WS21/22

Schriftliche Ausarbeitung Industrielle Softwareentwicklung

Gruppe 7

Vorgelegt von:

Ahmed Zoufal (1330834)

Clement Inparaj(1351363)

Mimoun Zaouzaou(1351237)

Imad Akhouaji (1264311)

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Problemstatement

In der heutigen digitalen Wirtschaft sehen sich Unternehmen mit der Herausforderung konfrontiert, große Mengen an Verkaufsdaten effizient zu nutzen, um fundierte Geschäftsentscheidungen zu treffen. Unternehmen wie Amazon, generieren täglich enorme Mengen an Verkaufsdaten. Diese Daten enthalten wertvolle Informationen über das Kaufverhalten der Kunden, Markttrends und die Leistung verschiedener Produkte. Die Hauptprobleme umfassen:

- **Datenflut**: Große Mengen an Verkaufsdaten erschweren die Identifikation relevanter Informationen.
- **Datenqualität**: Ungenaue, unvollständige oder inkonsistente Daten beeinträchtigen die Analyse.
- **Manuelle Verarbeitung**: Viele Unternehmen setzen noch auf nicht skalierbare, fehleranfällige Methoden wie Excel.
- **Mangelnde Visualisierung**: Rohdaten sind schwer interpretierbar, was datenbasierte Entscheidungen erschwert.
- **Fehlende Automatisierung**: Ohne ETL-Prozesse (Extract, Transform, Load) ist die Datenanalyse ineffizient.

Diese Herausforderungen führen dazu, dass wertvolle Erkenntnisse aus den Verkaufsdaten nicht effektiv genutzt werden können, wodurch Potenziale für Optimierung und Umsatzsteigerung ungenutzt bleiben.

1.1 Ziel des Projekts

Das Ziel des vorliegenden Projekts besteht in der Automatisierung der Verarbeitung und Analyse von Verkaufsdaten sowie deren visuelle Darstellung, um datenbasierte Entscheidungen effizient zu unterstützen. Zu diesem Zweck ist die Entwicklung einer ETL-Pipeline (Extract, Transform, Load) geplant, die Verkaufsdaten aus einer externen Quelle (Kaggle) extrahiert, bereinigt und in eine SQLite-Datenbank speichert. Die bereinigten Daten werden anschließend in einem interaktiven Dashboard visualisiert, das unter Verwendung von Dash und Plotly erstellt wird.

Ein wesentlicher Bestandteil des hier dargelegten Ansatzes ist die Automatisierung der Datenverarbeitung. Ziel ist es, manuelle und fehleranfällige Methoden, wie beispielsweise die Verwendung von Excel-Tabellen, zu ersetzen. Daher soll eine Funktion

implementiert werden, die Verkaufsdaten automatisch von Kaggle herunterlädt, mit Pandas einliest und bereinigt. In diesem Schritt sollen fehlerhafte oder fehlende Werte korrigiert, numerische Spalten konvertiert und einheitliche Formate angewendet werden.

Um die Datenqualität zu erhöhen, ist eine standardisierte Transformation der Daten geplant. Im Rahmen dessen werden Mengen- und Umsatzwerte in numerische Formate überführt, fehlende Werte durch Standardwerte ersetzt und eine neue Spalte für den Gesamtumsatz (Total Revenue) hinzugefügt. Nach erfolgter Bereinigung ist ein Laden der Daten in eine SQLite-Datenbank beabsichtigt, um so eine effiziente Abfrage und weitere Verarbeitung zu ermöglichen.

Ein weiteres Ziel ist die interaktive Visualisierung der Verkaufsdaten, um Trends, Umsatzentwicklungen und Muster verständlich darzustellen. Dazu ist die Entwicklung eines Dashboards mit Dash und Plotly geplant. Die Nutzer sollen den Gesamtumsatz, die Anzahl der Bestellungen und den durchschnittlichen Bestellwert analysieren können. Die Anpassungsfähigkeit der Anzeige der Daten soll durch Dropdown-Menüs und Datumsauswahl-Filter gewährleistet werden. Um eine effiziente Entscheidungsfindung zu ermöglichen, wollen wir das Dashboard um detaillierte Grafiken wie Trendanalysen, Umsatzverteilungen und kategorienbasierte Umsatzvergleiche erweitern

Ein entscheidender Aspekt des vorliegenden Projekts ist die Skalierbarkeit der Lösung. Durch den Einsatz einer Datenbankstruktur und modularer Funktionen soll sichergestellt werden, dass das System problemlos erweitert werden kann, um größere Datenmengen zu verarbeiten oder weitere Datenquellen einzubinden. Dies soll garantieren, dass die Anwendung auch mit wachsendem Datenvolumen effizient bleibt.

