UDEMY COURSE DATA ANALYSIS

Presented by: Marwan Mohamed Abdellah

IMT SCHOOL
Data Analysis Diploma

Table of Contents

1. Project Introduction

This project aims to apply data analysis techniques to a real-world dataset (Udemy Courses) obtained from Kaggle. The goal is to explore trends, identify insights, and potentially make recommendations for instructors or the Udemy platform itself.

2. Methodology

Importing Libraries

```
1. import numpy as np
2. import pandas as pd
3. import matplotlib.pyplot as plt
4. import seaborn as sns
5. import mysql.connector
6.
```

Importing Data

```
1. df =
pd.read_csv("D:/Courses/Data_Analysis/IMT/final_project/data/udemy_output_All_Finance__Acco
unting_p1_p626.csv")
2. df.head()
3.
```

3. Dataset Overview

Explanatory Data Analysis (EDA)

1. **Checking Data Formatting**
2. **Data Summary and Unique Values**
3. **Checking for Duplicated Values**
4. **Checking for Null Values**

Checking Data Formatting

```
1. df.shape        ## This function returns the number of rows and columns of the dataset

2. df.info()
## This function prints information about a DataFrame including the index dtype and column
dtypes, non-null values and memory usage.
```

Features Explanation

1. **id:** The course ID of that particular course.
2. **title:** Unique names of the courses under the development category on Udemy.
3. **url:** URL of the course.
4. **is_paid:** Boolean indicating whether the course is paid or free.
5. **num_subscribers:** Number of subscribers.
6. **avg_rating:** Average rating of the course.
7. **avg_rating_recent:** Reflects recent changes in ratings.
8. **num_reviews:** Number of reviews received.
9. **num_published_lectures:** Number of lectures offered.
10. **num_published_practice_tests:** Number of practice tests offered.
11. **created:** Time of course creation.
12. **published_time:** Time of publishing.
13. **discounted_price_amount:** Discounted price of the course.
14. **discounted_price_currency:** Currency of the discounted price.
15. **price_detail_amount:** Original price of the course.
16. **price_detail_currency:** Currency of the original price.

Data Summary and Unique Values

```
3. df.describe()
## This function generates descriptive statistics that summarize the central tendency, dispersion and
shape of a dataset's distribution, excluding NaN values.

4. df.nunique()
## This function returns the number of unique values in each column.
```

Using these functions, we successfully identified the essential columns for analysis and pinpointed those with null or NaN values that require imputation or handling to ensure data consistency and completeness.

Checking for Duplicated Values

```
1. df.duplicated().sum()
## This function returns the number of duplicate rows in the DataFrame.
```

Checking for Null Values

```
1. df.isna().sum()
## This function returns the number of missing values in the dataset.
```

---

4. Data Cleaning and Transformation

Dropping Unnecessary Columns

```
df.drop(['url', 'is_wishlisted',
         'discount_price__currency', 'discount_price__price_string',
         'avg_rating_recent','price_detail__price_string',
         'num_published_practice_tests'], axis = 1, inplace = True)

```

Renaming Columns

```
df.rename({"title":"Title","avg_rating": "Average rating",
        "rating":"Rating",
        "num_subscribers":"Subscribers", "published_time":"Time",
        "num_reviews": "Num reviews",
        "num_published_lectures":"Lectures",
        "discount_price__amount":"Discount price",
        "price_detail__amount":"Price",
        "price_detail__currency": "Currency"}, axis = 1, inplace = True)
```

Handling Data Formatting

1) Date Format for the DataFrame

```
df["Time"] = pd.to_datetime(df["Time"])
df['created'] = pd.to_datetime(df['created'])
## This function converts the argument to datetime.
```

2) Data Format for SQL Integration

```
df['created'] = df['created'].dt.strftime('%Y-%m-%d %H:%M:%S')
df['Time'] = df['Time'].dt.strftime('%Y-%m-%d %H:%M:%S')
## This function converts the argument to a string in the specified date format for SQL.
```

3) Converting the currency amount from INR to USD

```
conversion_rate = 0.0121817  # Example conversion rate (1 Rupee = 0.0121817 USD)
df['Price'] = df['Price'] * conversion_rate
df['Discount price'] = df['Discount price'] * conversion_rate
## This function multiplies the argument by the conversion_rate to convert the currency to USD.
```

4) Convert the currency detail from INR to USD

```
df['Currency'].replace("INR", "USD", inplace = True)
## This function replaces the value 'INR' with 'USD' in the 'Currency' column.
```

Adding ‘Discount Percentage’ Column

```
df['Discount Percentage'] = ((df['Price'] - df['Discount price']) / df['Price']) * 100
df['Discount Percentage'] = df['Discount Percentage'].round(2).astype(str) + "%"
## This function calculates the discount percentage and rounds it to 2 decimal places and adds it to the
'Discount Percentage' column.
```

