best.index(min(best))+1}")
    total=best.index(min(best))+1
    X = generate_degrees(data['x_train'], total)
    model = LinearRegression().fit(X, data['y_train'])
    y_pred = model.predict(X)
    error = mean_squared_error(data['y_train'], y_pred)
    print("Степень полинома %d Ошибка %.3f" % (total, error))

    plt.scatter(data['x_train'], data['y_train'], 10, 'g', 'o', alpha=0.8,
label='data')
    plt.plot(data['x_train'], y_pred)
    print()
```

```
degree=11
train_polynomial(degree, data)
```

```
Степень полинома 1 Ошибка 0.307
Степень полинома 2 Ошибка 0.298
Степень полинома 3 Ошибка 0.071
Степень полинома 4 Ошибка 0.067
Степень полинома 5 Ошибка 0.067
Степень полинома 6 Ошибка 0.064
Степень полинома 7 Ошибка 0.064
Степень полинома 8 Ошибка 0.058
Степень полинома 9 Ошибка 0.058
Степень полинома 10 Ошибка 0.058
Степень полинома 11 Ошибка 0.058
Наилучшая степень полинома: 11
Степень полинома 11 Ошибка 0.058
```



Функция выше находит среди первых i-степеней полинома наилучшую функцию, для описания поведения точек. В дальнейших уроках я, если не забуду, буду пользоваться простым правилом анилитика 80 на 20: на 80% данных строить модель, и проверять ее на оставшихся 20% данных, которые машина не видела. Так можно учиться отлаживать свою же программу быстро и корректно. На этом Всё!

Конец!

И не забудьте поставить лайк