Ishmametyev Nikolay, MAF'19

Data analysis in Python

Home assignment 3

Task 1

Task is to create a class of OLS regression *(class OLS).* It should contain attributes of linear coefficients *(.coef)* and variance-covariance matrix *(.coef_var_matr)*. Also the class should contain methods which allows to predict by the passed values of vector x*(.predict(x))* and to obtain its standard error *(.predict_dev(x)).* Also I have appended checking for appropriate size of x which has to be passed in methods of *OLS.*

```
class OLS:
    def __init__(self, X=0, y=0):
        import numpy as np
        self.X = X
        self.y = y
        self.XX = np.transpose(self.X).dot(X)
        self.coef = np.linalg.inv(self.XX).dot(np.transpose(self.X)).dot(self.y)
        n_k = 1/(self.X.shape[0]-self.X.shape[1])
        sigma = n_k*np.transpose(self.y-self.X.dot(self.coef)).dot(self.y-self.X.dot(self.coef))
        self.coef_var_matr = sigma*np.linalg.inv(self.XX)
    def predict(self,x):
        import numpy as np
        if np.shape(x)[0] == np.shape(self.X)[1]:
            return np.transpose(x).dot(self.coef)
        else:
            return 'Shape of x should be matched to number of X columns'
    def predict_dev(self,x):
        import numpy as np
        n_k = 1/(self.X.shape[0]-self.X.shape[1])
        sigma = n_k*np.transpose(self.y-self.X.dot(self.coef)).dot(self.y-self.X.dot(self.coef))
        XX_inverse = np.linalg.inv(self.XX)
        if np.shape(x)[0] == np.shape(self.X)[1]:
            return np.sqrt(sigma*(1+np.transpose(x).dot(XX_inverse).dot(x)))
        else:
            return 'Shape of x should be matched to number of X columns'
```

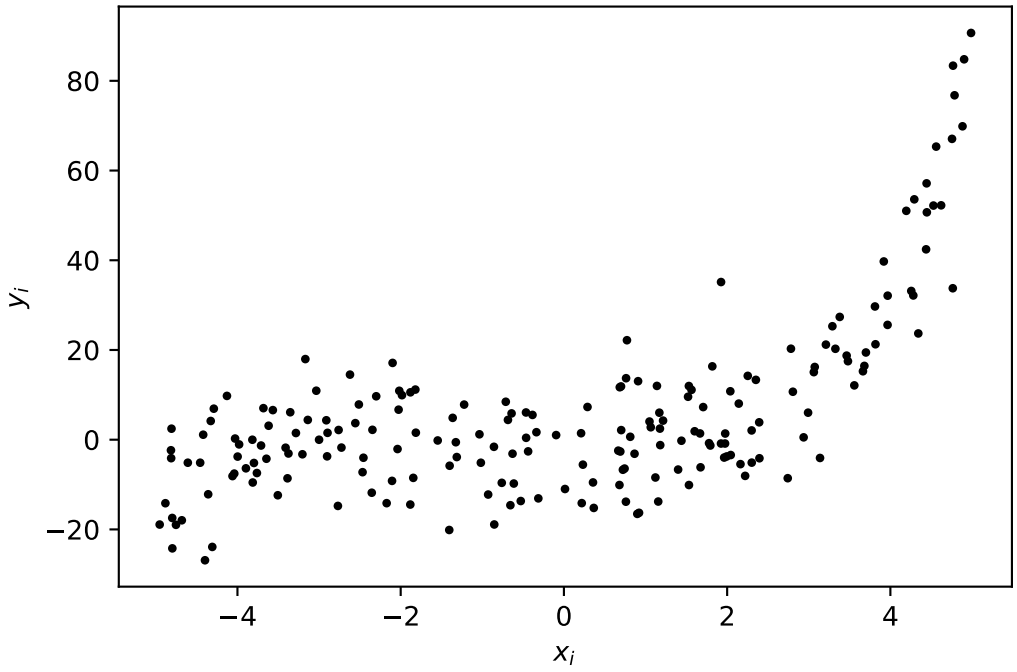
Task 2

To construct *y* I have created matrix (*x_matr)* which contains all degrees of x_i. Then we can obtain y_i via multiplication of vector β and (*x_matr).*

```
x_matr = np.ones((1,200))
for i in np.arange(1,11):
    x_matr = np.vstack((x_matr, (x ** i)/factorial(i)))
y = betas.dot(x_matr) + u
```

Ishmametyev Nikolay, MAF’19

Let's draw point cloud.

