MSP 2. projekt

Alexej Beňuš

xbenus01@stud.fit.vutbr.cz

1. část

Zadání:

1. Zapište zvolenou parametrizaci Weibullova rozdělení, logaritmickou-věrohodnostní funkci pro zadaná data a její parciální derivace podle parametrů (shape, scale).
2. Pomocí scipy.optimize nalezněte maximálně věrohodné odhady parametrů weibullova rozdělení.
3. Pomocí věrohodnostního poměru otestujte hypotézu, že exponenciální rozdělení je postačujícím modelem zapsaných dat (Parametr tvaru = 1)
4. Podle výsledku ze 3) použijte výsledné rozdělení pravděpodobnosti (s maximálně věrohodnými odhady jako parametry) a nalezněte bodové odhady pro střední dobu zaměstnání v oboru a 10% percentil zaměstnání v oboru (za jakou dobu odejde do jiného odboru 10 % absolventů).
5. [dobrovolná část] zkuste nějak slovně charakterizovat/popsat fungování doby zaměstnání v oboru jako náhodné veličiny, dle Vašich výsledků a parametrů

Jako první samozřejmě import potřebných knihoven, balíčků, modulů...

```
import numpy as np
import pandas as pd
from scipy.optimize import minimize
from scipy.stats import chi2
from scipy.special import gamma
```

Načtení dat z datasetu a převedení na numpy pole.

```
# Load data
data = pd.read_excel('Data_2024.xlsx', sheet_name='Data_věrohodnost')

time = data['doba práce v oboru [roky]'].to_numpy()
censored = data['censored'] .to_numpy()
```

1. Definice log-věrohodnostní funkce pro Weibullovo rozdělení, přijímá parametry beta a eta. Poté také funkce pro parciální derivace podle jednotlivých parametrů.

```
# Log-likelihood function for Weibull distribution
def logLikelihoodWeibull(params, data, censored):
    beta, eta = params
    uncensored = data[censored == 0]
    censored_data = data[censored == 1]

    term1 = len(uncensored) * np.log(beta)
    term2 = len(uncensored) * beta * np.log(eta)
```

```
    term3 = (beta - 1) * np.sum(np.log(uncensored))
    term4 = np.sum((uncensored / eta)**beta)

    term5 = np.sum((censored_data / eta)**beta)

    return -(term1 - term2 + term3 - term4 - term5)

# partial derivative of the log-likelihood function with respect to
beta
def partial_derivative_beta(beta, eta, data, censored):
    uncensored = data[censored == 0]
    censored_data = data[censored == 1]

    term1 = len(uncensored) / beta
    term2 = -len(uncensored) * np.log(eta)
    term3 = np.sum(np.log(uncensored))
    term4 = -np.sum((uncensored / eta)**beta * np.log(uncensored /
eta))
    term5 = -np.sum((censored_data / eta)**beta * np.log(censored_data
/ eta))

    return term1 + term2 + term3 + term4 + term5

# partial derivative of the log-likelihood function with respect to
eta
def partial_derivative_eta(beta, eta, data, censored):
    uncensored = data[censored == 0]
    censored_data = data[censored == 1]

    term1 = -len(uncensored) * beta / eta
    term2 = beta * np.sum((uncensored / eta)**beta / eta)
    term3 = beta * np.sum((censored_data / eta)**beta / eta)

    return term1 + term2 + term3
```

1. Nastavíme Betu a Etu na 1, poté zavoláme funkci minimize, která minimalizuje log-věrohodnostní funkci Weibullova rozdělení.

```
initial_guess = [1.0, 1.0]

# Obtaining the maximum likelihood estimates of the parameters beta
and eta
result = minimize(logLikelihoodWeibull, initial_guess, args=(time,
censored))
beta, eta = result.x

print(f"Beta(shape): ", beta.round(2))
print(f"Eta(scale): ", eta.round(2))

Beta(shape):  6.17
Eta(scale):  7.43
```

