EXTERNAL INTEGRATED SUMMATIVE ASSESSMENT

MOCK TEST MEMO

STUDENT NAME & SURNAME	
ID NUMBER	
EISA REGISTRATION NUMBER	
ASSESSMENT CENTRE	
ASSESSMENT CENTRE ACCREDITATION NUMBER	
QUALIFICATION	Occupational Certificate: Data Science Practitioner
SAQA ID	118708
CREDITS	185
PAPER	
DATE OF EISA	
DURATION	
TOTAL MARKS	

General EISA Rules

1. Students are **only** allowed to use the supplied EISA booklets.
2. Students are **only** allowed to use a black pen for their answers.
3. Students to ensure that their name, surname and EISA registration number appears on the front of your EISA booklet.
4. This is an open book examination.
5. All EISA booklets must be handed back to the invigilator intact. No pages may be torn off from the EISA booklet. The removal of EISA booklets from the examination room is prohibited.
6. Students may make use of a calculator in this EISA.
7. Unless this is an online examination where access to a computer will be made available to you; the use of any communication devices, including smart watches, cell phones, tablets, iPads, headphones and laptops are prohibited.
8. All cell phones are to be switched off for the duration of the EISA.
9. The invigilator will not assist you with the explanation of questions related to the EISA.
10. Students are prohibited from conversing in any manner with other students.
11. Students may not leave the examination venue within one hour of the start of the examination and in the last 10 minutes of the allotted examination period.
12. Students who are found to be disruptive and unruly in the assessment centre will be requested to leave the assessment centre by the invigilator.

I HEREBY CONFIRM THAT I HAVE READ THE ABOVE EISA RULES AND DECLARE THAT I UNDERSTAND AND ACCEPT THE RULES.

_____

SIGNATURE OF STUDENT

Candidate Instructions

- Candidates must complete all questions in this EISA.
- Candidates must ensure that they use only a black pen when completing this EISA.
- Should you require additional space to complete your answer, please request additional paper from your invigilator. Ensure that you indicate your name, surname and EISA registration number at the top of the additional paper. Also ensure that the question number is clearly marked on your additional paper.
- There is one handout in this paper, **Handout A – Project Charter (this must be handed in with your answer sheet).**

Section A: Theory Questions

1. Data Collection and Pre-Processing

1.1. Which of the following is a primary data source?

a) Social media posts
b) TV news broadcasts
c) News articles
d) Data from sensors

1.2. What is the first step in data preprocessing?

a) Data visualisation
b) Data cleaning
c) Data collection
d) Data analysis

1.3. Which of the following formats is commonly used to store structured relational data?

a) CSV
b) JPEG
c) MP4
d) JSON

1.4. True or False: Unstructured data is always stored in a database.

a) True
b) False

1.5. Which tool is most used for pre-processing data?

a) Excel
b) Python
c) SQL
d) Tableau

2. Data Analysis Techniques

2.1. Which of the following techniques is used to identify patterns in a dataset?

a) Data visualisation.
b) Data transformation.
c) Statistical analysis.
d) Data collection.

2.2. Which SQL command is used to retrieve data from a database?

a) INSERT
b) SELECT
c) UPDATE
d) DELETE

2.3. Which method would you use to detect trends in a time series dataset?

a) Histogram
b) Linear regression
c) Scatter plot
d) Pie chart

2.4. True or False: Data normalisation is a technique used to reduce data redundancy.

a) True
b) False

2.5. Which of the following tools is commonly used for statistical data analysis?

a) Excel
b) R
c) PowerPoint
d) Word

3. Data Visualization and Reporting

3.1. Which of the following is a common data visualisation tool?

a) Seaborn
b) Jupyter Notebook
c) SQL
d) Hadoop

3.2. True or False: Data storytelling involves explaining data patterns and trends through visualisations.

a) True
b) False

3.3. Which programming language is commonly used for creating data visualisations?

a) Java
b) Python
c) C++
d) HTML

3.4. Which type of chart is best for showing trends over time?

a) Bar chart
b) Pie chart
c) Line chart
d) Scatter plot

3.5. In a descriptive analytics report, what is the main purpose of a visualisation?

a) To summarise data insights
b) To store data
c) To clean data
d) To collect data

Section B: Practical Questions

1. SQL Queries

You are provided with a pre-populated SQLite database named sales.db. Your task is to explore this database and write a series of SQL queries to perform the tasks detailed below. Queries should be optimised to run within 20 seconds or less.