Das von uns verfolgte Ziel besteht in der Entwicklung einer vollautomatischen Verarbeitung, Bereinigung und Visualisierung von Verkaufsdaten. Diese soll Unternehmen dabei unterstützen, wertvolle Zeit und Ressourcen zu sparen und gleichzeitig präzisere Entscheidungen auf Basis fundierter Datenanalysen zu treffen.

2 Arbeit im Projektteam

2.1 Methode

Um eine effiziente Zusammenarbeit zu gewährleisten, wird eine strukturierte und agile Vorgehensweise angewendet, bei der an den Prinzipien der iterativen Entwicklung orientiert wird. Dadurch können einzelne Bestandteile schrittweise realisiert, getestet und verbessert werden.
Zunächst wurde eine technische Planung durchgeführt, um die Anforderungen zu definieren und die Architektur der Lösung festzulegen. Im Anschluss erfolgte eine Unterteilung des Projekts in verschiedene Aufgabenbereiche, die parallel oder sukzessiv umgesetzt werden. Mittels regelmäßiger Team-Meetings wird der Fortschritt kontrolliert, Herausforderungen frühzeitig identifiziert und die Arbeit flexibel angepasst.
Durch diese methodische Herangehensweise kann eine effiziente und strukturierte Umsetzung des Projekts sichergestellt werden, wobei die Berücksichtigung aller relevanten Aspekte gewährleistet ist.

2.2 Organisation

Die Organisation des Projekts erfolgt auf Grundlage einer präzise definierten Aufgaben- und Rollenverteilung innerhalb des Teams. Die Hauptaufgaben sind in vier zentrale Bereiche unterteilt: Die Datenverarbeitung (ETL-Pipeline) befasst sich mit der Extraktion, Bereinigung und Transformation der Verkaufsdaten, während das Datenbankmanagement die Speicherung und Abfrage der Daten in einer SQLite-Datenbank übernimmt. Darüber hinaus wird die Entwicklung des Dashboards vorangetrieben, bei der eine interaktive Visualisierung mit Dash und Plotly realisiert wird. Schließlich wird auch die Dokumentation und Qualitätssicherung durchgeführt, die den Fortschritt strukturiert festhält und durch Tests eine zuverlässige Funktionalität sicherstellt.

2.3 Ablauf

Der Projektablauf ist in mehrere Phasen unterteilt, die systematisch aufeinander aufbauen. In der Planungsphase wurde eine Analyse der Anforderungen durchgeführt, es wurden Ziele definiert und die technische Architektur festgelegt. Darauf folgt die Implementierungsphase, in der die einzelnen Komponenten umgesetzt werden:

1. **Datenextraktion:** Herunterladen und Einlesen der Verkaufsdaten aus einer externen Quelle.

2. **Datenbereinigung** und -transformation: Fehlerhafte Werte korrigieren, Daten standardisieren und eine konsistente Struktur schaffen.
3. **Datenbankintegration**: Speicherung der bereinigten Daten in einer SQLite-Datenbank.
4. **Dashboard-Entwicklung**: Erstellung einer interaktiven Visualisierung, um Trends und Kennzahlen darzustellen.
5. **Tests und Validierung:** Überprüfung der Datenqualität und Funktionalität der Anwendung.
6. **Dokumentation und Abschluss**: Zusammenfassung der Projektergebnisse und Vorbereitung der finalen Präsentation.

Durch diese klare Struktur stellen wir sicher, dass alle Schritte systematisch abgearbeitet werden und das Projekt effizient zum Ziel geführt wird.

3 Architektur

Unsere mehrschichtige Architektur für das Projekt trennt klar zwischen Datenverarbeitung, Speicherung und Visualisierung. So arbeitet das System nicht nur effizient, sondern bleibt auch flexibel erweiterbar.

Die ETL-Pipeline ist die erste Schicht. Sie extrahiert die Verkaufsdaten aus einer externen Quelle, bereinigt sie und überführt sie in ein einheitliches Format. Die Transformation erfolgt in einer SQLite-Datenbank, was eine strukturierte und performante Abfrage ermöglicht.

Unser Dashboard, das mit Dash und Plotly entwickelt wird, greift direkt auf diese Datenbank zu. Es stellt die Daten visuell dar und ermöglicht Nutzern, verschiedene Kennzahlen interaktiv zu analysieren. Durch eine dynamische Datenabfrage lassen sich Filter und Aktualisierungen effizient umsetzen. Unser System ist modular aufgebaut. Dadurch können die einzelnen Komponenten unabhängig voneinander weiterentwickelt oder erweitert werden. Das garantiert die Flexibilität der Architektur und ermöglicht die spätere Ergänzung um zusätzliche Datenquellen oder Analysefunktionen.