1. Nyní získáme maximálně věrohodný odhad, s beta = 1 pro exponenciální rozdělení. Poté spočítáme log-věrohodnost pro obě situace - Weibullovo rozdělení i exponenciální rozdělení a uděláme LRT.

```
# Obtaining MLE for exponential distribution (parameter beta is equal
to 1)
resultBetaOne = minimize(logLikelihoodWeibull, initial_guess,
args=(time, censored), bounds=[(1.0,1.0),(None,None)])

# Calculating log-likelihood for both situations
resWB = -result.fun
resWBBetaOne = -resultBetaOne.fun
LRT_statistic = 2 * (resWB - resWBBetaOne)

# Calculating the critical value (for 1 degree of freedom and
significance level 0.05)
value = chi2.ppf(0.95, df=1)

print("Test statistic:", round(LRT_statistic, 2))
print("Critical value:", value.round(2))
if LRT_statistic > value:
    print("The hypothesis of exponential distribution is rejected.")
else:
    print("The hypothesis of exponential distribution is not
rejected.")
```

```
Test statistic: 592.39
Critical value: 3.84
The hypothesis of exponential distribution is rejected.
```

1. Nyní spočítáme střední dobu zaměstnání v oboru, poté vypočítáme 10. percentil.

```
# (E[T])
mean_time = eta * gamma(1 + 1 / beta)

# Calculating 10th percentile
percentile_10 = eta * (-np.log(0.9))**(1 / beta)

print(f"Average time of employment in the field: {mean_time:.2f}
years")
print(f"10% percentile (10% of graduates leave by this time):
{percentile_10:.2f} years")
```

```
Average time of employment in the field: 6.90 years
10% percentile (10% of graduates leave by this time): 5.16 years
```

2. část

Zadání:

1. Pomocí zpětné eliminace určete vhodný regresní model. Za výchozí „plný“ model považujte plný kvadratický model (všechny interakce druhého řádu a všechny druhé mocniny, které dávají smysl).
   - Zapište rovnici Vašeho finálního modelu.
   - Diskutujte splnění předpokladů lineární regrese a základní regresní diagnostiky.
   - Pokud (až během regresního modelování) identifikujete některé „extrémně odlehlé hodnoty“ můžete ty „nejodlehlejší“ hodnoty, po alespoň krátkém zdůvodnění, vyřadit.
2. Pomocí Vašeho výsledného modelu identifikujte, pro které nastavení parametrů má odezva nejproblematičtější (největší) hodnotu (použijte model, nikoli samotná pozorování).
3. Odhadněte hodnotu odezvy uživatele s Windows, při průměrném nastavení ostatních parametrů a vypočtěte konfidenční interval a predikční interval pro toto nastavení.
4. Na základě jakýchkoli vypočtených charakteristik argumentujte, zdali je Váš model „vhodný“ pro další použití.

Jako první samozřejmě import potřebných knihoven, balíčků, modulů...

```
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt

import statsmodels.formula.api as smf
from statsmodels.stats.outliers_influence import
variance_inflation_factor
from pandas.api.types import is_string_dtype
from scipy.stats import shapiro
```

Nyní načteme data z dodaného excelu.

Z dat jde na první pohled vidět několik problematických částí.

První je použití kategorické proměnné, tu změníme na numerickou jinak by se požadovaná statistika dělala problematicky. Také můžeme použít drop_first, čímž dosáhneme menší redundance, jelikož počet kategorií 'n' jde vyjádřit i pomocí 'n-1' hodnot. Poté převedeme tyto hodnoty na numerické z boolu, aby mohly být používány v dalších knihovnách.

Také si jde všimnout, že 2 hodnoty jsou na sobě závislé - ScrollingPct a InteractingPct - jejich součet se rovná 1, jelikož je to klasifikace toho co zrovna dělá uživatel, jelikož jsou pouze dvě možnosti - projíždí síť nebo interaguje. Toto lze využít tím, že budeme dále používat pouze jednu proměnnou - zvolil jsem ScrollingPct.

```
# Load the data
data: pd.DataFrame = pd.read_excel("Data_2024.xlsx",
sheet_name='Data_regrese')

# One-hot encode the OSType column, as it is a categorical variable
# Also drop one column, as it is redundant - if all other columns are
```

```
0, then the dropped column is 1
data = pd.get_dummies(data, columns=['OSType'], drop_first=True)
data.rename(columns= {'Ping [ms]': 'Ping'}, inplace=True) # for some
reason, statsmodels doesn't like the brackets in the column name

# Convert the dummy columns to integer
for(column) in data.columns:
    if data[column].dtype == bool:
        data[column] = data[column].astype(int)
```

Zde definujeme funkci, která nám bude dávat výsledný model.

```
def getResults(data, equation):
    # Standard least squares model
    model=smf.ols(formula=equation, data=data)
    results=model.fit()
    return results, model
```

Nyní musíme definovat kvadratickou funkci pro model. Z úlohy vyplynulo, že chceme vyjádřit ping.