1.1. Determine which product has been purchased the most
Write a query to find the product that has been purchased the most in terms of quantity. The result should display the product name, and the total quantity sold.

See code memo below.

1.2. Identify which store has the widest range of products in stock
Write a query to find out which store carries the most unique products in its inventory (stock). The query should return the store name and the number of different products available.

See code memo below.

1.3. Identify which product was purchased by the most individual users
Write a query to find which product was purchased by the largest number of different users. The result should include the product name and the number of individual users who bought it.

See code memo below.

1.4. Identify the most popular shipping method
Write a query to find the most popular shipping method used in all sales. The result should display the shipping method and the total number of times it has been used.

See code memo below.

1.5. Find the product with the highest quantity in inventory (stock) across both store and warehouse

Write a query to determine which product has the highest combined quantity in both store and warehouse inventories. The result should display the product name and the total quantity in inventory.

See code memo below.

Code Memo:

```
import sqlite3
import os

DATABASE_NAME = 'sales.db'

def most_purchased_product(cursor):
    cursor.execute('''
        SELECT Product.ProductName, SUM(Sales.Quantity) AS TotalQuantity
        FROM Sales
        JOIN Product ON Sales.ProductID = Product.ProductID
        GROUP BY Product.ProductID
        ORDER BY TotalQuantity DESC
        LIMIT 1;
    ''')
    return cursor.fetchall()

def store_most_products(cursor):
    cursor.execute('''
        SELECT Store.Name, Product.ProductName, COUNT(DISTINCT
StoreInventory.ProductID) AS ProductCount
        FROM StoreInventory
        JOIN Store ON StoreInventory.StoreID = Store.StoreID
        JOIN Product ON StoreInventory.ProductID = Product.ProductID
        GROUP BY Store.StoreID
        ORDER BY ProductCount DESC
        LIMIT 1;
    ''')
    return cursor.fetchall()

def most_users_product(cursor):
    cursor.execute('''
        SELECT Product.ProductName, COUNT(DISTINCT Sales.UserID) AS UserCount
        FROM Sales
        JOIN Product ON Sales.ProductID = Product.ProductID
        GROUP BY Product.ProductID
        ORDER BY UserCount DESC
        LIMIT 1;
```

```
    ''')
    return cursor.fetchall()

def most_popular_shipping(cursor):
    cursor.execute('''
        SELECT ShippingOptions.ShippingMethod, COUNT(Sales.ShippingID) AS
ShippingCount
        FROM Sales
        JOIN ShippingOptions ON Sales.ShippingID = ShippingOptions.ShippingID
        GROUP BY ShippingOptions.ShippingID
        ORDER BY ShippingCount DESC
        LIMIT 1;
    ''')
    return cursor.fetchall()

def highest_inventory_product(cursor):
    cursor.execute('''
        SELECT Product.ProductName, SUM(StoreInventory.Quantity +
WarehouseInventory.Quantity) AS TotalQuantity
        FROM Product
        LEFT JOIN StoreInventory ON Product.ProductID = StoreInventory.ProductID
        LEFT JOIN WarehouseInventory ON Product.ProductID =
WarehouseInventory.ProductID
        GROUP BY Product.ProductID
        ORDER BY TotalQuantity DESC
        LIMIT 1;
    ''')
    return cursor.fetchall()

def print_results(results):
    for result in results:
        print(result)
    print()

if __name__ == '__main__':
    if not os.path.exists(DATABASE_NAME):
        print('Database not found')
        print('Please run create_insert_db.py first')
        exit()

    conn = sqlite3.connect(DATABASE_NAME)
    cursor = conn.cursor()
    print('Database successfully loaded')

    # 1.1. Determine which product has been purchased the most.
    print('Most Purchased Product:')
    print_results(most_purchased_product(cursor))
```

```
    # 1.2. Identify which store has the widest range of products in stock.
    print('Store with Most Unique Products:')
    print_results(store_most_products(cursor))

    # 1.3. Identify which product was purchased by the most individual users.
    print('Product which was purchased by the most users:')
    print_results(most_users_product(cursor))

    # 1.4. Identify the most popular shipping method.
    print('Most Popular Shipping method:')
    print_results(most_popular_shipping(cursor))

    # 1.5. Find the product with the highest quantity in inventory (stock) across both store and warehouse.
    print('Highest Inventory Product:')
    print_results(highest_inventory_product(cursor))

    conn.commit()
    conn.close()
```