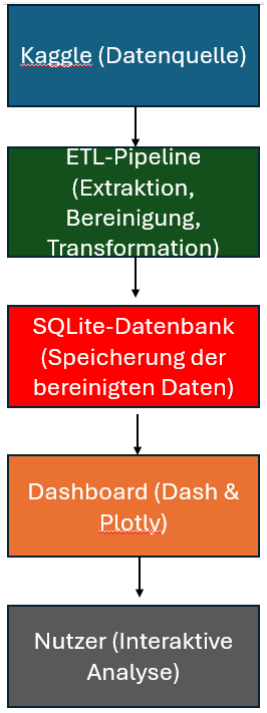


Abbildung 1: Architektur des Systems – Übersicht der Datenverarbeitung vom Import bis zur Visualisierung.

4 ETL-Pipeline

4.1 Einführung

In diesem Projekt spielt die ETL-Implementierung (Extract, Transform, Load) eine zentrale Rolle. Sie bildet die Grundlage für eine strukturierte und automatisierte Verarbeitung der Verkaufsdaten. Der ETL-Prozess umfasst drei wesentliche Schritte: Zunächst werden die Rohdaten aus einer externen Quelle, in diesem Fall Kaggle, extrahiert und in einem geeigneten Format eingelesen. Anschließend folgt die Transformation. Hier werden die Daten bereinigt, standardisiert und durch zusätzliche Berechnungen ergänzt. Dazu gehören die Korrektur fehlerhafter oder fehlender Werte, die Umwandlung von Spalten in die richtigen Datentypen sowie die Berechnung neuer Felder, beispielsweise der Gesamtumsatz pro Bestellung. Abschließend werden die transformierten Daten in einer SQLite-Datenbank gespeichert. So ist eine zuverlässige Abfrage und Weiterverarbeitung im Dashboard möglich. Die Implementierung einer ETL-Pipeline ist notwendig, da die Rohdaten oft unstrukturierte oder inkonsistente Formate enthalten, die eine direkte Analyse erschweren. Fehlende Werte, Duplikate oder uneinheitliche

Datentypen können die Qualität der Analysen beeinträchtigen. Durch den ETL-Prozess wird sichergestellt, dass die Daten in einem einheitlichen und bereinigten Zustand vorliegen, sodass sie zuverlässig für die Visualisierung und Analyse im Dashboard genutzt werden können.
Die Automatisierung des gesamten Prozesses ist ein entscheidender Faktor. Ohne ETL wäre der manuelle Download, die Bereinigung und die Übertragung der Daten in eine Datenbank oder ein anderes Speichermedium notwendig. Dies wäre nicht nur zeitaufwendig, sondern auch fehleranfällig. Die implementierte Lösung gewährleistet eine vollständige Automatisierung dieses Ablaufs und stellt so eine effiziente und reproduzierbare Datenverarbeitung sicher.
Die Speicherung der bereinigten Daten in einer Datenbank statt in einer einfachen CSV-Datei ist die beste Lösung für eine strukturierte und performante Datenverarbeitung. Das Dashboard kann die Daten direkt aus der Datenbank abrufen. Dadurch sind interaktive Analysen und Visualisierungen in Echtzeit möglich. Mit der ETL-Pipeline haben wir die beste Lösung für skalierbare und zuverlässige Datenverarbeitung. Sie verbessert die Datenqualität und unterstützt eine effiziente Analyse und Visualisierung der Verkaufszahlen.

4.2 Extract

Die Datenextraktion bildet den ersten Schritt der ETL-Pipeline und stellt sicher, dass die Verkaufsdaten aus einer externen Quelle in unser System geladen werden. Da die Daten nicht direkt in einer Datenbank gespeichert sind, müssen sie zunächst heruntergeladen und in ein geeignetes Format umgewandelt werden. Dies geschieht in unserem Fall durch den Zugriff auf das Kaggle-Dataset und das anschließende Speichern der Daten als Pandas DataFrame. Dabei wird sichergestellt, dass die Datei korrekt geladen wird und für die weiteren Schritte der Verarbeitung zur Verfügung steht.
Im folgenden Abschnitt wird die genaue Implementierung der Datenextraktions-Funktion erläutert. Dabei wird beschrieben, wie das Dataset aus Kaggle abgerufen, gespeichert und für die weiteren Verarbeitungsschritte vorbereitet wird.
Die Datenextraktion in diesem Projekt wird in der Funktion extract_data() durchgeführt. Diese Funktion ist dafür verantwortlich, die Verkaufsdaten aus Kaggle herunterzuladen und in ein Pandas DataFrame zu laden.