Handling Missing Values

1) Filling the Null Values in the Price and Discount Price Columns

```
1. for column in ['Price','Discount price']:
2.     df.fillna({column : df[column].median()}, inplace = True)
## This function fills the missing values in the 'Price' and 'Discount price' columns with the median
value of the respective columns.
```

2) Fixing the Null Values in the Currency Column

```
1. 1. df.fillna({"Currency":"Unknown"}, inplace = True)
## Changing all the null values in price currency column to 'unknown'

3. df['Currency'].unique()
## This function returns the unique values in the 'Currency' column.
```

Typos correction

```
1. df['Title'] = df['Title'].str.lower()
# Converting the title column to lower case
2. df['Title'] = df['Title'].str.strip().replace(r'\s+', ' ', regex=True)
# Removing extra spaces
3. df['Title'] = df['Title'].str.replace(r'[^\w\s]', '', regex=True)
# Removing special characters
```

Categorizing the Courses

```
 1. def categorize_title(title):
 2.     title_lower = title.lower()
 3.     if 'sql' in title_lower or 'mysql' in title_lower or 'database' in title_lower:
 4.         return 'Database'
 5.     elif 'tableau' in title_lower or 'power bi' in title_lower or 'data viz' in title_lower:
 6.         return 'Data Visualization'
 7.     elif 'excel' in title_lower or 'spreadsheet' in title_lower:
 8.         return 'Spreadsheet'
 9.     elif any(kw in title_lower for kw in ['agile', 'scrum', 'pmp', 'project management']):
10.         return 'Project Management'
11.     elif any(kw in title_lower for kw in ['financial', 'finance', 'accounting']):
12.         return 'Finance'
13.     elif 'mba' in title_lower or 'business' in title_lower or 'enterprise' in title_lower:
14.         return 'Business'
15.     elif any(kw in title_lower for kw in ['write', 'writing', 'editorial']):
16.         return 'Writing'
17.     elif 'sale' in title_lower or 'marketing' in title_lower:
18.         return 'Sales/Marketing'
19.     elif any(kw in title_lower for kw in ['data science', 'analytics', 'machine learning']):
20.         return 'Data Science'
21.     elif 'management' in title_lower:
22.         return 'Management'
23.     elif 'leadership' in title_lower:
24.         return 'Leadership'
25.     elif 'communication' in title_lower:
26.         return 'Communication'
27.     else:
28.         return 'Other'
29.
30. df['Category'] = df['Title'].apply(categorize_title)
```

After confirming that all necessary data is retained, columns were appropriately renamed, and date fields were formatted to ensure compatibility with SQL. Additional refinements included the creation of a new column for discount percentage, which was verified to contain no null values. Currency values were standardized to USD, and titles were cleaned by removing special characters, extra spaces, and capital letters to maintain consistency. Furthermore, the data was categorized into specific groups to facilitate more detailed analysis. Subsequently, we will proceed with outlier detection to identify and address any anomalies in the dataset.

5. Outlier Detection

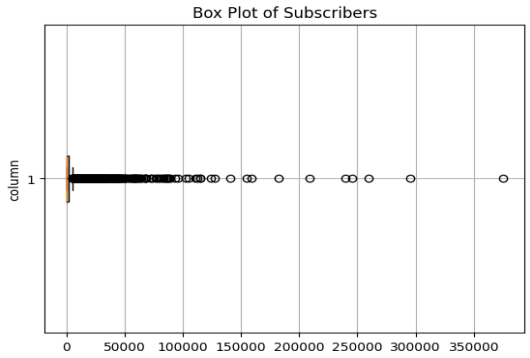
Identifying Numerical Values that might hold potential outliers

```
numerical_columns = ['Subscribers','Price','Discount price']
## This function returns the numerical columns in the dataset.
```

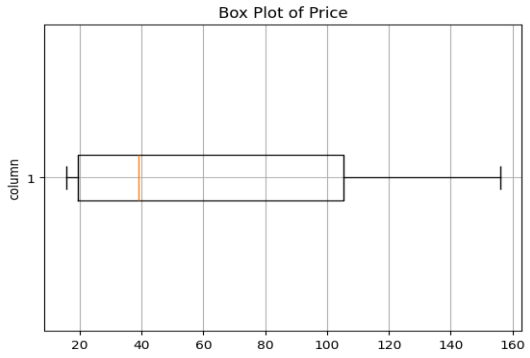
Using Boxplot from Matplotlib Library to see the Variance in our dataset

```
for column in numerical_columns:
    plt.boxplot(df[column],vert=False)
    plt.title(f'Box Plot of {column}')
    plt.ylabel('column')
    plt.grid()
    plt.show()
```

