Intro

Project Description

The Project's objective is to scrape products from Amazon using the **Robot Framework**. It involves data processing and analysis, as well as implementing **machine learning** techniques to predict prices based on review counts and rating stars. A user interface has been provided.

This manual consists of six slides:

1. **Intro** (current): Manual Structure, Project Files Description
2. **User Interface:** Showcasing initial user input and the product dashboard (the end product)
3. **Sequential Explanation of the Code Execution Steps:** shows the functions used and provides pseudo code, serving as a general guide to understanding the project's execution process.
4. **Robotic Part:** Further explanation of the robotic part of the project and the associated code.
5. **Machine Learning Part:** Further explanation of the machine learning part of the project and the relevant code.
6. **Full System Representation:** Provides a detailed understanding of the entire project process and its interdependencies, should you desire a comprehensive view.

The Project Manual abstracts the code, focusing on the relevant parts to aid comprehension. However, the full code can be found in the respective files. The goal is to present the information in an easy-to-understand manner. **PS:** Zooming might be required.

The project folder contains the following files:

- **README_Project_Manual.pdf** (current)

- **front_interface.py:** main file that integrates all the components of the project, including the execution of machine learning algorithms
- **amzn_product_scraping.txt:** robot script responsible for scraping products from Amazon
- **amzn_product_scraping_modified.txt:** modified version of the robot script, explained in detail in the project manual
    - **udf_robot.py:** A user-defined function required for time randomization
    - **output.xml:** The log file generated by executing the robot script
- **amzn_data_transformation.py:** handles data processing and transformation
- **streamlit_subprocess_amzn_scraping_script.sh:** shell script that sequentially executes the modified robot script and the python data transformation script
- **product_table.csv:** A CSV file containing the scraped product data

Product Scraping

Which product do you want to scrape? Example: headphones

usb c cable

Scrape Data

Product Dashboard

Searched Product

usb c cable

Products Scraped

466

Average Price	Most Expensive	Cheapest
13.24€	98.49€	2.92€

Product Table

Update Data

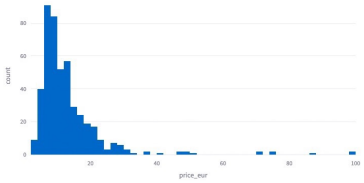
	product_id	title	price_eur	↓ review_count	rating_stars
24	product_25	AINOPE USB C Kabel, 2 S	8.05	88,296	4.6
311	product_312	Amazon Basics USB Typ	6.7	86,563	4.6
6	product_7	Amazon Basics USB Typ	13.26	86,563	4.6
71	product_72	Anker Premium 180 cm l	10.07	82,896	4.7
53	product_54	Amazon Basics Verbindu	7.25	37,810	4.6
0	product_1	Amazon Basics Verbindu	7.25	37,810	4.6
20	product_21	INIU USB C Kabel, Ladek	10.07	31,006	4.6
416	product_417	USB C Kabel RAMPOW, iF	5.09	29,939	4.6
26	product_27	USB C Kabel RAMPOW, iF	6.99	29,939	4.6
21	product_22	USB C Kabel RAMPOW, iF	5.09	29,939	4.6

Trends in Product Data

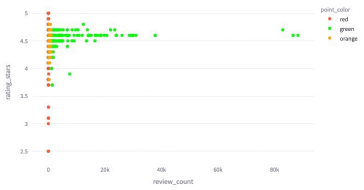
- The majority of products in the data have a high review count, indicating that they are popular among customers.
- The products have a high average rating of 4.6 stars, suggesting that they are well-received by customers.
- There is a wide range of price points, with some products priced as low as 4.63 EUR and others as high as 13.26 EUR, providing options for different budget ranges.