2. Database Population and Data Validation

2.1. Database Population

You are provided with a file named books_data.json which contains unstructured data from an online bookstore. Your task is to explore this file, structure, and validate the data as needed to perform the tasks detailed below.

2.1.1. Write a python script that creates an SQLite database named books.db with the following tables and the appropriate FKs (foreign keys) to maintain relationships between the tables

- Categories**:** Stores the category name of each book
- Books**:** Stores the title, price, rating, category ID, and description of each book.
- Stock**:** Stores the stock status (in stock or not) and the stock count for each book.
- ProductIDs**:** Stores the unique product ID (UPC) for each book.

See code memo below.

2.1.2. Write a python script to structure and insert the unstructured data from books_data.json into books.db

- Data should be written to the most suitable table.
- Data should be correctly linked via suitable foreign keys.

See code memo below.

Code Memo:

```
import os
import json
import sqlite3

def load_books_data():
    if not os.path.exists('books_data.json'):
        raise FileNotFoundError('books_data.json not found')

    with open('books_data.json', 'r') as f:
        books_data = json.load(f)
```

```
    return books_data

def create_db(conn, cursor):
    cursor.execute('''
    CREATE TABLE Categories (
        category_id INTEGER PRIMARY KEY AUTOINCREMENT,
        category_name TEXT UNIQUE
    )
    ''')

    cursor.execute('''
    CREATE TABLE Books (
        book_id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT,
        price REAL,
        rating INTEGER,
        category_id INTEGER,
        description TEXT,
        FOREIGN KEY (category_id) REFERENCES Categories(category_id)
    )
    ''')

    cursor.execute('''
    CREATE TABLE Stock (
        book_id INTEGER,
        in_stock BOOLEAN,
        stock_count INTEGER,
        FOREIGN KEY (book_id) REFERENCES Books(book_id)
    )
    ''')

    cursor.execute('''
    CREATE TABLE ProductIDs (
        book_id INTEGER,
        product_id TEXT,
        FOREIGN KEY (book_id) REFERENCES Books(book_id)
    )
    ''')

    conn.commit()

    return conn, cursor

def insert_data(conn, cursor, books_data):
    for book in books_data:
        category = book['category']
```

```
        # Insert or ignore the category (Avoids duplicate entries)
        cursor.execute('''
        INSERT OR IGNORE INTO Categories (category_name)
        VALUES (?)
        ''', (category,))

        # Get the category_id
        cursor.execute('''
        SELECT category_id FROM Categories WHERE category_name = ?
        ''', (category,))
        category_id = cursor.fetchone()[0]

        # Insert the book details
        cursor.execute('''
        INSERT INTO Books (title, price, rating, category_id, description)
        VALUES (?, ?, ?, ?, ?)
        ''', (book['title'], book['price'], book['rating'], category_id, book['description']))

        # Get the book_id
        cursor.execute('''
        SELECT book_id FROM Books WHERE title = ?
        ''', (book['title'],))
        book_id = cursor.fetchone()[0]

        # Insert the stock details
        cursor.execute('''
        INSERT INTO Stock (book_id, in_stock, stock_count)
        VALUES (?, ?, ?)
        ''', (book_id, book['in_stock'], book['stock_count']))

        # Insert the product_id
        cursor.execute('''
        INSERT INTO ProductIDs (book_id, product_id)
        VALUES (?, ?)
        ''', (book_id, book['product_id']))

    conn.commit()

    return conn, cursor

if __name__ == '__main__':
    books_data = load_books_data()

    conn = sqlite3.connect('books.db')
    cursor = conn.cursor()

    # 2.1.1. Write a python script that creates an SQLite database named books.db
```

```
conn, cursor = create_db(conn, cursor)

# 2.1.2. Write a python script to structure and insert the unstructured data from books_data.json into books.db
conn, cursor = insert_data(conn, cursor, books_data)
```