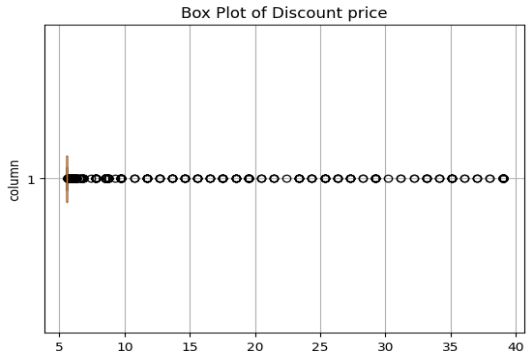
The Diagrams



The "Subscribers" column. The plot indicates that most courses have relatively low subscriber counts, with several outliers exceeding 350,000. This suggests that while the majority of courses attract smaller audiences, some naturally draw significantly larger numbers, reflecting their popularity or market appeal.



The "Price" column has no outliers, reflecting a consistent range of course prices. This variation is typical, as different courses cater to varying market demands and pricing strategies.



The "Discount Price" column. The plot indicates that most of the discount prices are tightly clustered near the lower end of the range, with a few outliers extending toward higher values. This suggests that while the majority of discounts are small or consistent, some courses have significantly larger discount prices, which might reflect promotional strategies or premium pricing adjustments.

I wouldn't remove any outliers, as they represent valuable data points that contribute to a more accurate and meaningful analysis.

6. Exporting the final dataset

Exporting the final dataset to a CSV file using Pandas library

```
df.to_csv('cleaned_data.csv', index=False)
## This function exports the cleaned data to a CSV file.
```

7. Database Integration

MySQL Implementation

1) Creating a connection function.

```
def create_connection():
    """Create and return a MySQL database connection and cursor."""

    # Establish the connection
    mydb = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database=""        # Need to fill this path for the intended database
    )

    # Create a cursor
    conn = mydb.cursor()

    print("Connection to the database was successful.")
    print("=" * 50)
    return mydb, conn
## This function creates a connection to the MySQL database and returns the connection and cursor.
```

2) Creating the table using our dataset features

```
# Re-connecting to the database
mydb, conn = create_connection()

# Dropping the existing database if it exists and creating a new one
conn.execute("DROP DATABASE IF EXISTS Course_db")
conn.execute("CREATE DATABASE IF NOT EXISTS Course_db")
conn.execute("USE Course_db")

# Creating the table with the specified column names
create_sql = """
CREATE TABLE IF NOT EXISTS courses_table (
    id INT PRIMARY KEY,
    Title TEXT,
    is_paid BOOLEAN,
    Subscribers INT,
    `Average rating` FLOAT,
    Rating FLOAT,
    `Num reviews` INT,
    Lectures INT,
    created DATETIME,
    Time DATETIME,
    `Discount price` FLOAT,
    Price FLOAT,
    Currency TEXT,
    `Discount Percentage` FLOAT,
    Category TEXT
);
"""
conn.execute(create_sql)

conn.close()
```

3) **Populating the database with the cleaned data.**

```
# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Inserting data into the MySQL table
insert_sql = """
INSERT INTO courses_table (id, Title, is_paid, Subscribers, `Average rating`, Rating, `Num reviews`,
                                    Lectures, created, Time, `Discount price`, Price, Currency,
`Discount Percentage`, Category)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
"""

data_values = [
    (
        row['id'], row['Title'], row['is_paid'], row['Subscribers'], row['Average rating'],
        row['Rating'], row['Num reviews'], row['Lectures'],
        row['created'], row['Time'], row['Discount price'], row['Price'],
        row['Currency'], row['Discount Percentage'], row['Category']
    )
    for _, row in df.iterrows()
]

# Executing batch insert
conn.executemany(insert_sql, data_values)
mydb.commit()

print("Data successfully saved to the MySQL database.")

# Closing the connection
conn.close()
mydb.close()
```

---

8. Advanced Analysis

MySQL Queries

1) Top Rated Courses

```
# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Sorting by Rating (Top 10 Courses)
Sort_by_Rating = """
SELECT Title, `Average rating`
FROM courses_table
ORDER BY `Average rating` DESC
LIMIT 10;
"""

# Execute the query
conn.execute(Sort_by_Rating)

# Fetch the results
results = conn.fetchall()

# Check if any results are found and print them
if results:
    print("Top 10 Courses by Rating:")
    print("-"*30)
    for row in results:
        print(row)
        print("-"*70)
else:
    print("No results found.")
```

2) Top Subscribed Courses

```
# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Sorting by Subscribers (Top 10 Courses)
Sort_by_Subscribers = """
SELECT Title, Subscribers
FROM courses_table
ORDER BY Subscribers DESC
LIMIT 10;
"""

# Execute the query
conn.execute(Sort_by_Subscribers)

# Fetch the results
results = conn.fetchall()

# Check if any results are found and print them
if results:
    print("Top 10 Courses by Subscribers:")
    print("-"*30)
    for row in results:
        print(row)
        print("-"*70)
else:
    print("No results found.")