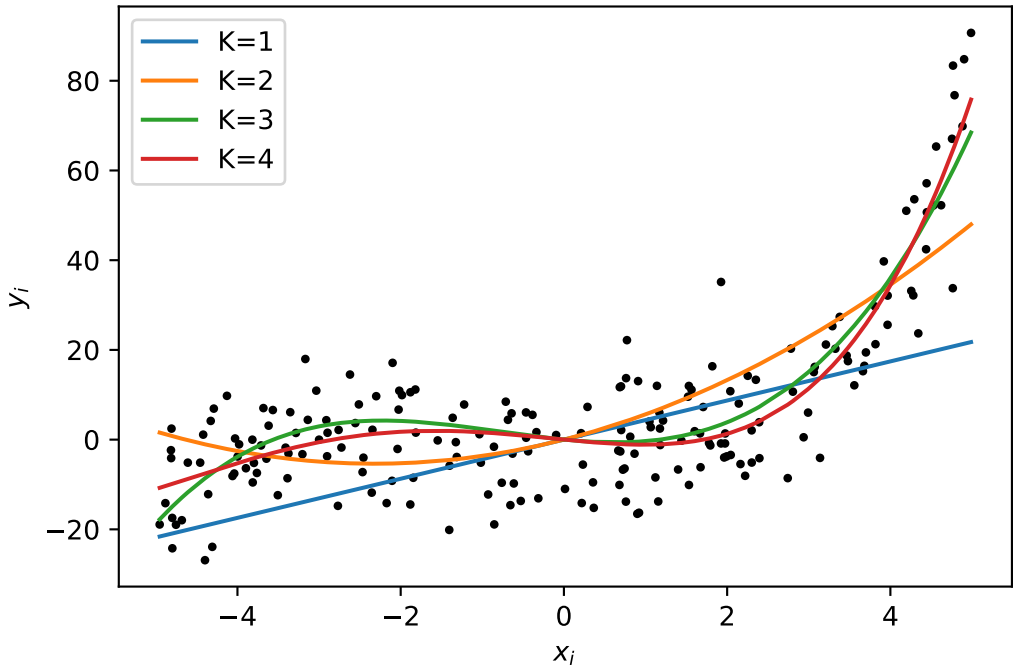



```
plt.scatter(x, y, s =5, color = 'black')
plt.xlabel("$x_i$")
plt.ylabel(‘$y_i$’)
```

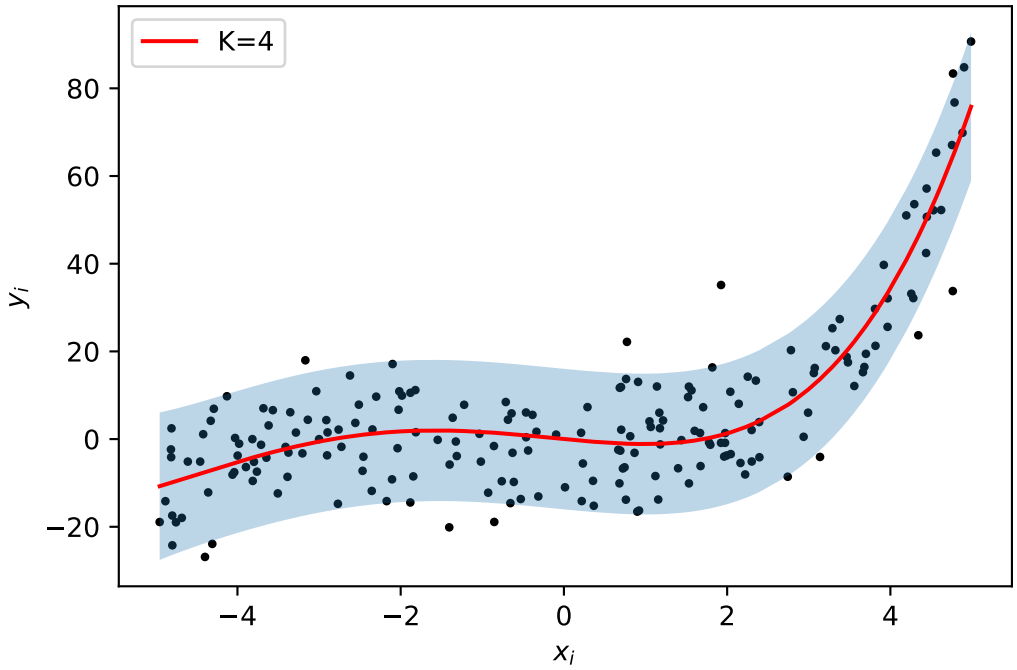
Create *y_pred* which will contain y_i for all regressions. *Order* is required to draw line without intersections. They appear because the data is not sorted.