2.2. Data validation

Using books.db, write a series of SQL queries to perform the tasks detailed below to validate the data.

2.2.1. Get all books in a category

Write a Python function that, given a category name, returns a list of books (title and price) that belong to that category from the Books table.

See code memo below.

2.2.2. Check stock status for a book

Write a Python function that, given a book title, checks if the book is currently in stock by querying the Stock table.

See code memo below.

2.2.3. Find the average rating for each category

Write a Python function that, given a category name, calculates the average rating of all books in that category.

See code memo below.

2.2.4. Find books that are below a certain price

Write a Python function that retrieves the title and price of all books that are priced below a given threshold.

See code memo below.

2.2.5. Identify the best-stocked book within a specific category

Write a Python function that, given a category name, finds the book with the highest stock count in that category.

See code memo below.

Code Memo:

```
import sqlite3

def get_books_by_category(cursor, category_name):
    cursor.execute('''
    SELECT b.title, b.price
    FROM Books b
    JOIN Categories c ON b.category_id = c.category_id
    Where c.category_name = ?
    ''', (category_name,))
    books = cursor.fetchall()
    return books

def is_book_in_stock(cursor, book_title):
    cursor.execute('''
    SELECT s.in_stock
    FROM Books b
    JOIN Stock s ON b.book_id = s.book_id
    WHERE b.title = ?
    ''', (book_title,))
    in_stock = cursor.fetchone()[0]
    return in_stock

def get_average_rating_by_category(cursor, category_name):
    cursor.execute('''
    SELECT AVG(b.rating)
    FROM Books b
    JOIN Categories c ON b.category_id = c.category_id
    WHERE c.category_name = ?
    ''', (category_name,))
    avg_rating = cursor.fetchone()[0]
    return avg_rating

def get_books_below_price(cursor, price):
    cursor.execute('''
    SELECT b.title, b.price
    FROM Books b
    JOIN ProductIDs p ON b.book_id = p.book_id
    WHERE b.price < ?
    ''', (price,))
    books = cursor.fetchall()
    return books
```

```
def get_most_stocked_book_by_category(cursor, category_name):
    cursor.execute('''
    SELECT b.title, s.stock_count
    FROM Books b
    JOIN Stock s ON b.book_id = s.book_id
    JOIN Categories c ON b.category_id = c.category_id
    WHERE c.category_name = ?
    ORDER BY s.stock_count DESC
    LIMIT 1
    ''', (category_name,))
    most_stocked_book = cursor.fetchone()
    return most_stocked_book

if __name__ == '__main__':
    conn = sqlite3.connect('books.db')
    cursor = conn.cursor()

    # 2.2.1. Get all books in a category
    category_name = 'New Adult'
    books = get_books_by_category(cursor, category_name)
    print(f'Books in the category "{category_name}":')

    for book in books:
        print(f'\t{book[0]} - ${book[1]}')

    print()

    # 2.2.2. Check stock status for a book
    book_title = "Its Only the Himalayas"
    in_stock = is_book_in_stock(cursor, book_title)
    print(f'Is "{book_title}" in stock? {True if in_stock==1 else False}')

    print()

    # 2.2.3. Find the average rating for each category
    category_name = 'Fiction'
    avg_rating = get_average_rating_by_category(cursor, category_name)
    print(f'Average rating for the category "{category_name}": {avg_rating}')

    print()

    # 2.2.4. Find books that are below a certain price
    price = 12.00
    books = get_books_below_price(cursor, price)
    print(f'Books below ${price}:')
    for book in books:
        print(f'\t{book[0]} - ${book[1]}')
```

```
    print()

    # 2.2.5. Identify the best-stocked book within a specific category.
    category_name = 'Science'
    most_stocked_book = get_most_stocked_book_by_category(cursor, category_name)
    print(f'Most stocked book in the category "{category_name}": {most_stocked_book[0]}
({most_stocked_book[1]} in stock)')

    print()
```