Ask AI anything about your scraped products (first 30 rows)

Price Distribution



Correlation Scatter Plot



Machine Learning Performace

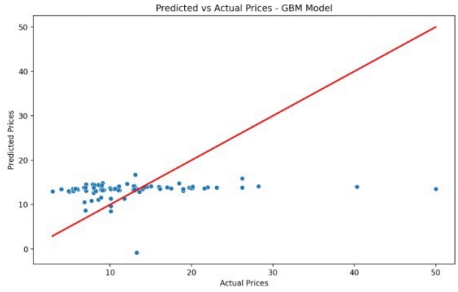
Model: Linear Regression

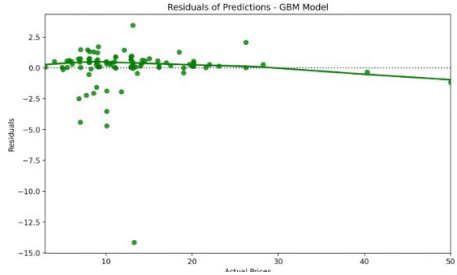
Mean Squared Error

51.6017

R-Squared

-0.003





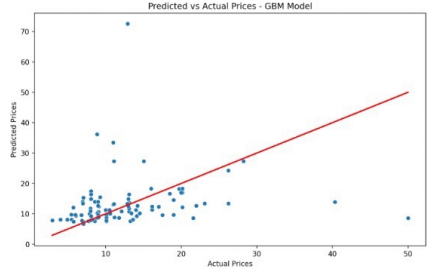
Model: Gradient Boosting

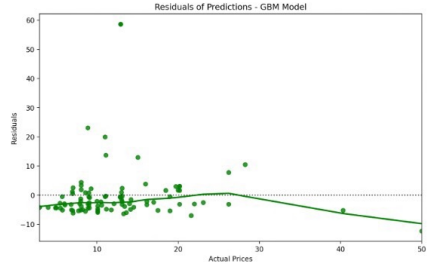
Mean Squared Error

137.404

R-Squared

-1.6709





```python
def welcome():...

def scrape_products(search_term):...

def product_dashboard(search_term):...

def perform_machine_learning(df):...

def perform_linear_regression(X_train, X_test, y_train, y_test):...

def perform_gradient_boosting(X_train, X_test, y_train, y_test):...

def ask_ai_for_trends(product_data_string):...

def ask_ai_anything(product_data_string, user_prompt):...
```

execution inside of product_dashboard fct.

Pseudo code of the project execution process

```python
# 1. User types in a product that he wants to search
search_term = st.text_input('Which product do you want to scrape? Example: headphones')

# 2. product is being scraped from Amazon (with the help of Robot Framework)
scrape_products(search_term)                Further explanation below (Robotic Part)

# 3. after scraping is completed, dashboard is being created with the given searched product
product_dashboard(search_term)

# 4. machine learning is then executed inside of the product_dashboard
perform_machine_learning(df)

    # specific algorithms being applied    Further explanation below (Machine Learning Part)
    perform_linear_regression()
    perform_gradient_boosting()

# additional functionality and features (inside the product dashboard)

    # AI Section
    ask_ai_for_trends() # Generative Pretrained Transformer returns some trends in the product set
    ask_ai_anything()   # user AI prompt input related to the scraped product set
                        # example: list the companies of the products listed

    # Price Distribution Histogram
    fig = px.histogram(df, x='price_eur')

    # Correlation Scatter Plot
    fig2 = px.scatter(df, x='review_count', y='rating_stars', color='point_color', color_discrete_map=color_scale)

    # Sidebar with DataFrame Filtering Options
    st.sidebar.header('Filters')
    min_price = st.sidebar.number_input('Min Price', min_value=0, key=1, on_change=filter_data)
    max_price = st.sidebar.number_input('Max Price', min_value=0, key=2, on_change=filter_data)
    min_review_count = st.sidebar.number_input('Min Review Count', min_value=0, key=3)
    min_stars = st.sidebar.number_input('Min Stars', min_value=1, max_value=5)
    max_stars = st.sidebar.number_input('Max Stars', min_value=1, max_value=5)
```

Robotic Part

When welcome() function is executed, this is what happens:

```python
st.title('Product Scraping')
search_term = st.text_input('Which product do you want to scrape? Example: headphones')
st.session_state.search_term = search_term

if st.button("Scrape Data"):
    if len(search_term) != 0:
        st.write(f'Started scraping data for {search_term}...')
        scrape_products(search_term)  # running the shell script, robot framework, and python data processing script
        st.write('Scraping Completed ✅ ')

        time.sleep(2)

        # creating dashboard
        product_dashboard(search_term)
    else:
        st.write('Please enter a search term.')
```

```python
def scrape_products(search_term):
    # opening the robot script to replace <<<replace_string>>> with the actual searched product
    robot_script = open('amzn_product_scraping.txt', 'r').read()
    robot_script = robot_script.replace('<<<replace_string>>>', search_term)

    # writing modified script to a temporary file
    with open('amzn_product_scraping_modified.txt', 'w') as file:
        file.write(robot_script)

    # running the script which executes the modified robot script and python script
    subprocess.run("./streamlit_subprocess_amzn_scraping_script.sh", shell=True)
```