```

3) Top 10 Courses by Discount Percentage

```
# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Sorting by Discount Percentage (Top 10 Courses)
Sort_by_Discount_Percentage = """
SELECT Title, `Discount Percentage`
FROM courses_table
ORDER BY `Discount Percentage` DESC
LIMIT 10;
"""

# Execute the query
conn.execute(Sort_by_Discount_Percentage)

# Fetch the results
results = conn.fetchall()

# Check if any results are found and print them
if results:
    print("Top 10 Courses by Discount Percentage:")
    print("-"*40)
    for row in results:
        print(row)
        print("-"*70)
else:
    print("No results found.")

```

4) Top 10 Most Expensive Courses

```
# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Sorting by Discount Percentage (Top 10 Courses)
Sort_by_most_expensive = """
SELECT Title, price
FROM courses_table
ORDER BY price DESC
LIMIT 10;
"""

# Execute the query
conn.execute(Sort_by_most_expensive)

# Fetch the results
results = conn.fetchall()

# Check if any results are found and print them
if results:
    print("Top 10 Most Expensive Courses in USD: ")
    print("-"*40)
    for row in results:
        print(row)
        print("-"*70)
else:
    print("No results found.")
```

Group Insights

1) Number of courses in each category

```
# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Calculating the number of courses per price category
Sort_by_price_category = """
SELECT Category, COUNT(*) AS num_courses
FROM courses_table
GROUP BY Category;
"""

# Execute the query
conn.execute(Sort_by_price_category)

# Fetch the results
results = conn.fetchall()

# Check if any results are found and print them
if results:
    print("Count of courses for each Price Category: ")
    print("-"*40)
    for row in results:
        print(row)
        print("-"*70)
else:
    print("No results found.")
```

2) Average number of reviews per category

```
# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Calculating the average number of reviews per Price category
Sort_by_reviews_category = """
SELECT Category, AVG(`Num reviews`) AS avg_reviews
FROM courses_table
GROUP BY Category;
"""

# Execute the query
conn.execute(Sort_by_reviews_category)

# Fetch the results
results = conn.fetchall()

# Check if any results are found and print them
if results:
    print("Average Number of Reviews per Price Category: ")
    print("-"*50)
    for row in results:
        print(row)
        print("-"*70)
else:
    print("No results found.")
```

Pricing and Discounts

1) Calculate average discount percentage offered per category

```
# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Categorizing by Price Range
Sort_by_avg_discount_percentage = """
SELECT Category, AVG(`Discount Percentage`) AS avg_discount_percentage
FROM courses_table
GROUP BY Category;
"""

# Execute the query
conn.execute(Sort_by_avg_discount_percentage)

# Fetch the results
results = conn.fetchall()

# Check if any results are found and print them
if results:
    print("Average Discount Percentage Offered per Price Category: ")
    print("-"*50)
    for row in results:
        print(row)
        print("-"*70)
else:
    print("No results found.")
```

2) Calculate the median price for courses in each category. (only using paid courses)

```
# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Categorizing paid and free courses based on 'is_paid' column
calculate_median = """
SELECT
    price
FROM courses_table
WHERE is_paid = 1 AND price IS NOT NULL
ORDER BY price;
"""

# Execute the query
conn.execute(calculate_median)

# Fetch all the prices
prices = conn.fetchall()

# Calculate median
num_prices = len(prices)
if num_prices > 0:
    sorted_prices = [price[0] for price in prices]
    if num_prices % 2 == 1:  # Odd number of prices
        median_price = sorted_prices[num_prices // 2]
    else:  # Even number of prices
        median_price = (sorted_prices[num_prices // 2 - 1] + sorted_prices[num_prices // 2]) / 2
    print(f"Median Price for Paid Courses: {median_price}")
else:
    print("No valid price data found for Paid courses.")

# Closing the connection
conn.close()
mydb.close()
```

9. Conclusion

Key Findings

1) **Price Patterns:** Most courses are priced under $50, indicating a preference for affordable content.Trends in course pricing and discounts.

2) **Discount Effect:** Courses with higher discounts attract more enrollments, emphasizing the role of pricing strategies.

3) **Currency Variability:** Some courses listed in non-USD currencies may affect global accessibility.

4) **Ratings and Enrollment:** Higher-rated courses tend to have more enrollments, highlighting the importance of quality.