Schritt 1: Download des Datasets von Kaggle

Der folgende Code zeigt, wie das Dataset aus Kaggle bezogen wird:

```
def extract_data():
    """
    Extract Amazon Sales Data from Kaggle.
    Downloads the dataset and loads it into a Pandas DataFrame.
    """
    print(" Downloading 'Unlock Profits with E-Commerce Sales Data' from Kaggle...")

    # Dataset Name from Kaggle
    dataset_name = "thedevastator/unlock-profits-with-e-commerce-sales-data"

    # Download dataset from KaggleHub
    path = kagglehub.dataset_download(dataset_name)
    file_path = os.path.join(path, "Amazon Sale Report.csv")  # Ensure correct filename
```

Abbildung 2: Funktion `extract_data()`, die Verkaufsdaten von Kaggle herunterlädt und als Pandas DataFrame lädt.

Hier wird das Dataset von Kaggle mit Hilfe von kagglehub.dataset_download(dataset_name) heruntergeladen. Anschließend wird der Pfad zur CSV-Datei festgelegt, sodass der Code später darauf zugreifen kann.

Schritt 2: Überprüfung, ob die Datei existiert

```
# Check if file exists
if not os.path.exists(file_path):
    print(f" Dataset file NOT found: {file_path}")
    return None

print(f" Dataset file found: ")
```

Abbildung 3: Überprüfung der Existenz der Dataset-Datei, mittels if-Anwesung

Bevor die Datei geladen wird, überprüft der Code, ob die CSV-Datei überhaupt existiert. Falls nicht, gibt das Programm eine Fehlermeldung aus und bricht ab.

Schritt 3: Laden der CSV-Datei in Pandas

```
# Load dataset into Pandas DataFrame
try:
    data = pd.read_csv(file_path, low_memory=False)
    print(" Data successfully loaded from CSV!")
    return data
except Exception as e:
    print(f" Error while reading CSV file: {e}")
    return None
```

Abbildung 4: Laden der CSV-Datei in ein Pandas DataFrame mit Fehlerbehandlung.

Falls die Datei vorhanden ist, wird sie mit pandas eingelesen (pd.read_csv()). Die Option low_memory=False wird genutzt, um Probleme mit gemischten Datentypen zu

vermeiden. Falls ein Fehler beim Laden auftritt, wird dieser abgefangen und eine entsprechende Fehlermeldung ausgegeben.

4.3 Transformation

Die Daten-Transformation stellt einen essenziellen Schritt in der ETL-Pipeline dar, um die Qualität und Konsistenz der Verkaufsdaten zu gewährleisten. Nach der Extraktion werden die Daten häufig in einem unstrukturierten oder fehlerhaften Zustand vorgefunden. In diesem Schritt werden fehlende Werte behandelt, Datenformate vereinheitlicht, Spalten umbenannt und berechnete Felder hinzugefügt, um eine aussagekräftige Analyse zu ermöglichen.Das Ziel besteht darin, die Rohdaten in eine strukturierte und bereinigte Form zu überführen, sodass sie ohne weitere Anpassungen für Visualisierungen und Analysen genutzt werden können. Eine der zentralen Anpassungen in diesem Projekt ist die Berechnung des Gesamtumsatzes (Total Revenue) durch Multiplikation der verkauften Menge mit dem Preis.
Im folgenden Abschnitt wird die Implementierung der Transformations-Logik im Detail erklärt.

Schritt 1: Spaltennamen vereinheitlichen

Der folgende Code zeigt, wie die Spaltennamen in Kleinbuchstaben umgewandelt werden:

```
# Convert all column names to lowercase for consistency
data.columns = map(str.lower, data.columns)
```

Abbildung 5: Umwandlung aller Spaltennamen in Kleinbuchstaben zur Konsistenzsicherung.

Hier werden alle Spaltennamen in Kleinbuchstaben konvertiert, um eine einheitliche Struktur zu gewährleisten. Dadurch werden mögliche Fehler durch inkonsistente Schreibweisen vermieden.

Schritt 2: Fehlende Werte ersetzen

Bevor die Daten weiterverarbeitet werden, überprüft der Code, ob fehlende Werte (NaN) vorhanden sind. Falls ja, werden sie durch 0 ersetzt, um Berechnungsfehler zu vermeiden.

```
# Handle missing values by replacing with 0
data.fillna(0, inplace=True)
```

Abbildung 6: Ersetzen fehlender Werte durch 0 zur Datenbereinigung.