3. Wine Quality Dataset Analysis

You are provided with a file named wine_quality.zip which contains two datasets related to red and white vinho verde wine samples, from the north of Portugal.

Your goal is to explore this dataset. Clean it if necessary. Identify any biases. Then perform a series of regression analyses to uncover trends. Detect potential biases due to excluded variables, and finally make predictions on wine quality.

3.1. Loading the Dataset and Data Preparation

3.1.1. Data loading

Load the dataset using pandas.DataFrame and ensure the data is correctly formatted (i.e., all columns should be numeric).

See code memo below.

3.1.2. Data Preparation

Write a function to separate the target variable (quality) from the features and prepare the data for further analysis (e.g., scaling or normalisation if necessary).

See code memo below.

3.2. Visualisations and Exploratory Data Analysis

Analyse the distribution of classes and the relationships between features to identify patterns or biases in the data.

3.2.1. Class Distribution Visualisation

Task: Implement a function that visualises the distribution of the wine quality classes. This will help identify any imbalances in the dataset.

See code memo below.

Question: Is the dataset imbalanced in terms of wine quality classes? What impact could this have on your regression models?

Class Distribution of Wine Quality

The bar chart visualizing the class distribution of wine quality shows a clear imbalance across the different classes. The majority of wines fall in the middle

range of quality, particularly between 5 and 6, with much fewer wines rated 3, 4, 7, or 8

Imbalance Analysis: *There is a heavy concentration of samples in the mid-range quality scores (5 and 6), with fewer instances at both the lower (3, 4) and higher (7, 8) ends. This class imbalance could bias regression models, leading to poorer predictions for wines of lower or higher quality. Since the dataset contains fewer samples for these extremes, the model might under-perform when predicting these classes.*

3.2.2. Feature Distribution and Outlier Detection

Task: Create box plots for each feature to visualise the distribution of values and detect any outliers.

See code memo below.

Question: Which features have significant outliers? How might these outliers affect model performance?

Feature Distribution and Outlier Detection

The box plots for various features in the dataset reveal some critical observations:

Outliers:

- ***Residual sugar:*** *There are extreme values beyond the upper whisker, indicating significant outliers.*
- ***Chlorides:*** *Similarly, this feature exhibits a few values significantly above the typical range.*
- ***Sulfates and total sulfur dioxide:*** *Both features also have visible outliers.*

Impact on Model Performance:

Outliers can have a disproportionate impact on model performance, especially for regression models that are sensitive to variance in the data. Without handling these, the model could be skewed, potentially misrepresenting the relationships between features and wine quality. Standard preprocessing techniques like scaling or trimming could help mitigate their impact.

3.2.3. Correlation Heatmap

Task: Generate a heatmap that shows the correlation (relationship) between features and the target variable (quality). This will help you understand which features are most important to the prediction of wine quality.

See code memo below.

Question: Which features are strongly correlated with wine quality? Do these correlations suggest any biases or trends that need further investigation?

Correlation Heatmap

The heatmap shows the correlation between features and the target variable (quality)

***Strong Correlations with Quality**:*

- ***Alcohol** (0.48): Positively correlated with wine quality, indicating that higher alcohol levels tend to be associated with better quality.*
- ***Sulphates** (0.25): Moderate positive correlation, which may suggest that wines with more sulphates are perceived as higher quality.*

***Negative Correlations**:*

- ***Volatile acidity** (-0.39): Strong negative correlation with quality, suggesting that higher acidity levels are associated with lower quality wines.*
- ***Total sulfur dioxide** (-0.19): Slight negative correlation, indicating wines with higher sulfur dioxide levels might be rated lower.*

***Potential Biases**:*

The correlation heatmap suggests some trends, such as the strong influence of alcohol content, which could introduce bias if not properly considered. High alcohol wines might be systematically rated higher, possibly skewing the model's understanding of "quality."