5) **Popularity Distribution:** Courses on technology, programming, and data science dominate enrollments.

Recommendations for Udemy Instructors and Organizations

1) **Quality Improvement:** Prioritize maintaining high course ratings by focusing on user feedback.

2) **Competitive Pricing:** Offer courses under $50 or provide substantial discounts to attract price-sensitive learners.

3) **Global Standardization:** Use USD as the default currency for international appeal.

4) **Marketing Strategies:** Highlight discounted courses prominently in promotions to boost enrolments.

5) **Content Focus:** Invest in creating more courses related to trending topics like technology and data science.

---

10. Appendix

Code Snippets

```
# Project Introduction
 10. -   This project aims to apply data analysis techniques to a real-world dataset (Udemy
Courses)obtained from Kaggle. Students will explore trends, identify insights, and potentially make
recommendations for instructors or the Udemy platform itself
 11.
 12. # Importing Libraries
 13. """
 14.
 15. import numpy as np
 16. import pandas as pd
 17. import matplotlib.pyplot as plt
 18. import seaborn as sns
 19. import mysql.connector
 20. import kagglehub
 21.
 22. """# Importing Data"""
 23.
 24. df =
pd.read_csv("D:/Courses/Data_Analysis/IMT/final_project/data/udemy_output_All_Finance__Accounting_p1_p626
.csv")
 25. df.head()
 26.
 27. """# Explanatory Data Analysis (EDA)
 28.
 29. ## Inspecting the DataFrame
 30. """
 31.
 32. df.shape
 33.
 34. df.info()
 35.
 36.
 71. ## Generating a summary for all the numerical values we have
 72. """
 73.
 74. df.describe()
 75.
 76. df.nunique() # This function calculates the number of unique values in each Column
 77.
 78. """## Dropping the unnecessary columns"""
 79.
 80. df.drop(['url', 'is_wishlisted',
 81.          'discount_price__currency', 'discount_price__price_string',
 82.          'avg_rating_recent','price_detail__price_string',
 83.          'num_published_practice_tests'], axis = 1, inplace = True)
 84.
 85. df.head()
 86.
 87. """## Renaming the Columns so it gives more insight to what data each column holds"""
 88.
 89. df.rename({"title":"Title","avg_rating": "Average rating",
 90.         "rating":"Rating",
 91.         "num_subscribers":"Subscribers", "published_time":"Time",
 92.         "num_reviews": "Num reviews",
 93.         "num_published_lectures":"Lectures",
 94.         "discount_price__amount":"Discount price",
 95.         "price_detail__amount":"Price",
 96.         "price_detail__currency": "Currency"}, axis = 1, inplace = True)
 97. df.head()
 98.
 99. """##  Checking the Data Formating"""
100.
101. df.dtypes
102.
```

```
"""## Checking for Duplicated Values"""

df.duplicated().sum()

"""## Fixing the Date Format"""

df["Time"] = pd.to_datetime(df["Time"])
df['created'] = pd.to_datetime(df['created'])
df.head()

"""### Convert datetime columns to string format compatible with MySQL

"""

df['created'] = df['created'].dt.strftime('%Y-%m-%d %H:%M:%S')
df['Time'] = df['Time'].dt.strftime('%Y-%m-%d %H:%M:%S')
df.head()

"""# Fixing the conversion rate for the price

### Convert currency to USD for consistency in analysis.

"""

conversion_rate = 0.0013  # Example conversion rate (1 Rupee = 0.012 USD)
df['Price'] = df['Price'] * conversion_rate
df['Discount price'] = df['Discount price'] * conversion_rate
df.head()