Tato funkce obsahuje všechny kombinace a druhé mocniny všech proměnných.

Před samotnou eliminací ještě kontrola multikolinearity za pomocí VIF.

```
# Full quadratic model
equation = "Ping~ ActiveUsers + ScrollingPct + OSType_MacOS +
OSType_Windows + OSType_iOS + I(ActiveUsers**2)\
+ ActiveUsers:ScrollingPct + ActiveUsers:OSType_MacOS +
ActiveUsers:OSType_Windows + ActiveUsers:OSType_iOS\
+ I(ScrollingPct**2) + ScrollingPct:OSType_MacOS +
ScrollingPct:OSType_Windows + ScrollingPct:OSType_iOS"

results, model = getResults(data, equation)

# Calculate the VIF for each variable
X = pd.DataFrame(model.exog, columns=model.exog_names)
vif = pd.Series([variance_inflation_factor(X.values, i) for i in
range(X.shape[1])], index=X.columns)
print(vif)
```

```
Intercept                      91.854488
ActiveUsers                    32.087036
ScrollingPct                   25.577021
OSType_MacOS                   15.217934
OSType_Windows                 16.447197
OSType_iOS                     14.840501
I(ActiveUsers ** 2)            22.499134
ActiveUsers:ScrollingPct        8.705628
ActiveUsers:OSType_MacOS       10.454073
```

```
ActiveUsers:OSType_Windows     10.189613
ActiveUsers:OSType_iOS          9.061267
I(ScrollingPct ** 2)           16.422293
ScrollingPct:OSType_MacOS       6.489601
ScrollingPct:OSType_Windows     7.671741
ScrollingPct:OSType_iOS         7.692924
dtype: float64
```

Můžeme vidět, že hodně VIF hodnot je > 10. Problematické jsou očividně ActiveUsers a InteractingPct.

Proto standardizujeme tyto hodnoty pomocí Z-score a spočítáme vše znovu.