1. robot amzn_product_scraping_modified.txt
2. python amzn_data_transformation.py

```
# first it runs the modified robot script with the actual searched product which scrapes product data from Amazon

# when scraping is completed it executes the python script which grabs the scraped product data from the output.xml log

# in the log file it finds title, price, review count and star rating information for each product
# then data processing is performed, saved into a dictionary which is transformed to a pandas dataframe

# finally the dataframe is saved as a CSV file named product_table.csv
# only after this step the final dashboard is created and machine learning on the dataset is performed and results shown
```

Robot Script - amzn_product_scraping.txt

```
*** Settings ***
Documentation       Scrape prices from Amazon
Library             SeleniumLibrary
Test Setup          Pause Inbetween
Test Teardown       Pause Inbetween
Library             udf_robot.py        # user defined function containing randomization of time between page flips

*** Variables ***
${SEARCH_TERM}      <<<replace_string>>>        #string which will be replaced with 'search_term' of the user input
${URL}              https://www.amazon.de/s?k=${SEARCH_TERM}
${BROWSER}          Firefox
${NUM_PAGES}        5           # number of pages to scrape

*** Keywords ***
Pause Inbetween
    ${PAUSE_LENGTH}=        Random Pause Length
    Sleep       ${PAUSE_LENGTH}     #Pauses between Page Flips to mimique human bahivour
    Log         Pause length: ${PAUSE_LENGTH}

Go To Page
    [Arguments]         ${page_number}
    Go To               ${URL}&page=${page_number}
    Log                 Page flipped successfully

*** Test Cases ***

Perform Scraping
    Open Browser    ${URL}      ${BROWSER}
    Maximize Browser Window
    Pause Inbetween

    FOR    ${page_num}    IN RANGE    2    ${NUM_PAGES}                # iterate through amazon pages
        Pause Inbetween-
        Wait Until Element Is Visible  xpath=//div[starts-with(@cel_widget_id, 'MAIN-SEARCH_RESULTS')]  #ELEMENT VISIBLE # this element is a container of each individual product listing

        @{element_list} =  Get WebElements  xpath=//div[starts-with(@cel_widget_id, 'MAIN-SEARCH_RESULTS')]  # each product container is identified by a @cel_widget_id starting with 'MAIN-SEARCH-RESULTS
        FOR  ${element}  IN  @{element_list}

            ${html}=    Get Element Attribute    ${element}    outerHTML                                      # we save each found product container as an element into our element_list

            Log    ${html}     # every box container containing product listing information is logged to output.xml which is later handed over to python script to perform data processing

        END

        Go To Page    ${page_num}

    END

    [Teardown]      Close Browser
```

```python
def random_pause_length():
    import random
    """Creates a random number in the given interval.
    This way the pauses between the flipping of Amazon pages are randomized to simulate human behavior."""
    return str(random.uniform(2.0324, 3.4824))
```

→ amzn_product_scraping_modified.txt

```
*** Variables ***
${SEARCH_TERM}      usb c cable
${URL}              https://www.amazon.de/s?k=${SEARCH_TERM}
```

Machine Learning Part

```
# Goal is to predict the price based on the
review count and rating stars of a product
```

```python
        st.title('Product Scraping')
        search_term = st.text_input('Which product do you want to scrape? Example: headphones')
        st.session_state.search_term = search_term

        if st.button("Scrape Data"):
            if len(search_term) != 0:
                st.write(f'Started scraping data for {search_term}...')
                scrape_products(search_term)  # running the shell script, robot framework, and python data processing script
                st.write('Scraping Completed ✅ ')

                time.sleep(2)

                # creating dashboard
                product_dashboard(search_term)
            else:
                st.write('Please enter a search term.')
```

```python
def product_dashboard(search_term):

    def load_data():
        data = pd.read_csv('product_table.csv')
        return data

    df = load_data()

    # _____________MACHINE LEARNING _________
    perform_machine_learning(df)
```

—> Code inbetween is hidden for demonstration purposes (focus on Machine Learning)