"""### replacing the currency with USD"""

df['Currency'].replace("INR", "USD", inplace = True)
df.head()

"""## Adding a discount percentage column"""

df['Discount Percentage'] = ((df['Price'] - df['Discount price']) / df['Price']) * 100
df['Discount Percentage'] = df['Discount Percentage'].round(2).astype(str) + "%"
df.head()

"""# Checking for null values"""

df.isna().sum()

"""# Fixing the missing values

## Fixing the Prices
"""

for column in ['Price','Discount price']:
    df.fillna({column : df[column].median()}, inplace = True)
df.isna().sum()

"""## Fixing the missing currency"""

df.fillna({"Currency":"Unknown"}, inplace = True) # Changing all the null values in price currency
column to 'unknown'
print(df['Currency'].isna().sum())

df['Currency'].unique()

"""# Typo correction"""

df['Title'] = df['Title'].str.lower()   # Converting the title column to lower case
df['Title'] = df['Title'].str.strip().replace(r'\s+', ' ', regex=True)      # Removing extra spaces
df['Title'] = df['Title'].str.replace(r'[^\w\s]', '', regex=True)           # Removing special
characters
```

```
"""# Categorizing the Courses"""

def categorize_title(title):
    title_lower = title.lower()
    if 'sql' in title_lower or 'mysql' in title_lower or 'database' in title_lower:
        return 'Database'
    elif 'tableau' in title_lower or 'power bi' in title_lower or 'data viz' in title_lower:
        return 'Data Visualization'
    elif 'excel' in title_lower or 'spreadsheet' in title_lower:
        return 'Spreadsheet'
    elif any(kw in title_lower for kw in ['agile', 'scrum', 'pmp', 'project management']):
        return 'Project Management'
    elif any(kw in title_lower for kw in ['financial', 'finance', 'accounting']):
        return 'Finance'
    elif 'mba' in title_lower or 'business' in title_lower or 'enterprise' in title_lower:
        return 'Business'
    elif any(kw in title_lower for kw in ['write', 'writing', 'editorial']):
        return 'Writing'
    elif 'sale' in title_lower or 'marketing' in title_lower:
        return 'Sales/Marketing'
    elif any(kw in title_lower for kw in ['data science', 'analytics', 'machine learning']):
        return 'Data Science'
    elif 'management' in title_lower:
        return 'Management'
    elif 'leadership' in title_lower:
        return 'Leadership'
    elif 'communication' in title_lower:
        return 'Communication'
    else:
        return 'Other'

df['Category'] = df['Title'].apply(categorize_title)
df.head()

df['Category'].value_counts()

"""# Outlier detection and deletion

## Visualizing Box Plot Graph for Columns with potential outliers
"""

numerical_columns = ['Subscribers','Price','Discount price']

for column in numerical_columns:
    plt.boxplot(df[column],vert=False)
    plt.title(f'Box Plot of {column}')
    plt.ylabel('column')
    plt.grid()
    plt.show()

"""## Calculating the standard deviation and mean for each feature"""

for column in numerical_columns:
    mu = df[column].mean()
    sigma = df[column].std()
    plt.hist(df[column])
    plt.title(f'Box Plot of {column}')
    plt.ylabel('column')
    plt.axvline(mu, color="black", linestyle="solid", label=f"Mean = {mu:.2f}")
    plt.axvline(mu + 3 * sigma, color='red', linestyle="dashed", label=f"+{3} Std Dev")
    plt.axvline(mu - 3 * sigma, color='red', linestyle="dashed", label=f"-{3} Std Dev")
    plt.grid()
    plt.legend()
    plt.show()

