xsesta07 (2)

December 11, 2022

1 MSP Projekt

Autor: Pavel Šesták

1.1 1. task

1.1.1 Import modules

[1]:
```
from scipy.stats import binom_test
from scipy.stats import chi2
import pandas as pd
import numpy as np
import scipy.stats as stats
import scipy.stats as st
import scipy.spatial as sp
```

1.1.2 Data and constants

[2]:
```
DATA_LEN = 8
#                       Praha, Brno, Znojmo, Tišnov, Paseky, Horní Lomná, Dolní␣
 ↪Věstonice, okolí studenta
numberOfRespondents = [1327,  915,  681,    587,    284,    176,         215,  ␣
 ↪          32          ]
winterTime          = [510,   324,  302,    257,    147,    66,          87,   ␣
 ↪           5          ]
summerTime          = [352,   284,  185,    178,    87,     58,          65,   ␣
 ↪          12          ]
changingTime        = [257,   178,  124,    78,     44,     33,          31,   ␣
 ↪           7          ]
OpinionLess         = [208,   129,  70,     74,     6,      19,          32,   ␣
 ↪           8          ]

DF = 4 - 1 #number of groups - 1
ALPHA = .05

BIG_CITIES = 0
SMALL_CITIES = 1
```

```
VILLAGES = 2
STUDENT = 3

NUMBER_OF_RESPONDENTS = "numberOfRespondents"
WINTER = "winter"
SUMMER = "summer"
CHANGE = "change"
OPINION_LESS = "opinionLess"
CATEGORIES = [WINTER, SUMMER, CHANGE, OPINION_LESS]
CATEGORIES_WITH_RESPONDETS = [WINTER, SUMMER, CHANGE, OPINION_LESS,␣
 ↪NUMBER_OF_RESPONDENTS]

indexes = [None] * 4
indexes[BIG_CITIES] = [0, 1]
indexes[SMALL_CITIES] = [2, 3]
indexes[VILLAGES] = [4,5,6]
indexes[STUDENT] = [7]
```

1.1.3 Dataset functions

[3]:
```
def initData(dataSet, cats):
    """
    This function initialize data

    :param dataSet: array with 4 elements
    :param cats: array of categories to initialize, different for dataset and␣
 ↪expected
    :return: initialized dataset for this task
    """
    for i in range(4):
        dataSet[i] = {}
        for cat in cats:
            dataSet[i][cat] = []

def fillData(dataSet, indexes):
    """
    This function fill dataset and aggregate to four categories

    :pre initData
    :param dataSet: initialized dataset
    :param indexes: array of indexes for each categories to aggregate data
    :return: aggregated data in dataset
    """
    for i in range(4):
        for j in indexes[i]:
            dataSet[i][NUMBER_OF_RESPONDENTS].append(numberOfRespondents[j])
            dataSet[i][WINTER].append(winterTime[j])
```

```
            dataSet[i][SUMMER].append(summerTime[j])
            dataSet[i][CHANGE].append(changingTime[j])
            dataSet[i][OPINION_LESS].append(OpinionLess[j])
            if(numberOfRespondents[j] != (winterTime[j] + summerTime[j] +␣
↪changingTime[j] + OpinionLess[j])):
                print("ERROR>>>"+" corrupted data for index: ", i)
```

1.1.4 Initialize dataset

[4]:
```
dataSet = [None] * 4
initData(dataSet, CATEGORIES_WITH_RESPONDETS)
fillData(dataSet, indexes)
```

1.1.5 Calculate X2 for all categories in one test

https://sites.ualberta.ca/~lkgray/uploads/7/3/6/2/7362679/slides_-_binomialproportionaltests.pdf

Calculate totals

[5]:
```
totals = {}
totals[NUMBER_OF_RESPONDENTS] = sum(numberOfRespondents)
totals[WINTER] = sum(winterTime)
totals[SUMMER] = sum(summerTime)
totals[CHANGE] = sum(changingTime)
totals[OPINION_LESS] = sum(OpinionLess)
```

Calculate averages

[6]:
```
averages = {}
for cat in CATEGORIES:
    averages[cat] = totals[cat]/totals[NUMBER_OF_RESPONDENTS]
```

Expected functions

[7]:
```
def fillExpected(expectedDataset, dataSet):
    """
    This function calculate expected matrix

    :pre initData for both params
    :param expectedDataset: initialized expected matrix
    :param dataSet: filled dataset
    :return: filled expected matrix
    """
    for i in range(4):
        for cat in CATEGORIES:
            expectedDataset[i][cat] = sum(dataSet[i][NUMBER_OF_RESPONDENTS]) *␣
↪averages[cat]
```