Code Memo:

```
import pandas as pd
import seaborn
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import os

# https://archive.ics.uci.edu/dataset/186/wine+quality
DATA_PATH = 'wine+quality/winequality-red.csv'

def load_data():
    if not os.path.exists(DATA_PATH):
        raise FileNotFoundError(f"File not found: {DATA_PATH}, please check the file path
variable DATA_PATH")

    data = pd.read_csv(DATA_PATH, delimiter=';')
    data = pd.DataFrame(data)
    return data

def process_data(data):
```

```
    wine_quality = data['quality']
    data = data.drop('quality', axis=1)
    data = data.apply(pd.to_numeric, errors='coerce')

    return data, wine_quality

def class_distribution_check(data, quality):
    plt.figure(figsize=(10, 6))
    seaborn.countplot(x=quality)
    plt.title('Class Distribution of Wine Quality')
    plt.xlabel('Wine Quality')
    plt.ylabel('Count')
    plt.savefig('plots/visualizations/class_distribution.pdf', format='pdf')

def feature_distribution_and_outliers(data):
    num_features = len(data.columns)
    fig, axes = plt.subplots(nrows=(num_features + 1) // 2, ncols=2, figsize=(20, 5 *
((num_features + 1) // 2)))
    axes = axes.flatten()

    for i, feature in enumerate(data.columns):
        seaborn.boxplot(x=data[feature], ax=axes[i])
        axes[i].set_title(f'Distribution of {feature}')
        axes[i].set_xlabel(feature)

    plt.tight_layout()
    plt.savefig('plots/visualizations/feature_distribution.pdf', format='pdf')

def correlation_analysis(data, quality):
    data['quality'] = quality

    plt.figure(figsize=(12, 10))
    seaborn.heatmap(data.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
    plt.title('Correlation Heatmap of Wine Features')
    plt.savefig('plots/visualizations/correlation_heatmap.pdf', format='pdf')

def main():
    data, wine_quality = process_data(load_data())

    # 3.2.1. Class Distribution Visualisation
    class_distribution_check(data, wine_quality)

    # 3.2.2. Feature Distribution and Outlier Detection
    feature_distribution_and_outliers(data)

    # 3.2.3. Correlation Heatmap
    correlation_analysis(data, wine_quality)
```

```
if __name__ == '__main__':
    if not os.path.exists('plots'):
        os.mkdir('plots')
    if not os.path.exists('plots/visualizations'):
        os.mkdir('plots/visualizations')
    main()
```

3.3. Regression Models

Apply different regression techniques to predict wine quality based on the dataset, evaluate the performance of their models, and visualise the results.

3.3.1. Simple Linear Regression

Implement a simple linear regression model using one feature (e.g., alcohol) to predict wine quality. Visualise the regression line and the residuals.

See code memo below.

3.3.2. Multiple Linear Regression

Write a function to perform multiple linear regression using several features (e.g., alcohol, volatile acidity, sulphates, citric acid) to predict wine quality. Visualise the model coefficients.

See code memo below.

3.3.3. Polynomial Regression

Create a polynomial regression model using one feature (e.g., alcohol) with a degree of 2 or higher. Compare its performance to the simple linear regression model.

See code memo below.