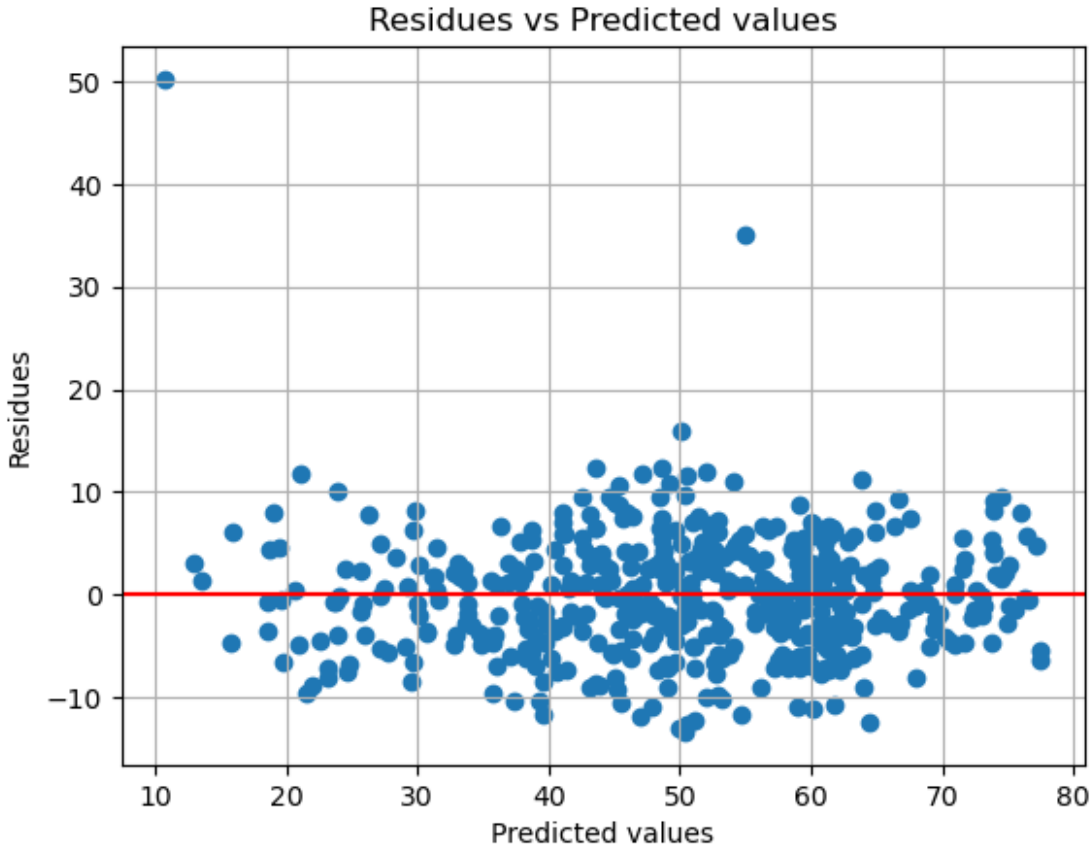
```
# Standardize the data
data2 = data.copy()
data2AUMean = data2['ActiveUsers'].mean()
data2AUStd = data2['ActiveUsers'].std()
data2SPStd = data2['ScrollingPct'].std()
data2SPMean = data2['ScrollingPct'].mean()

data2['ActiveUsers']=(data2['ActiveUsers']-data2AUMean)/data2AUStd
data2['ScrollingPct']=(data2['ScrollingPct']-data2SPMean)/data2SPStd

# Compute again
results, model = getResults(data2, equation)

X = pd.DataFrame(model.exog, columns=model.exog_names)
vif = pd.Series([variance_inflation_factor(X.values, i) for i in
range(X.shape[1])], index=X.columns)
print(vif)
```

```
Intercept                      6.781856
ActiveUsers                    4.918200
ScrollingPct                   4.737489
OSType_MacOS                   1.654677
OSType_Windows                 1.630480
OSType_iOS                     1.606613
I(ActiveUsers ** 2)            1.019635
ActiveUsers:ScrollingPct       1.041103
ActiveUsers:OSType_MacOS       2.335540
ActiveUsers:OSType_Windows     2.383411
ActiveUsers:OSType_iOS         2.247663
I(ScrollingPct ** 2)           1.032984
ScrollingPct:OSType_MacOS      2.503341
ScrollingPct:OSType_Windows    2.092440
ScrollingPct:OSType_iOS        2.198356
dtype: float64
```

Nyní jsou všechny VIF hodnoty < 5 a pouze konstanta je > 5, což pokud chápu správně ničemu nevadí. Takže nyní budeme pokračovat v eliminaci.

V eliminaci jsem došel k následné formuli. Postupně jsem odstraňoval jednotlivé proměnné podle hodnoty P.

```
eliminationFormula = "Ping ~ ActiveUsers + ScrollingPct + OSType_MacOS
+ OSType_Windows + OSType_iOS + I(ActiveUsers**2)\
+ ActiveUsers:ScrollingPct + ActiveUsers:OSType_MacOS +
ActiveUsers:OSType_Windows + ActiveUsers:OSType_iOS"

results, model = getResults(data2, eliminationFormula)


plt.scatter(results.fittedvalues, results.resid)
plt.axhline(y=0, color='r', linestyle='-')
plt.grid(True)
plt.xlabel('Predicted values')
plt.ylabel('Residues')
plt.title('Residues vs Predicted values')
plt.show()
```