Calculate x2 and check hypothesis

[8]:
```
expected = [None] * 4
initData(expected, CATEGORIES)
fillExpected(expected, dataSet)

#calc x2
x2 = 0

for i in range(4):
    for k in CATEGORIES:
        x2 += ((sum(dataSet[i][k]) - expected[i][k]) ** 2) / expected[i][k]

print("x2: ", x2)

critical = chi2.ppf(1-ALPHA, df=DF)

print("critical: ", critical)

if x2 > critical:
    print("H0 rejected")
else:
    print("H0 NOT rejected")
```

```
x2:  51.18185323789891
critical:  7.814727903251179
H0 rejected
```

We reject the hypothesis of equality of all categories, now we can test the hypothesis for individual pairs. We define function to calculate x2, then we will call it for each category

[9]:
```
def calculateX2(cat):
    """
    This function calculate x2

    :pre calculate dataset and expected
    :param cat: defines which category we will test
    :return: result of x2
    """
    data_ = []
    expected_ = []
    for i in range(4):
        data_.append(np.sum(dataSet[i][cat]))
        expected_.append(expected[i][cat])

    print("dataset: ",data_)
    print("expected: ", expected_)
    x2Result = st.chisquare(data_, expected_, ddof=DF)
    print("x2 test result:", x2Result)
```

```
    if x2Result.pvalue < ALPHA:
        print("H0 rejected")
    else:
        print("H0 NOT rejected")
```

Task 1 a)

[10]:
```
calculateX2(WINTER)
```

```
dataset:  [834, 559, 300, 5]
expected:  [902.754564856533, 510.56770215793216, 271.7927436566279,
12.884989328906805]
x2 test result: Power_divergenceResult(statistic=17.58332339926807, pvalue=nan)
H0 NOT rejected
```

Task 1 b)

[11]:
```
calculateX2(SUMMER)
```

```
dataset:  [636, 363, 210, 12]
expected:  [649.153900877401, 367.13967275314207, 195.4410718520275,
9.265354517429452]
x2 test result: Power_divergenceResult(statistic=2.2048732412486354, pvalue=nan)
H0 NOT rejected
```

Task 1 c)

[12]:
```
calculateX2(CHANGE)
```

```
dataset:  [435, 202, 108, 7]
expected:  [399.80649751007826, 226.11714488973203, 120.36993123073275,
5.70642636945696]
x2 test result: Power_divergenceResult(statistic=7.234679869193312, pvalue=nan)
H0 NOT rejected
```

Task 1 d)

[13]:
```
winterVector = np.empty(shape=3)
for i in range(3):
    winterVector[i] = (np.sum(dataSet[i][WINTER]))

print("winterVector:", winterVector)

winterSum = 0
respondentSum = 0
#winterPst = np.empty(shape=3, dtype=float)
for i in range(3):
    winterSum += np.sum(dataSet[i][WINTER])
    respondentSum += np.sum(dataSet[i][NUMBER_OF_RESPONDENTS])
```

```
winterPst = winterSum / respondentSum
print("winterPst:", winterPst)

expectedWinter = np.empty(shape=3, dtype=float)

for i in range(3):
    expectedWinter[i] = np.sum(dataSet[i][NUMBER_OF_RESPONDENTS]) * winterPst

print("expectedWinter:", expectedWinter)

x2Result = st.chisquare(winterVector, expectedWinter,ddof=DF)
print(x2Result)

if x2Result.pvalue < ALPHA:
    print("H0 rejected")
else:
    print("H0 NOT rejected")
```

```
winterVector: [834. 559. 300.]
winterPst: 0.4045400238948626
expectedWinter: [906.97873357 512.9567503  273.06451613]
Power_divergenceResult(statistic=12.661948651569508, pvalue=nan)
H0 NOT rejected
```

Task 1 e)