Dies stellt sicher, dass alle Spalten vollständig sind und spätere Berechnungen nicht durch fehlende Werte beeinträchtigt werden.

Schritt 3: Überprüfung auf erforderliche Spalten

Der Code prüft vor der Fortsetzung der Transformation, ob die für die Berechnungen notwendigen Spalten vorhanden sind.

```
# Ensure required columns exist before proceeding
required_cols = ['qty', 'amount']
if not all(col in data.columns for col in required_cols):
    print(f" Required columns missing: {required_cols}")
    return None
```

Abbildung 7: Überprüfung, ob alle erforderlichen Spalten in den Daten vorhanden sind.

Falls die Spalten qty (Menge) oder amount (Preis) fehlen, wird die Transformation gestoppt und eine Fehlermeldung ausgegeben.

Schritt 4: Umwandlung der Spalten in numerische Werte

Wenn "qty" und "amount" fälschlicherweise als Zeichenketten (Strings) gespeichert sind, werden sie in numerische Werte (int oder float) umgewandelt.

```
# Convert 'qty' and 'amount' to numeric, replacing errors with 0
data['qty'] = pd.to_numeric(data['qty'], errors='coerce').fillna(0)
data['amount'] = pd.to_numeric(data['amount'], errors='coerce').fillna(0)
```

Abbildung 8: Umwandlung der Spalten qty und amount in numerische Werte, Ersetzen fehlerhafter Einträge durch 0.

Ungültige Zeichen in diesen Spalten werden automatisch in 0 umgewandelt. So werden Berechnungsfehler verhindert.

Schritt 5: Berechnung der neuen Spalte „Total Revenue“

Der Gesamtumsatz (Total Revenue) wird als Produkt der Menge (qty) und des Preises (amount) berechnet.

```
# Create a new column 'Total Revenue'
data['total revenue'] = data['qty'] * data['amount']
```

Abbildung 9: Berechnung des Gesamtumsatzes durch Multiplikation von qty und amount.

Diese neue Spalte wird später für Umsatzanalysen im Dashboard verwendet.

Nach Abschluss der beschriebenen Schritte sind die Daten bereinigt, strukturiert und für die Speicherung vorbereitet. Im Anschluss werden die transformierten Daten in die SQLite-Datenbank geladen und können für Visualisierungen und Analysen genutzt werden.

4.4 Load

Nach der Extraktion und Transformation der Verkaufsdaten erfolgt die Speicherung für eine effiziente Analyse und spätere Abfragen. Zu diesem Zweck wird eine SQLite-Datenbank genutzt, die eine strukturierte Speicherung ermöglicht und schnelle Abfragen durch das Dashboard unterstützt.

Die Speicherung in einer Datenbank anstelle einer einfachen CSV-Datei ermöglicht eine flexiblere Verarbeitung und effizientere Abfragen der Daten. In diesem Schritt werden die bereinigten und umgewandelten Daten in eine SQLite-Tabelle geschrieben, sodass sie jederzeit für weitere Analysen zur Verfügung stehen.

Im folgenden Abschnitt wird die Implementierung der Speicherlogik erläutert, die sicherstellt, dass die transformierten Verkaufsdaten korrekt in die Datenbank übernommen werden.

Schritt 1: Vorbereitung der Datenbankverbindung

Der folgende Code zeigt, wie eine Verbindung zur SQLite-Datenbank hergestellt wird.

```
import sqlite3
from datetime import datetime

# Database Configuration
DB_FILE = "amazon_sales.db"
TABLE_NAME = "sales_data"
```

Abbildung 10: Import relevanter Bibliotheken und Konfiguration der SQLite-Datenbank.

Hier definieren wir die Datenbankdatei (amazon_sales.db). Sollte diese Datei nicht existieren, wird sie beim ersten Speichern automatisch erstellt.

Schritt 2: Laden der bereinigten Daten in die Datenbank

Bevor die Daten gespeichert werden, überprüft der Code, ob das DataFrame existiert. Falls keine Daten vorliegen, wird der Ladeprozess abgebrochen.

```
def load_data(data):
    """
    Load the cleaned sales data into an SQLite database.
    """
    if data is None:
        print(" No transformed data available to load into the database.")
        return
```

Abbildung 11: Funktion load_data(), die bereinigte Verkaufsdaten in eine SQLite-Datenbank lädt.

Im Falle der Ermittlung eines None-Wertes im data-DataFrame wird seitens des Programms eine Fehlermeldung ausgegeben und der Ladeprozess folglich beendet.