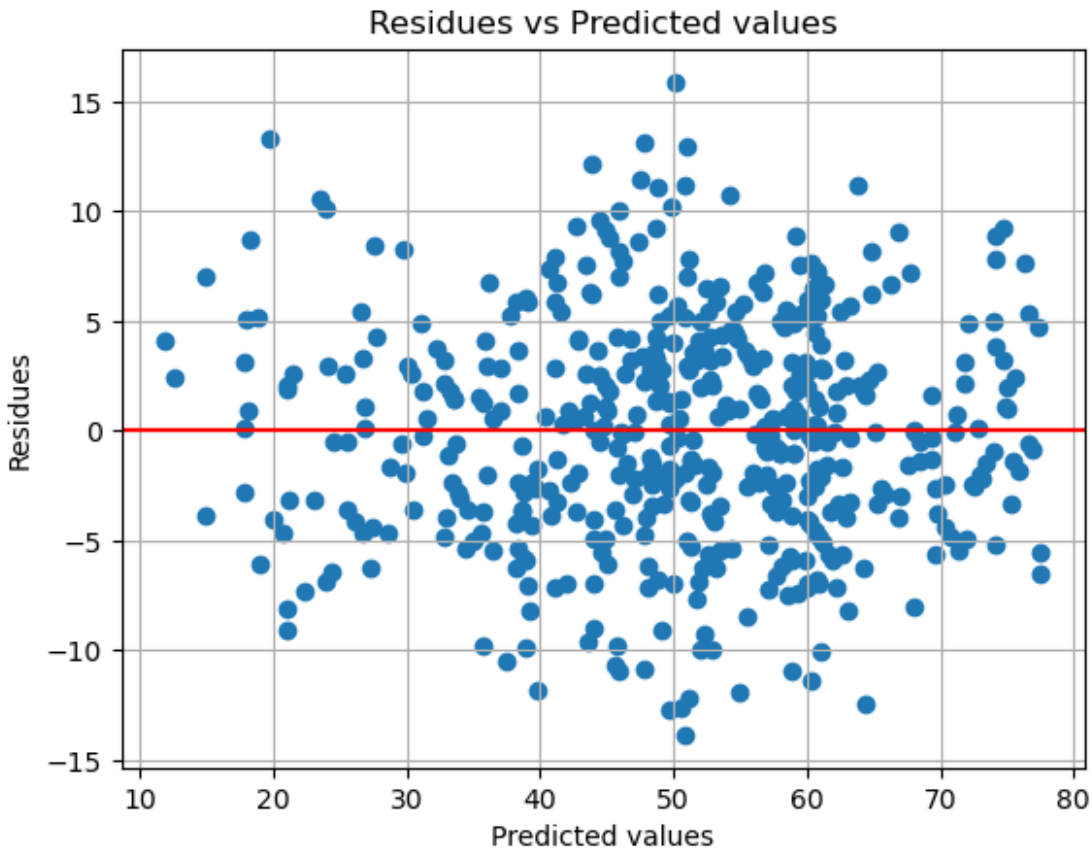


Z grafu jde jasně vidět, že má 2 extrémní hodnoty. Tyto hodnoty ačkoliv nevypadají nijak nevalidní odstraním aby se zlepšila hodnota R-Squared.

K odstranění jsem použil Studentized residuals. Ten používá samotné residua, páky a standard error k nalezení "outliers".

```
# Check for outliers
# Studentized residuals > 3 are considered outliers
influence = results.get_influence()
studentized_residuals = influence.resid_studentized_internal
outliers = abs(studentized_residuals) > 3  # Threshold: 3 standard
deviations
print("Outliers:\n", data2[outliers])

# Remove outliers
data2_cleaned = data2[~outliers]
results_cleaned, model_cleaned = getResults(data2_cleaned,
eliminationFormula)

plt.scatter(results_cleaned.fittedvalues, results_cleaned.resid)
plt.axhline(y=0, color='r', linestyle='-')
plt.grid(True)
plt.xlabel('Predicted values')
plt.ylabel('Residues')
plt.title('Residues vs Predicted values')
plt.show()
```

```
Outliers:
      ActiveUsers  InteractingPct  ScrollingPct  Ping  OSType_MacOS  \
255     0.010659          0.4912     -0.008739    90             0
476    -2.092179          0.2111      0.937545    61             1

     OSType_Windows  OSType_iOS
255               1           0
476               0           0
```