[14]:
```
opinionLessVector = np.empty(shape=3)
for i in range(3):
    opinionLessVector[i] = (np.sum(dataSet[i][OPINION_LESS]))

print("opinionLessVector:", opinionLessVector)

opinionLessSum = 0
respondentSum = 0
#winterPst = np.empty(shape=3, dtype=float)
for i in range(3):
    opinionLessSum += np.sum(dataSet[i][OPINION_LESS])
    respondentSum += np.sum(dataSet[i][NUMBER_OF_RESPONDENTS])

opinionLessPst = opinionLessSum / respondentSum
print("winterPst:", opinionLessPst)

expectedWinter = np.empty(shape=3, dtype=float)

for i in range(3):
    expectedWinter[i] = np.sum(dataSet[i][NUMBER_OF_RESPONDENTS]) *␣
 ↪opinionLessPst
```

```
print("expectedWinter:", expectedWinter)

x2Result = st.chisquare(opinionLessVector, expectedWinter, ddof=DF)
print(x2Result)

if x2Result.pvalue < ALPHA:
    print("H0 rejected")
else:
    print("H0 NOT rejected")
```

```
opinionLessVector: [337. 144.  57.]
winterPst: 0.12855436081242533
expectedWinter: [288.21887694 163.00692951  86.77419355]
Power_divergenceResult(statistic=20.688664757394136, pvalue=nan)
H0 NOT rejected
```

Task 1 f) https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.jensenshannon.html

[15]:
```
def getPstVec(entity):
    pstVec = np.empty(shape=4, dtype=float)
    index = 0
    for cat in CATEGORIES:
        pstVec[index] = np.sum(dataSet[entity][cat]) / np.
 ↪sum(dataSet[entity][NUMBER_OF_RESPONDENTS])
        index += 1
    return pstVec

psts = np.eye(4,4)
for i in range(4):
    psts[i] = getPstVec(i)

print("psts")
print(psts)

deltas = []
for i in range(3):
    deltas.append(psts[STUDENT] - psts[i])

print("Deltas")
print(deltas)

squareErrorVector = []
for i in range(3):
    squareErrorVector.append(np.sum(np.abs(deltas[i])))

print("Square errors")
print("BigCities\t\tSmallCities\t\tVillages")
```

```
print(squareErrorVector)

jensenshannonDistanceVector = []
for i in range(3):
    jensenshannonDistanceVector.append(sp.distance.jensenshannon(psts[STUDENT],␣
 ↪psts[i]))

print("Jensen-Shannon divergence")
print("BigCities\t\tSmallCities\t\tVillages")
print(jensenshannonDistanceVector)
```

```
psts
[[0.3719893  0.28367529 0.19402319 0.15031222]
 [0.44085174 0.2862776  0.15930599 0.11356467]
 [0.44444444 0.31111111 0.16       0.08444444]
 [0.15625    0.375      0.21875    0.25      ]]
Deltas
[array([-0.2157393 ,  0.09132471,  0.02472681,  0.09968778]),
array([-0.28460174,  0.0887224 ,  0.05944401,  0.13643533]), array([-0.28819444,
0.06388889,  0.05875   ,  0.16555556])]
Square errors
BigCities               SmallCities             Villages
[0.43147859054415705, 0.5692034700315457, 0.5763888888888888]
Jensen-Shannon divergence
BigCities               SmallCities             Villages
[0.18028641401227227, 0.2319063474640471, 0.2474451229320161]
```

The least squares error is in large cities. The Jensen-Shannon divergence is also smallest for large cities. This corresponds to where the data was collected. The data was collected in Brno and the questionnaire was filled out by my family and friends.

1.2 2. task

1.2.1 Import modules

- Pandas - Pandas is used to store data in data frames
- Statsmodels - This library is used for linear regression
- Numpy - Math library for basic calculations

[16]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
```

1.2.2 Define dataset

Z index data is defined in MSP_Projekt_2022-23_Zadani_Ct-10.xls with index 31.

[17]:
```

```