Schritt 3: Speichern der Daten in SQLite mit Pandas

Falls die Daten vorhanden sind, werden sie mit to_sql() in die Datenbank geschrieben.

```
print(" Storing transformed sales data into SQLite database...")

try:
    with sqlite3.connect(DB_FILE) as conn:
        data.to_sql(TABLE_NAME, conn, if_exists='replace', index=False)
        print(f" Data successfully stored in {DB_FILE}, table: {TABLE_NAME}")
except Exception as e:
    print(f" Error while inserting data into database: {e}")
```

Abbildung 12: Speicherung der transformierten Verkaufsdaten in die SQLite-Datenbank mit Fehlerbehandlung.

Hier wird die SQLite-Verbindung hergestellt, und die Daten werden mit to_sql() in die Datenbank geschrieben.
if_exists='replace' sorgt dafür, dass die alte Tabelle überschrieben wird.
index=False stellt sicher, dass der Pandas-Index nicht mit gespeichert wird.
Falls ein Fehler auftritt, wird dieser abgefangen und eine Fehlermeldung ausgegeben.
Nach diesen Schritten sind die Daten erfolgreich in der SQLite-Datenbank gespeichert und können später für Analysen und Visualisierungen ausgelesen werden.

5 Dashboard

Das Amazon Sales Dashboard stellt eine interaktive Webanwendung dar, welche die Analyse von Verkaufsdaten aus einer SQLite-Datenbank ermöglicht. Zu den Funktionalitäten der Anwendung zählen das Laden von Daten, die Darstellung dieser in verschiedenen Diagrammen sowie das Setzen von Filtern zur Durchführung gezielter Analysen.

5.1 Datenverarbeitung und Filterung

Das Dashboard lädt die Daten mit der Funktion load_data(). Diese extrahiert die Verkaufsinformationen aus der SQLite-Datenbank (amazon_sales.db). Existiert eine Datumsspalte (date), wird diese in das Datetime-Format umgewandelt. So wird eine präzise Filterung ermöglicht.

```python
def load_data():
    """
    Load cleaned sales data from SQLite database into a Pandas DataFrame.
    """
    conn = sqlite3.connect(DB_FILE)
    query = f"SELECT * FROM {TABLE_NAME}"
    data = pd.read_sql_query(query, conn)
    conn.close()

    # Convert 'date' column to datetime type if it exists
    if 'date' in data.columns:
        data['date'] = pd.to_datetime(data['date'])

    return data
```

Abbildung 13: Laden der bereinigten Verkaufsdaten aus der SQLite-Datenbank in ein Pandas DataFrame.

Diese Daten sind die Grundlage für das gesamte Dashboard. Nutzer können über Dropdowns und Datumsfilter auswählen, welche Werte angezeigt werden sollen.

5.2 Dynamische Aktualisierung von Metriken

Drei zentrale Kennzahlen werden als Bootstrap-Karten dargestellt:

- **Gesamtumsatz (Total Revenue)**
- **Anzahl der Bestellungen (Total Orders)**
- **Durchschnittlicher Bestellwert (Average Order Value)**

Diese Werte werden dynamisch aktualisiert, wenn der Nutzer einen anderen Zeitraum oder eine andere Metrik wählt. Die folgende Callback-Funktion berechnet die Kennzahlen aus den gefilterten Daten:

```python
@app.callback(
    [Output('total-revenue', 'children'),
     Output('total-orders', 'children'),
     Output('avg-order', 'children')],
    [Input('column-selector', 'value'),
     Input('date-range', 'start_date'),
     Input('date-range', 'end_date')]
)
def update_summary_cards(selected_column, start_date, end_date):
    filtered_data = data.copy()
    if start_date and end_date:
        filtered_data = filtered_data[
            (filtered_data['date'] >= start_date) &
            (filtered_data['date'] <= end_date)
        ]

    total_revenue = f"${filtered_data['total revenue'].sum():,.2f}"
    total_orders = len(filtered_data)
    avg_order = f"${filtered_data['total revenue'].mean():,.2f}"
    return total_revenue, total_orders, avg_order
```

Abbildung 14: Callback-Funktion zur dynamischen Berechnung und Aktualisierung der Umsatz-, Bestell- und Durchschnittswerte im Dashboard.

In diesem Prozessschritt erfolgt zunächst die Duplikation und Filterung des ursprünglichen Datasets, sodass ausschließlich die Werte innerhalb des spezifizierten Zeitraums berücksichtigt werden. Daraufhin werden der Gesamtumsatz, die Anzahl der Bestellungen sowie der Durchschnittswert ermittelt und im Dashboard aktualisiert.