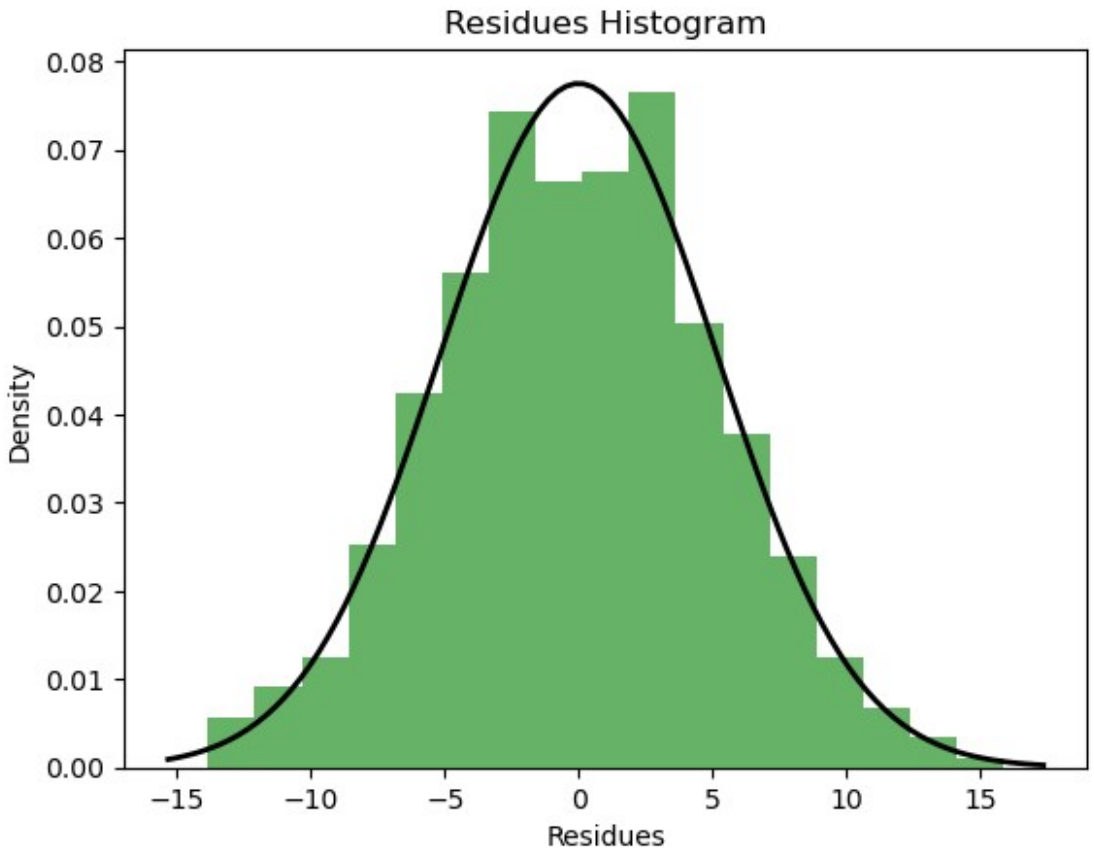
Finální verze modelu je po odstranění extrémních hodnot a eliminování proměnných následující: "Ping ~ ActiveUsers + ScrollingPct + OSType_MacOS + OSType_Windows + OSType_iOS + I(ActiveUsers**2) + ActiveUsers:ScrollingPct + ActiveUsers:OSType_MacOS + ActiveUsers:OSType_Windows + ActiveUsers:OSType_iOS"

```
print(results_cleaned.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Ping   R-squared:
0.877
Model:                            OLS   Adj. R-squared:
0.875
Method:                 Least Squares   F-statistic:
349.9
Date:                Sun, 15 Dec 2024   Prob (F-statistic):
1.28e-215
Time:                        17:22:51   Log-Likelihood:
-1528.7
No. Observations:                 500   AIC:
3079.
```

```
Df Residuals:                      489   BIC:
3126.
Df Model:                           10

Covariance Type:            nonrobust

==============================================================================
=========================
                                 coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
-------------------------
Intercept                     51.2936      0.547     93.743      0.000
50.219      52.369
ActiveUsers                   10.0457      0.515     19.522      0.000
9.035      11.057
ScrollingPct                  -5.1410      0.234    -21.928      0.000
-5.602      -4.680
OSType_MacOS                   9.0073      0.667     13.501      0.000
7.696      10.318
OSType_Windows                 3.6657      0.671      5.463      0.000
2.347       4.984
OSType_iOS                    -5.7223      0.692     -8.267      0.000
-7.082      -4.362
I(ActiveUsers ** 2)           -3.0081      0.254    -11.832      0.000
-3.508      -2.509
ActiveUsers:ScrollingPct       2.5596      0.238     10.752      0.000
2.092       3.027
ActiveUsers:OSType_MacOS       4.4373      0.692      6.415      0.000
3.078       5.796
ActiveUsers:OSType_Windows    -1.9127      0.677     -2.827      0.005
-3.242      -0.583
ActiveUsers:OSType_iOS        -2.7423      0.700     -3.915      0.000
-4.119      -1.366
==============================================================================
========
Omnibus:                        0.661   Durbin-Watson:
1.990
Prob(Omnibus):                  0.719   Jarque-Bera (JB):
0.750
Skew:                           0.014   Prob(JB):
0.687
Kurtosis:                       2.812   Cond. No.
8.01
==============================================================================
========

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