```
data = {
    'x': [ 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 2.22, 2.22, 2.22, 2.22, 2.
 ↪22, 2.22, 2.22, 4.44, 4.44, 4.44, 4.44, 4.44, 4.44, 4.44, 6.67, 6.67, 6.67,␣
 ↪6.67, 6.67, 6.67, 6.67, 8.89, 8.89, 8.89, 8.89, 8.89, 8.89, 8.89, 11.11, 11.
 ↪11, 11.11, 11.11, 11.11, 11.11, 11.11, 13.33, 13.33, 13.33, 13.33, 13.33, 13.
 ↪33, 13.33, 15.56, 15.56, 15.56, 15.56, 15.56, 15.56, 15.56, 17.78, 17.78, 17.
 ↪78, 17.78, 17.78, 17.78, 17.78, 20.00, 20.00, 20.00, 20.00, 20.00, 20.00, 20.
 ↪00],
    'y': [ 0.00, 1.67, 3.33, 5.00, 6.67, 8.33, 10.00, 0.00, 1.67, 3.33, 5.00, 6.
 ↪67, 8.33, 10.00, 0.00, 1.67, 3.33, 5.00, 6.67, 8.33, 10.00, 0.00, 1.67, 3.
 ↪33, 5.00, 6.67, 8.33, 10.00, 0.00, 1.67, 3.33, 5.00, 6.67, 8.33, 10.00, 0.
 ↪00, 1.67, 3.33, 5.00, 6.67, 8.33, 10.00, 0.00, 1.67, 3.33, 5.00, 6.67, 8.33,␣
 ↪10.00, 0.00, 1.67, 3.33, 5.00, 6.67, 8.33, 10.00, 0.00, 1.67, 3.33, 5.00, 6.
 ↪67, 8.33, 10.00, 0.00, 1.67, 3.33, 5.00, 6.67, 8.33, 10.00 ],
    'z': [-35.61, 78.11, -39.84, 64.09, 117.58, 55.45, 40.28, -20.67, 17.31, 79.
 ↪04, -213.48, -144.45, -94.36, -193.02, -66.38, 161.15, -59.7, -66.83, 14.17,␣
 ↪-207.41, -179.98, 264.75, 172.09, 51.32, 67.66, -20.13, -144.37, -174.38,␣
 ↪442.82, 340.21, 201.94, -34.12, -5.78, 31.38, -261.55, 457.5, 618.18, 335.
 ↪88, 218.06, 76.66, 100.49, -101.64, 737.57, 568.3, 473.15, 370.03, 322.74,␣
 ↪179.21, -5.03, 1030.95, 851.61, 726.03, 573.27, 398.04, 364.79, 313.34, 1430.
 ↪86, 1112.19, 1052.76, 701.14, 605.7, 458.74, 326.59, 1772, 1473.25, 1358.47,␣
 ↪1137.05, 920.43, 688.66, 498.32],
}

df = pd.DataFrame(data)
```

1.2.3 Model fitting

In this part of code we will try to find model which best fit our data. We iterate over formulas and store the best model with the highest R2.
Our base model is: $Z = \beta_1 + \beta_2 X + \beta_3 Y + \beta_4 X^2 \beta_5 Y^2 + \beta_6 XY$

[18]:
```
formulas = [
        (df.x, df.y, df.x**2, df.y**2, df.x*df.y),
        (df.x, df.y, df.x**2, df.y**2),
        (df.x, df.y, df.x**2),
        (df.x, df.y),
        (df.x),
        (),
]
index = 0
bestFormula = None
bestModel = None
bestR2 = 0.0

for formula in formulas:
        if index < 4:
```

```
                f = np.column_stack(formula)
                f = sm.add_constant(f)
        elif index == 4:
                f = formula
                f = sm.add_constant(f)
        else:
                f = sm.add_constant(df[['x','y']])

        model = sm.OLS(df.z, f).fit()

        if bestR2 < model.rsquared:
                bestFormula = formula
                bestModel = model
                bestR2 = model.rsquared
        index += 1
```

1.2.4 Analyze summary statistics of the best found model

Our coefficient of determination is 0.975. This is very good result, maybe is our model overfitted. Method how to check is model is overfitted is to split dataset to two sets. One set for training and one for evaluation and check results. In second datatable we analyze p-values of t-test which verify H0 coefficient is equal to zero which implies its doesnt add any information to our model. t-test is comapred to aplha confidence level 0.05. We can remove from our model constant, x1, x2 and x4 without loss accurance of our model.

[19]:
```
print("Best model summary")
print(bestModel.summary())
tt = bestModel.t_test(np.eye(len(bestModel.params)))
```

```
Best model summary
                            OLS Regression Results
==============================================================================
Dep. Variable:                      z   R-squared:                       0.975
Model:                            OLS   Adj. R-squared:                  0.973
Method:                 Least Squares   F-statistic:                     491.3
Date:                Sun, 11 Dec 2022   Prob (F-statistic):           1.24e-49
Time:                        13:29:04   Log-Likelihood:                -399.09
No. Observations:                  70   AIC:                             810.2
Df Residuals:                      64   BIC:                             823.7
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.4333     36.246      0.067      0.947     -69.976      74.842
x1             4.2885      5.655      0.758      0.451      -7.008      15.585
x2            -2.6971     10.677     -0.253      0.801     -24.026      18.632
x3             4.0409      0.252     16.023      0.000       3.537       4.545
```