6 Visualisierung

6.1 Verkaufsverlauf über die Zeit – Sales Trend Diagramm

Abbildung 15: Liniendiagramm zur Darstellung des Umsatztrends über die Zeit.

Das Liniendiagramm (Sales Trend) zeigt klar und deutlich die Entwicklung der gewählten Metrik (Umsatz oder Verkaufsmenge) über die Zeit. Damit lassen sich saisonale Schwankungen, Wachstumstrends und Verkaufsspitzen eindeutig erkennen.
Die Daten werden exakt nach dem Datum aggregiert. So werden für jeden Tag die Gesamtwerte summiert und klar visualisiert. Die Callback-Funktion garantiert, dass das Diagramm immer über die aktuellen Daten aus der Datenbank verfügt:

```
@app.callback(
    Output('sales-trend', 'figure'),
    [Input('column-selector', 'value'),
     Input('date-range', 'start_date'),
     Input('date-range', 'end_date')]
)
def update_sales_trend(selected_column, start_date, end_date):
    filtered_data = data.copy()
    if start_date and end_date:
        filtered_data = filtered_data[
            (filtered_data['date'] >= start_date) &
            (filtered_data['date'] <= end_date)
        ]

    daily_data = filtered_data.groupby('date')[selected_column].sum().reset_index()
    return px.line(daily_data, x='date', y=selected_column, title=" Sales Trend")
```

Abbildung 16: Callback-Funktion zur Aktualisierung des Umsatztrends im Liniendiagramm basierend auf Nutzereingaben.

In diesem Diagramm zeigt der Nutzer durch Auswahl eines Zeitraums und einer Metrik, wie sich die Verkäufe im gewählten Zeitraum entwickelt haben.

6.2 Umsatzverteilung nach Kategorien – Kreisdiagramm

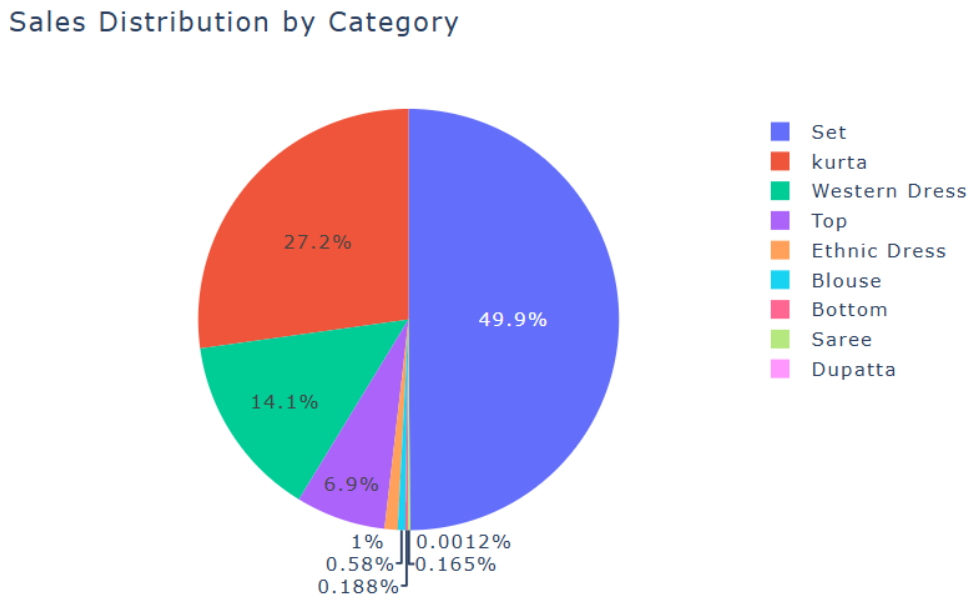


Abbildung 17: Kreisdiagramm zur Verteilung des Umsatzes nach Produktkategorien.

Das Kreisdiagramm "Sales Distribution by Category" veranschaulicht die prozentuale Verteilung des Umsatzes auf verschiedene Produktkategorien. Diese Visualisierung ermöglicht es, die Produktgruppen mit dem höchsten und niedrigsten Umsatz zu identifizieren und somit die effektivsten und weniger effektiven Produktkategorien zu ermitteln.