```
# Control normality of residuals - graph and Shapiro test
# Graph
plt.hist(results_cleaned.resid, bins='auto', density=True, alpha=0.6,
color='g')
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = stats.norm.pdf(x, np.mean(results_cleaned.resid),
np.std(results_cleaned.resid))
plt.plot(x, p, 'k', linewidth=2)
plt.title("Residues Histogram")
plt.xlabel("Residues")
plt.ylabel("Density")
plt.show()

# Shapiro
from scipy.stats import shapiro
if shapiro(results_cleaned.resid)[1] > 0.01: # 99% spolehlivost
    print("Data suggets normal distribution")
else:
    print("Data are not normally distributed")

# Cook-Weisberg test, significance level 5%
# H0: Homoskedasticity
# H1: Heteroskedasticity
from statsmodels.stats.diagnostic import het_breuschpagan
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-
value"]
test = het_breuschpagan(results_cleaned.resid,
results_cleaned.model.exog)
res = list(zip(name, test))
if (res[1][1] < 0.05):
    print("H0 rejected, data are heteroskedastic")
else:
    print("H0 not rejected, data can be homoskedastic")


# Test for mean of residuals = 0, alt. hypothesis != 0, use T-test
with significance level 5%
if stats.ttest_1samp(results_cleaned.resid, 0).pvalue < 0.05:
    print("Rejecting null hypothesis, mean of residuals != 0")
else:
    print("Not rejecting null hypothesis, mean of residuals can be 0")
```




```
Data suggets normal distribution
H0 not rejected, data can be homoskedastic
Not rejecting null hypothesis, mean of residuals can be 0
```

Diskuze předpokladů a diagnostika Pro matici plánu zjevně platí, že , protože je 500 (No. observations v modelu), zatímco m je v podstatě počet členů ve formuli, který je mnohem nižší (viz 2.1.1 Formule modelu).

Residua mají normální rozdělení - ověřeno pomocí Shapiro testu a lze vidět i z grafu.

Durbin-Watson test (lze vidět v summary) vychází pouze 1,990 a to značí že autokorelace je nízká.

Data neobsahují heteroskedasticitu.

Také jsme nezamítli hypotézu, že střední hodnota residuí se nerovná 0.

Podstatné regresní předpoklady jsou splněny a proto je diagnostika v pořádku.

2.2 V tomto úkolu mám identifikovat, pro které nastavení parametrů má odezva největší hodnotu.