```

```
"""## Calculating each outlier info"""

outlier_info = {}   # Creating an empty dictionary with key:value pair and each value has muliple sub-values for the same column

# Looping through the numerical columns to calculate mean, standard deviation, and outliers
for column in numerical_columns:
    mean_value = df[column].mean()
    std_dev = df[column].std()

    # Finding outliers using 3 standard deviations from the mean
    outliers = df[(df[column] < mean_value - 3 * std_dev) | (df[column] > mean_value + 3 * std_dev)]

    # Storing the details in the dictionary
    outlier_info[column] = {
        'mean': round(mean_value, 2),
        'std_dev': round(std_dev, 2),
        'num_outliers': len(outliers),
        'outlier_values': outliers[column].values.tolist()[:5]  # Show first 5 outliers
        }

# Iterating through the outlier_info dictionary and printing each column's details
for column in outlier_info:
    print(f"Outlier info for {column}:")
    print(f"  Mean: {outlier_info[column]['mean']}")
    print(f"  Standard Deviation: {outlier_info[column]['std_dev']}")
    print(f"  Number of Outliers: {outlier_info[column]['num_outliers']}")
    print(f"  Sample Outliers: {outlier_info[column]['outlier_values']}")
    print("-" * 50)

"""# Exporting the Cleaned data to a CSV file"""

df.to_csv('cleaned_data.csv', index=False)

print("Data exported successfully as 'cleaned_data.csv'")
```

```
"""# MySQL Implementation

## Creating a connection creation function
"""

def create_connection():
    """Create and return a MySQL database connection and cursor."""

    # Establish the connection
    mydb = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database=""        # Need to fill this path for the intended database
    )

    # Create a cursor
    conn = mydb.cursor()

    print("Connection to the database was successful.")
    print("≡" * 50)
    return mydb, conn

"""## Connecting the Database"""

mydb, conn = create_connection()

"""## Database code"""

file_path = "cleaned_data.csv"

data = pd.read_csv(file_path)
data.columns

# Re-connecting to the database
mydb, conn = create_connection()

# Dropping the existing database if it exists and creating a new one
conn.execute("DROP DATABASE IF EXISTS Course_db")
conn.execute("CREATE DATABASE IF NOT EXISTS Course_db")
conn.execute("USE Course_db")

# Creating the table with the specified column names
create_sql = """
CREATE TABLE IF NOT EXISTS courses_table (
    id INT PRIMARY KEY,
    Title TEXT,
    is_paid BOOLEAN,
    Subscribers INT,
    `Average rating` FLOAT,
    Rating FLOAT,
    `Num reviews` INT,
    Lectures INT,
    created DATETIME,
    Time DATETIME,
    `Discount price` FLOAT,
    Price FLOAT,
    Currency TEXT,
    `Discount Percentage` FLOAT,
    Category TEXT
);
"""
conn.execute(create_sql)

conn.close()
```

```
 """## Making the max allowed packet size larger so the process of data population becomes easier"""

# Seting the max_allowed_packet size to a larger value
mydb, conn = create_connection()
conn.execute("SET GLOBAL max_allowed_packet=64*1024*1024")  # Set to 64MB

"""## Populating the Database with the data in our exported CSV file"""

# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Inserting data into the MySQL table
insert_sql = """
INSERT INTO courses_table (id, Title, is_paid, Subscribers, `Average rating`, Rating, `Num reviews`,
                                    Lectures, created, Time, `Discount price`, Price, Currency,
`Discount Percentage`, Category)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
"""

data_values = [
    (
        row['id'], row['Title'], row['is_paid'], row['Subscribers'], row['Average rating'],
        row['Rating'], row['Num reviews'], row['Lectures'],
        row['created'], row['Time'], row['Discount price'], row['Price'],
        row['Currency'], row['Discount Percentage'], row['Category']
    )
    for _, row in df.iterrows()
]

# Executing batch insert
conn.executemany(insert_sql, data_values)
mydb.commit()

print("Data successfully saved to the MySQL database.")

# Closing the connection
conn.close()
mydb.close()

"""# Checking the Columns in order to Ensure consistency"""

# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

conn.execute("DESCRIBE courses_table;")
columns = conn.fetchall()
print("Table columns:", columns)
```

```
"""# MySQL Queries
385.
386. ## Sort and identify top courses by:
387.
388. ### Rating
389. """
390.
391. # Re-connecting to the database
392. mydb, conn = create_connection()
393. conn.execute("USE Course_db")
394.
395. # Sorting by Rating (Top 10 Courses)
396. Sort_by_Rating = """
397. SELECT Title, `Average rating`
398. FROM courses_table
399. ORDER BY `Average rating` DESC
400. LIMIT 10;
401. """
402.
403. # Execute the query
404. conn.execute(Sort_by_Rating)
405.
406. # Fetch the results
407. results = conn.fetchall()
408.
409. # Check if any results are found and print them
410. if results:
411.     print("Top 10 Courses by Rating:")
412.     print("-"*30)
413.     for row in results:
414.         print(row)
415.         print("-"*70)
416. else:
417.     print("No results found.")
418.
419. """### Number of subscribers (if available)
420.
421. """
422.
423. # Re-connecting to the database
424. mydb, conn = create_connection()
425. conn.execute("USE Course_db")
426.
427. # Sorting by Subscribers (Top 10 Courses)
428. Sort_by_Subscribers = """
429. SELECT Title, Subscribers
430. FROM courses_table
431. ORDER BY Subscribers DESC
432. LIMIT 10;
433. """
434.
435. # Execute the query
436. conn.execute(Sort_by_Subscribers)
437.
438. # Fetch the results
439. results = conn.fetchall()
440.
441. # Check if any results are found and print them
442. if results:
443.     print("Top 10 Courses by Subscribers:")
444.     print("-"*30)
445.     for row in results:
446.         print(row)
447.         print("-"*70)
448. else:
449.     print("No results found.")
```

```
"""### Discount percentage (created during preprocessing)"""

# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Sorting by Discount Percentage (Top 10 Courses)
Sort_by_Discount_Percentage = """
SELECT Title, `Discount Percentage`
FROM courses_table
ORDER BY `Discount Percentage` DESC
LIMIT 10;
"""

# Execute the query
conn.execute(Sort_by_Discount_Percentage)

# Fetch the results
results = conn.fetchall()

# Check if any results are found and print them
if results:
    print("Top 10 Courses by Discount Percentage:")
    print("-"*40)
    for row in results:
        print(row)
        print("-"*70)
else:
    print("No results found.")

"""## Identify the most expensive courses

"""

# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Sorting by Discount Percentage (Top 10 Courses)
Sort_by_most_expensive = """
SELECT Title, price
FROM courses_table
ORDER BY price DESC
LIMIT 10;
"""

# Execute the query
conn.execute(Sort_by_most_expensive)

# Fetch the results
results = conn.fetchall()

# Check if any results are found and print them
if results:
    print("Top 10 Most Expensive Courses in USD: ")
    print("-"*40)
    for row in results:
        print(row)
        print("-"*70)
else:
    print("No results found.")