```
x4             0.3091      0.941      0.329      0.744      -1.570       2.189
x5            -5.8401      0.425    -13.729      0.000      -6.690      -4.990
==============================================================================
Omnibus:                        0.111   Durbin-Watson:                   1.971
Prob(Omnibus):                  0.946   Jarque-Bera (JB):                0.014
Skew:                           0.031   Prob(JB):                        0.993
Kurtosis:                       2.970   Cond. No.                         839.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

1.2.5 Reduce best model and remove unnecesary coefficiets

In this section we will verify if our small model trully didnt loss accurance. Our coefficient of determination is 0.983 which is better result then the model with all coefficients and we can finish our analyzation with this model (Without constant). We need constant in our model, because without constant factor we cannot compute r-squared. t_test show p values are equal to zero, this coefficients are most signifficant for our model.

Our final model is $Z = \beta_1 + \beta_3 X^2 + \beta_5 XY$

[20]:
```
smallModelF = np.column_stack((df.x**2, df.x*df.y))

smallModelF = sm.add_constant(smallModelF)
smallModel = sm.OLS(df.z, smallModelF).fit()

print("Small model summary")
print(smallModel.summary())
```

```
Small model summary
                            OLS Regression Results
==============================================================================
Dep. Variable:                      z   R-squared:                       0.974
Model:                            OLS   Adj. R-squared:                  0.974
Method:                 Least Squares   F-statistic:                     1271.
Date:                Sun, 11 Dec 2022   Prob (F-statistic):           5.23e-54
Time:                        13:29:04   Log-Likelihood:                -399.47
No. Observations:                  70   AIC:                             804.9
Df Residuals:                      67   BIC:                             811.7
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         13.8804     13.522      1.027      0.308     -13.109      40.870
x1             4.2243      0.084     50.166      0.000       4.056       4.392
x2            -5.7743      0.220    -26.274      0.000      -6.213      -5.336
```

```
==============================================================================
Omnibus:                        0.360   Durbin-Watson:                   1.943
Prob(Omnibus):                  0.835   Jarque-Bera (JB):                0.067
Skew:                           0.053   Prob(JB):                        0.967
Kurtosis:                       3.108   Cond. No.                         307.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

1.2.6 Estimation of regression parameters

Estimations of regression parameters by the method least squares and their 95% confidence intervals which is calculated in model summary.
$\beta_1 \in < -13.109; 40.870 >$
$\beta_3 \in < 4.102; 4.413 >$
$\beta_5 \in < -6.131; -5.290 >$

1.2.7 Estimation of dispersion

Our final model is $Z = \beta_1 + \beta_3 X^2 + \beta_5 XY$
For estimation of dispersion we use modified equation from lecture MSP about linear regresion
$s^2 = \frac{1}{n-2}(\sum_i^n z_i^2 - \beta_3 \sum_i^n x_i^2 z_i - \beta_5 \sum_i^n x_i y_i z_i)$

[21]:
```
s2 = smallModel.ssr/(len(df.z)-3)
print("Estimation of dispersion: ", s2)

xiyi = np.column_stack((np.power(df.x,2), df.x*df.y))
xiyi = sm.add_constant(xiyi)
z_predict = smallModel.predict(xiyi)

print("Dispersion from model: ", np.var(df.z - z_predict))
```

```
Estimation of dispersion:  5538.279525128777
Dispersion from model:  5300.9246883375445
```

1.2.8 Test hypothesis $\beta_3 = \beta_5 = 0$

p-value is lower than alpha, we reject hypothese

[22]:
```
smallModel.f_test("x1 = x2 = 0").summary()
```

[22]:
```
'<F test: F=1271.2900920794098, p=5.228646164190812e-54, df_denom=67, df_num=2>'
```

Zero equality f test is rejected ### Test hypothesis $\beta_3 = \beta_5$ t test and f test both reject hypotheses about equality of coefficients

[23]:
```
print(smallModel.t_test("x1 = x2").summary())
print(smallModel.f_test("x1 = x2").summary())
```

```
                             Test for Constraints
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
c0             9.9987      0.279     35.853      0.000       9.442      10.555
==============================================================================
<F test: F=1285.469081250755, p=1.8998855078551433e-45, df_denom=67, df_num=1>
```

Coefficient equality f test and t test is rejected