```
# 2.2
# Predict Ping for all observations
```

```
predicted_vals = results_cleaned.predict(data2_cleaned)

# Find the index of the most extreme predicted value (maximum)
max_ind = np.argmax(predicted_vals)  # Index of maximum predicted Ping
most_problematic = data2_cleaned.iloc[max_ind]

# Find the index of the minimum predicted value (optional)
min_ind = np.argmin(predicted_vals)  # Index of minimum predicted Ping
(if needed)
least_problematic = data2_cleaned.iloc[min_ind]

# De-standardize variables for interpretability (if standardized)
most_problematic_original = most_problematic.copy()
most_problematic_original['ActiveUsers'] = (
    most_problematic['ActiveUsers'] * data2AUStd + data2AUMean
)
most_problematic_original['ScrollingPct'] = (
    most_problematic['ScrollingPct'] * data2SPStd + data2SPMean
)

# Display results for the most problematic point
print("\nMost problematic point based on predicted Ping:")
print(most_problematic_original)
print("\nPredicted Ping for this point:",
predicted_vals.iloc[max_ind])


Most problematic point based on predicted Ping:
ActiveUsers       9657.000
InteractingPct       0.973
ScrollingPct         0.027
Ping                72.000
OSType_MacOS         1.000
OSType_Windows       0.000
OSType_iOS           0.000
Name: 10, dtype: float64

Predicted Ping for this point: 77.50462838943064
```

Z výsledků vychází, že nejproblematičtější hodnoty jsou v moment kdy je aktivních skoro 10 000 uživatelů a skoro všichni tito uživatelé (9369 uživatelů) interagují se stránkou, což je celkem logický závěr - zátěž serveru je větší při interakci, než při projíždění stránky. Tento ping vychází na 77.5ms, což je podle mě přijatelná hodnota pro stránky, ikdyž děláme konkurenci twitteru a facebooku... Nekonkurujeme Riot games ani jinému vývojáři her, u kterých je důležitá latence, ale děláme sociální síť.

2.3

V tomto úkolu mám odhadnout odezvu uživatele s windows, při průměrném nastavení ostatních parametrů. Pro tento odhad zase používám standardizované hodnoty - proto nemusím zadávat

data2["ActiveUsers"].mean() a data2["ScrollingPct"].mean(), jelikož tyto hodnoty vrátí 0 (reálně vrátí 1.123456e-16 - zjednodušeno na 0...). Jako operační systém je specifikovaný windows.

```
# 2.3
# Create a new data point for a Windows user with average settings
new_data = pd.DataFrame({
    'ActiveUsers': [0],  # Standardized mean is 0
    'ScrollingPct': [0],  # Standardized mean is 0
    'OSType_MacOS': [0],
    'OSType_Windows': [1],  # User is on Windows
    'OSType_iOS': [0],
})
print("\nNew data point for a Windows user with average settings:")
print(new_data)

# Predict Ping with confidence and prediction intervals
prediction = results_cleaned.get_prediction(new_data)
summary_frame = prediction.summary_frame(alpha=0.05)  # 95% confidence
level

# Extract intervals
predicted_ping = summary_frame['mean'].iloc[0]
confidence_interval = summary_frame[['mean_ci_lower',
'mean_ci_upper']].iloc[0]
prediction_interval = summary_frame[['obs_ci_lower',
'obs_ci_upper']].iloc[0]

# Print results
print(f"Predicted Ping: {predicted_ping:.2f}")
print(f"95% Confidence Interval:
({confidence_interval['mean_ci_lower']:.2f},
{confidence_interval['mean_ci_upper']:.2f})")
print(f"95% Prediction Interval:
({prediction_interval['obs_ci_lower']:.2f},
{prediction_interval['obs_ci_upper']:.2f})")


New data point for a Windows user with average settings:
   ActiveUsers  ScrollingPct  OSType_MacOS  OSType_Windows  OSType_iOS
0            0             0             0               1           0
Predicted Ping: 54.96
95% Confidence Interval: (53.93, 55.99)
95% Prediction Interval: (44.68, 65.24)
```

2.4 Na základě jakýchkoli vypočtených charakteristik argumentujte, zdali je Váš model „vhodný“ pro další použití.

R-2 hodnota je 0.877. Jsou splněny (snad) všechny podmínky regrese. Tímto argumentuji, že je vhodný na další použití.

Kdyby bylo za úkol okomentovat proč není vhodný, tak je to asi jednodušší... Hodně ve zkratce - málo dat - mít o jeden spíše dva řády (klidně i více...) více, tak by model byl mnohem lepší a přesnější. Také by se mohla z části dat udělat validační sada na nějakou verifikaci modelu.