```

```
"""## Group the data by category and calculate:
514.
515. ### Number of courses in each category
516. """
517.
518. # Re-connecting to the database
519. mydb, conn = create_connection()
520. conn.execute("USE Course_db")
521.
522. # Calculating the number of courses per price category
523. Sort_by_price_category = """
524. SELECT Category, COUNT(*) AS num_courses
525. FROM courses_table
526. GROUP BY Category;
527. """
528.
529. # Execute the query
530. conn.execute(Sort_by_price_category)
531.
532. # Fetch the results
533. results = conn.fetchall()
534.
535. # Check if any results are found and print them
536. if results:
537.     print("Count of courses for each Price Category: ")
538.     print("-"*40)
539.     for row in results:
540.         print(row)
541.         print("-"*70)
542. else:
543.     print("No results found.")
544.
545. """### Average number of reviews per category"""
546.
547. # Re-connecting to the database
548. mydb, conn = create_connection()
549. conn.execute("USE Course_db")
550.
551. # Calculating the average number of reviews per Price category
552. Sort_by_reviews_category = """
553. SELECT Category, AVG(`Num reviews`) AS avg_reviews
554. FROM courses_table
555. GROUP BY Category;
556. """
557.
558. # Execute the query
559. conn.execute(Sort_by_reviews_category)
560.
561. # Fetch the results
562. results = conn.fetchall()
563.
564. # Check if any results are found and print them
565. if results:
566.     print("Average Number of Reviews per Price Category: ")
567.     print("-"*50)
568.     for row in results:
569.         print(row)
570.         print("-"*70)
571. else:
572.     print("No results found.")
573.
```

```
"""## Analyze discounts:

### Calculate average discount percentage offered per category
"""

# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Categorizing by Price Range
Sort_by_avg_discount_percentage = """
SELECT Category, AVG(`Discount Percentage`) AS avg_discount_percentage
FROM courses_table
GROUP BY Category;
"""
# Execute the query
conn.execute(Sort_by_avg_discount_percentage)

# Fetch the results
results = conn.fetchall()

# Check if any results are found and print them
if results:
    print("Average Discount Percentage Offered per Price Category: ")
    print("-"*50)
    for row in results:
        print(row)
        print("-"*70)
else:
    print("No results found.")

"""## Analyze price by paid/free status (if available):

### Calculate the median price for courses in each category.
"""
# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Categorizing paid and free courses based on 'is_paid' column
checking_for_paid_or_free = """
SELECT
    CASE
        WHEN is_paid = 0 THEN 'Free'
        WHEN is_paid = 1 THEN 'Paid'
    END AS Status,
    COUNT(*) AS num_courses
FROM courses_table
GROUP BY Status;
"""
# Execute the query
conn.execute(checking_for_paid_or_free)

# Fetch the results
results = conn.fetchall()

# Check if any results are found and print them
if results:
    print("Paid / Free Category count: ")
    print("-"*50)
    for row in results:
        print(row)
        print("-"*70)
else:
    print("No results found.")

# Closing the connection
conn.close()
mydb.close()
```

```
"""As the Paid courses are the only ones I can actually calculate their median price value, since Free courses are free so their median would be zero. I will only calculate for the paid courses"""

# Re-connecting to the database
mydb, conn = create_connection()
conn.execute("USE Course_db")

# Categorizing paid and free courses based on 'is_paid' column
calculate_median = """
SELECT
    price
FROM courses_table
WHERE is_paid = 1 AND price IS NOT NULL
ORDER BY price;
"""

# Execute the query
conn.execute(calculate_median)

# Fetch all the prices
prices = conn.fetchall()

# Calculate median
num_prices = len(prices)
if num_prices > 0:
    sorted_prices = [price[0] for price in prices]
    if num_prices % 2 == 1:  # Odd number of prices
        median_price = sorted_prices[num_prices // 2]
    else:  # Even number of prices
        median_price = (sorted_prices[num_prices // 2 - 1] + sorted_prices[num_prices // 2]) / 2
    print(f"Median Price for Paid Courses: {median_price}")
else:
    print("No valid price data found for Paid courses.")

# Closing the connection
conn.close()
mydb.close()

```

References

1) **Kaggle Dataset**

"Finance & Accounting Courses - Udemy (13K+ course) - Kaggle." https://www.kaggle.com/datasets/jilkothari/finance-accounting-courses-udemy-13k-course.

2) **Libraries and Tools used.**

1) Visual Studio Code Application
2) Python Programming Language
3) Jupyter Notebook
4) Numpy Library
5) Pandas Library
6) Matplotlib Library
7) Seaborn Library
8) mysql.connector Library
9) Tableau Dashboard