To create all regressions I have used *np.vstack()* which is used for obtaining all data in one matrix. Usage of method *.predict(x)* of class *OLS* from the Task 1 is complexed because it predicts only one point. That’s why I have to use loop for predicting array of points. Also there is checking for size of *x.* If x is 1-d array, I add new dimension, otherwise it already has both dimensions.

```
plt.scatter(x[np.newaxis,:], y, s =5, color = 'black')
y_predicted = np.zeros((200,4))
order = np.argsort(x)
for k in np.arange(1,5):
    x_temp = x
    i = 1
    while i<k:
        x_temp = np.vstack((x_temp, x ** (i+1)))
        i+=1
    if k == 1:
        model = OLS(x_temp[:,np.newaxis],y)
        y_predicted[:,0] = model.predict(x_temp[:,np.newaxis].T)
        plt.plot(x[order],y_predicted[order,0], label= 'K=%d'%k)
    else:
        model = OLS(x_temp.T,y)
        for c in np.arange(0,200):
            y_predicted[c,k-1] = model.predict(x_temp[:,c])
        plt.plot(x[order],y_predicted[order,k-1], label= ‘K=%d'%k)
```



Let's create lists of predicted values by regression of the 4th degree *y_predicted_4* and their errors *y_predicted_4_error* using method of *OLS class (.predict_dev(x))*. Quantile of t-distribution will be calculated with methods of SciPy.stats.t.ppf().




```
                                                                    for
c in np.arange(0,200):
        y_predicted_4_error[c] = model.predict_dev(x_temp[:,c])

    p = stats.t.ppf(1-0.05, y_predicted[:,3].shape)
    plt.scatter(x[np.newaxis,:], y, s =5, color = 'black')
    plt.plot(x[order],y_predicted[order,3], label= 'K=4', color = 'r')
    plt.fill_between(x[order],y_predicted_4[order]-p*np.sqrt(y_predicted_4_error[order]),
                              y_predicted_4[order]+p*np.sqrt(y_predicted_4_error[order]), alpha=0.3)
```

Ishmametyev Nikolay, MAF'19

Task 3

Let's calculate means and deviations with built-in tools of NumPy library (methods mean and std). Information about columns and rows confidence intervals is contained in variables with addition *_col* and *_row* correspondingly. Required logic arrays and number of cases when 0 is in the confidence interval is carried out by filling the corresponding ones via logic expressions.

```
mu_col = np.mean(A, axis = 0)
sigma_col = np.std(A, axis = 0)
p = stats.t.ppf(1-0.05, A.shape[0])
size = np.sqrt(A.shape[0]-1)
logic_col = [True if (x-p*y <=0 and x+p*y/size >=0) else False for x,y in zip(mu_col,sigma_col)]
true_col = sum([1 if x == True else 0 for x in logic_col])
```

Generator of random variables is quite good. It's quite good that we are able to choose particular random statement by specifying *number* in method *seed(number)* of *np.random* otherwise python choses this number randomly for you.

Task 4

1. First of all, we need to read excel file and show first five rows and first six columns.

```
path = '/Users/nik/Documents/РЭШ/3 модуль/Data_analysis/goalies-2014-2016.csv'
df = pd.read_csv(path, sep=';',header = 0)
df.iloc[0:5,0:6]
```

2. Let's calculate '*save_percentage*' via pandas tools by the division of according columns from DataFrame. Result of comparison will be settled in list *'logic'*. If there is *False* in '*logic*' then difference will be calculated in '*dev*' list, otherwise there will be zero.

```
save_percentage = df['saves']/df['shots_against']
logic = [True if round(x,3) == round(y,3) else False for x,y in
                                                    zip(save_percentage,df['save_percentage'])]
dev = [0 if b else np.abs(x-y) for b,x,y in zip(logic,save_percentage,df['save_percentage'])]
max(dev)
```

3. Means and standard deviations of columns '*games_played*', '*goals_against*', *'save_percentage*' could be calculated via in-built functions of pandas.

```
df[ ['games_played', 'goals_against', 'save_percentage'] ].mean(axis = 0)
df[ ['games_played', 'goals_against', 'save_percentage'] ].std(axis = 0)
```

4. Firstly, we should select number of players in a whole list with '*games_played*' > 40. Then we choose the *'player'* and its 'save_percentage' from this shortened list player with the highest *'save_percentage*'.

This is number of row:

```
df ['save_percentage']
                [np.array(df['games_played']>40) & np.array(df['season']=='2016-17')].idxmax()
```

These are required columns:

[‘player’,’save_percentage']

5. Let’s find out number of different seasons via *np.unique()*. Then we will looking for player with the biggest number of '*saves*' and particular value of ‘*season*’ via *DataFrame* method *.idxmax()* which return index of first occurrence of maximum over *df[‘saves’]*. Then we will place this data in temp and eventually in variable '*info*'.

```
season = np.unique(df['season'])
info = []
for s in season:
    player = df[‘player'] [df['saves'][df['season']==s].idxmax()]
    saves = df[‘saves'] [df['saves'][df['season']==s].idxmax()]
    temp = (s,player,saves)
    info.append(temp)
```

6. In this task we have to check two conditions. For this purpose we should transform our arrays into *np.array* which is able to work with boolean variables. Results are supposed to be a list, that's why we use *np.hstack()* to match information in matrix for corresponding seasons. To add information for different seasons we use *np.vstack()* and then we have to delete the first row with zeros by *np.delete()*.
To find all appropriate keeps I have counted how much each name is in the array *keeps_clean,* and then to add name in a variable *names.*

Example of filling appropriate players

```
for s in season:
    player = np.array(df [‘player'] [ np.array(df['wins']>30) & np.array(df[‘season']==s) ] )
```