Packages for ML

```python
# modules for machine learning
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```python
def perform_machine_learning(df):
    """Goal is to predict the price based on the review count and rating stars of a product"""
    st.header('Machine Learning Performance')

    # subset dataframe for machine learning
    df_for_ml = df[['price_eur', 'review_count', 'rating_stars']]

    # feature and target variable selection
    X = df_for_ml[['review_count', 'rating_stars']]
    y = df_for_ml['price_eur']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Linear Regression

```python
# LINEAR REGRESSION
st.subheader('Model: Linear Regression')
mse_linreg, r2_linreg, y_test, y_pred_linreg = perform_linear_regression(X_train, X_test, y_train, y_test)

# show model performance metrics
st.metric('Mean Squared Error', round(mse_linreg, 4))
st.metric('R-Squared', round(r2_linreg, 4))

# embedding predicted vs actual price plot for linear regression
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred_linreg)
plt.title('Predicted vs Actual Prices - Linear Regression Model')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red',
         lw=2)
st.pyplot(plt)

# embedding residual plot
plt.figure(figsize=(10, 6))
sns.residplot(x=y_test, y=y_pred_linreg, lowess=True, color="g")
plt.title('Residuals of Predictions - Linear Regression Model')
plt.xlabel('Actual Prices')
plt.ylabel('Residuals')
st.pyplot(plt)
```

```python
def perform_linear_regression(X_train, X_test, y_train, y_test):
    linreg_model = LinearRegression()

    # train the model
    linreg_model.fit(X_train, y_train)

    # predict prices
    y_pred_linreg = linreg_model.predict(X_test)

    # evaluate the model
    mse_linreg = mean_squared_error(y_test, y_pred_linreg)
    r2_linreg = r2_score(y_test, y_pred_linreg)

    return [mse_linreg, r2_linreg, y_test, y_pred_linreg]
```

MSE (Mean Squared Error):
Measures average squared prediction error magnitude.

R2 (R-Squared):
coefficient of determination, how close the data are to the fitted regression line, helps assess goodness of fit

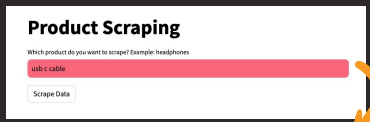
Linear Regression:
MSE: 51.60, R-squared: -0.003
Moderately inaccurate, poor explanatory power

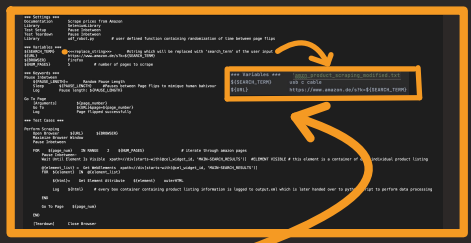
Gradient Boosting:
MSE: 137.40, R-squared: -1.6709
Highly inaccurate, very poor fit.

Gradient Boosting

```python
# GRADIENT BOOSTING
st.subheader('Model: Gradient Boosting')
mse_gbm, r2_gbm, y_test, y_pred_gbm = perform_gradient_boosting(X_train, X_test, y_train, y_test)

# show model performance metrics
st.metric('Mean Squared Error', round(mse_gbm, 4))
st.metric('R-Squared', round(r2_gbm, 4))

# embedding predicted vs actual price plot for gradient boosting
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred_gbm)
plt.title('Predicted vs Actual Prices - GBM Model')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red',
         lw=2)
st.pyplot(plt)

# embedding residual plot
plt.figure(figsize=(10, 6))
sns.residplot(x=y_test, y=y_pred_gbm, lowess=True, color="g")
plt.title('Residuals of Predictions - GBM Model')
plt.xlabel('Actual Prices')
plt.ylabel('Residuals')
st.pyplot(plt)
```

```python
def perform_gradient_boosting(X_train, X_test, y_train, y_test):
    gbm_model = GradientBoostingRegressor()

    # training the model
    gbm_model.fit(X_train, y_train)

    # predict prices
    y_pred_gbm = gbm_model.predict(X_test)

    # model evaluation
    mse_gbm = mean_squared_error(y_test, y_pred_gbm)
    r2_gbm = r2_score(y_test, y_pred_gbm)

    return [mse_gbm, r2_gbm, y_test, y_pred_gbm]
```



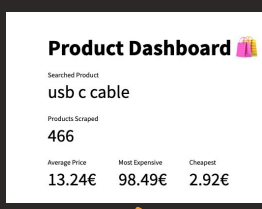