Code Memo:

```
import os
import seaborn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import PolynomialFeatures

# https://archive.ics.uci.edu/dataset/186/wine+quality
DATA_PATH = 'wine+quality/winequality-red.csv'

def load_data():
    if not os.path.exists(DATA_PATH):
        raise FileNotFoundError(f"File not found: {DATA_PATH}, please check
the file path variable DATA_PATH")

    data = pd.read_csv(DATA_PATH, delimiter=';')
    data = pd.DataFrame(data)
    df = data.apply(pd.to_numeric, errors='coerce')
    return df

def perform_simple_regression(df, feature='alcohol'):
    X = df[[feature]]
    y = df['quality']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    # Visualization
    plt.figure(figsize=(10, 6))
    seaborn.regplot(x=X_test[feature], y=y_test, scatter_kws={'alpha':0.5})
    plt.title(f'Simple Linear Regression: {feature} vs Wine Quality')
    plt.xlabel(feature)
    plt.ylabel('Wine Quality')
    plt.savefig(f'plots/regression/simple_regression_{feature}.pdf',
format='pdf')

    # Residual plot
    plt.figure(figsize=(10, 6))
    seaborn.residplot(x=y_pred, y=y_test - y_pred, lowess=True,
scatter_kws={'alpha': 0.5})
    plt.title('Residual Plot')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.savefig(f'plots/regression/residual_plot_{feature}.pdf', format='pdf')
```

```
    print(f'R-squared: {r2_score(y_test, y_pred):.4f}')
    print(f'Mean Squared Error: {mean_squared_error(y_test, y_pred):.4f}')

    return model

def perform_multiple_regression(df, features=['alcohol', 'volatile acidity',
'sulphates', 'citric acid']):
    X = df[features]
    y = df['quality']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    # Visualization of coefficients
    plt.figure(figsize=(10, 6))
    seaborn.barplot(x=model.coef_, y=features)
    plt.title('Multiple Linear Regression Coefficients')
    plt.xlabel('Coefficient Value')
    plt.ylabel('Features')
    plt.savefig('plots/regression/multiple_regression.pdf', format='pdf')

    print(f'R-squared: {r2_score(y_test, y_pred):.4f}')
    print(f'Mean Squared Error: {mean_squared_error(y_test, y_pred):.4f}')

    return model

def perform_polynomial_regression(df, feature='alcohol', degree=2):
    X = df[[feature]]
    y = df['quality']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    poly = PolynomialFeatures(degree=degree)
    X_poly_train = poly.fit_transform(X_train)
    X_poly_test = poly.transform(X_test)

    model = LinearRegression()
    model.fit(X_poly_train, y_train)

    y_pred = model.predict(X_poly_test)

    # Visualization
    plt.figure(figsize=(10, 6))
```

```
    seaborn.scatterplot(x=X_test[feature], y=y_test, alpha=0.5)
    X_plot = np.linspace(X_test[feature].min(), X_test[feature].max(),
100).reshape(-1, 1)
    y_plot = model.predict(poly.transform(X_plot))
    plt.plot(X_plot, y_plot, color='red')
    plt.title(f'Polynomial Regression (degree {degree}): {feature} vs Wine
Quality')
    plt.xlabel(feature)
    plt.ylabel('Wine Quality')
    plt.savefig(f'plots/regression/polynomial_regression_{feature}.pdf',
format='pdf')

    print(f'R-squared: {r2_score(y_test, y_pred):.4f}')
    print(f'Mean Squared Error: {mean_squared_error(y_test, y_pred):.4f}')

    return model

def main():
    df = load_data()

    # 3.3.1. Simple Linear Regression
    perform_simple_regression(df)
    perform_simple_regression(df, 'volatile acidity')
    perform_simple_regression(df, 'sulphates')

    # 3.3.2. Multiple Linear Regression
    perform_multiple_regression(df)

    # 3.3.3. Polynomial Regression
    perform_polynomial_regression(df)
    perform_polynomial_regression(df, 'volatile acidity')
    perform_polynomial_regression(df, 'sulphates')

if __name__ == '__main__':
    if not os.path.exists('plots'):
        os.mkdir('plots')
    if not os.path.exists('plots/regression'):
        os.makedirs('plots/regression')
    main()
```

MICTSETA | Media, Information And Communication Technologies Sector Education And Training Authority

SHAPING SKILLS, PIONEERING INDUSTRIES, EMPOWERING FUTURES

MICTSETA | Media, Information And Communication Technologies Sector Education And Training Authority

SHAPING SKILLS, PIONEERING INDUSTRIES, EMPOWERING FUTURES