```
st.title('Product Scraping')
search_term = st.text_input('Which product do you want to scrape? Example: headphones')
st.session_state.search_term = search_term

if st.button("Scrape Data"):
    if len(search_term) != 0:
        st.write(f'Started scraping data for {search_term}...')
        scrape_products(search_term)   # running the script, robot framework and scraping products
        st.write('Scraping Completed ✅ ')

        time.sleep(2)

        # creating dashboard
        product_dashboard(search_term)
    else:
        st.write('Please enter a search term.')
```

```
robot_script = open('amzn_product_scraping.txt', 'r').read()
robot_script = robot_script.replace('<<<replace_string>>>', search_term)
print(robot_script)

# writing modified script to a temporary file
with open('amzn_product_scraping_modified.txt', 'w') as file:
    file.write(robot_script)

# running the script, robot framework and scraping products

subprocess.run("./streamlit_subprocess_amzn_scraping_script.sh", shell=True)
```

Scraping Completed ✅

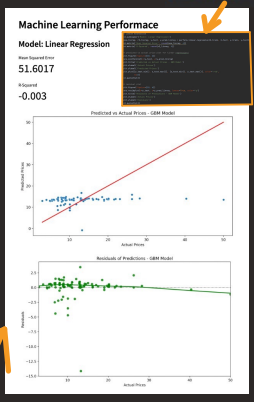
```
robot amzn_product_scraping_modified.txt
python amzn_data_transformation.py
```

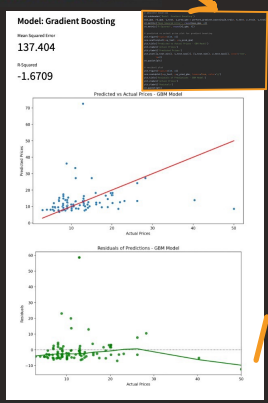



```
def product_dashboard(search_term):

    def load_data():
        data = pd.read_csv('product_table.csv')
        return data

    # Reading in the dataframe
    df = load_data()
```

```
import xml.etree.ElementTree as ET
from lxml import html
import pandas as pd
import json

# Parse the XML file
tree = ET.parse('output.xml')
root = tree.getroot()

product_dict = {}
```

```
def perform_machine_learning(df):...
```



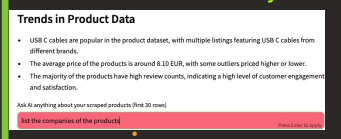



```
# subset dataframe for machine learning
df_for_ml = df[['price_eur', 'review_count', 'rating_stars']]

# feature and target variable selection
X = df_for_ml[['review_count', 'rating_stars']]
y = df_for_ml['price_eur']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

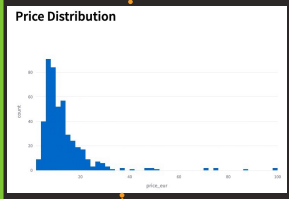
```
def perform_linear_regression(X_train, X_test, y_train, y_test):
    """Goal is to predict the price based on the review count and rating stars of a product"""
    linreg_model = LinearRegression()

    # Train the model
    linreg_model.fit(X_train, y_train)

    # Predict prices
    y_pred_linreg = linreg_model.predict(X_test)

    # Evaluate the model
    mse_linreg = mean_squared_error(y_test, y_pred_linreg)
    r2_linreg = r2_score(y_test, y_pred_linreg)

    return [mse_linreg, r2_linreg, y_test, y_pred_linreg]
```

```
def perform_gradient_boosting(X_train, X_test, y_train, y_test):
    """Goal is to predict the price based on the review count and rating stars of a product"""

    gbm_model = GradientBoostingRegressor(random_state=0)
    gbm_model.fit(X_train, y_train)        # training model
    y_pred_gbm = gbm_model.predict(X_test)  # predicting prices

    # model evaluation
    mse_gbm = mean_squared_error(y_test, y_pred_gbm)
    r2_gbm = r2_score(y_test, y_pred_gbm)

    return [mse_gbm, r2_gbm, y_test, y_pred_gbm]
```

Additional Functionality and Features



AI Prompting



Histogram

Correlation Scatter Plot