Die Callback-Funktion gruppiert die Verkaufsdaten nach der Kategorie und berechnet den Umsatz für jede Kategorie.

```
@app.callback(
    Output('sales-pie-chart', 'figure'),
    [Input('column-selector', 'value'),
     Input('date-range', 'start_date'),
     Input('date-range', 'end_date')]
)
def update_pie_chart(selected_column, start_date, end_date):
    filtered_data = data.copy()
    if start_date and end_date:
        filtered_data = filtered_data[
            (filtered_data['date'] >= start_date) &
            (filtered_data['date'] <= end_date)
        ]

    category_data = filtered_data.groupby('category')[selected_column].sum().reset_index()
    return px.pie(category_data, names='category', values=selected_column,
                  title="📊 Sales Distribution by Category")
```

Abbildung 18: Callback-Funktion zur Aktualisierung des Kreisdiagramms für die Umsatzverteilung nach Kategorien basierend auf Nutzereingaben.

Das vorliegende Diagramm unterstützt den Prozess der Identifizierung starker und schwacher Produktsegmente und leistet damit einen Beitrag zur Entscheidungsfindung in Bezug auf die strategische Ausrichtung der Produktpalette.

6.3 Tägliche Verkaufsverteilung – Boxplot (Daily Sales Distribution)

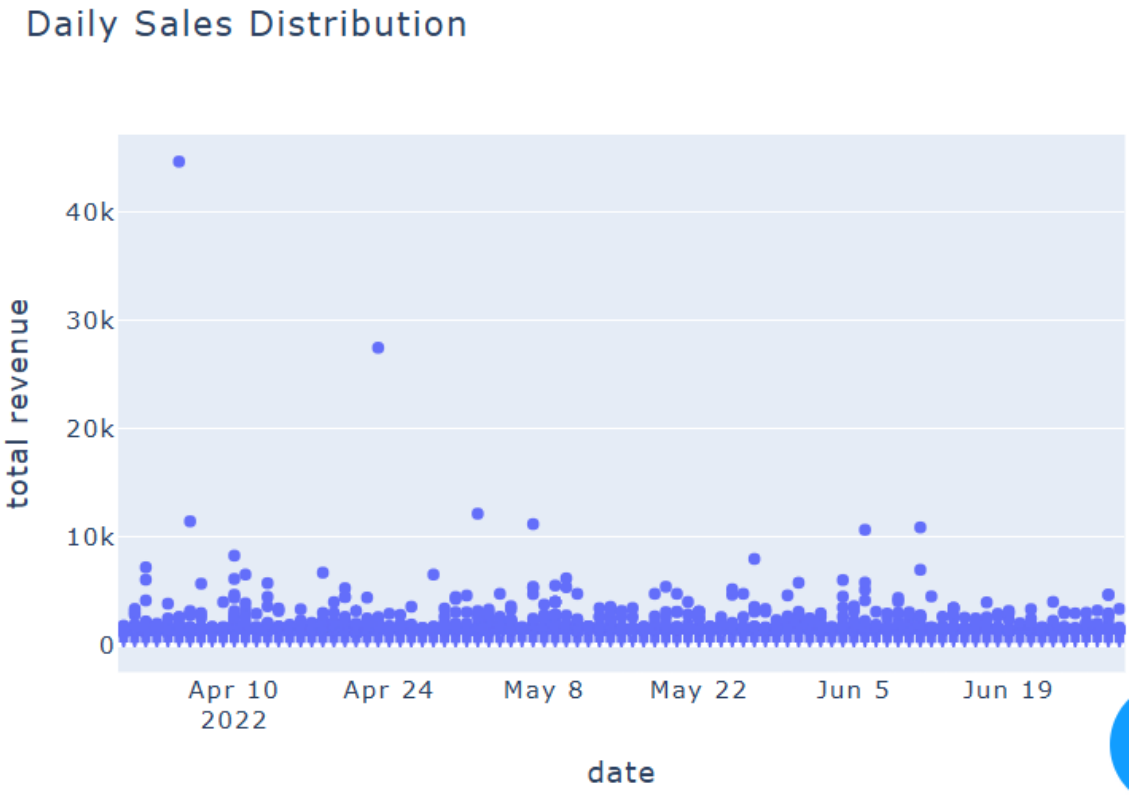


Abbildung 19: Abb. 18: Scatter-Plot zur täglichen Verkaufsverteilung mit Ausreißeranalyse.

Das Boxplot-Diagramm (Daily Sales Distribution) ist das ideale Tool, um Schwankungen und die Verteilung der Verkaufswerte pro Tag zu analysieren. Es identifiziert zuverlässig Ausreißer, ungewöhnlich hohe oder niedrige Verkäufe und liefert klare Einblicke in die allgemeinen Trends im Tagesverlauf.
Die Callback-Funktion aggregiert die Daten für die tägliche Verteilung und stellt sie als Boxplot dar.

```python
@app.callback(
    Output('daily-distribution', 'figure'),
    [Input('column-selector', 'value'),
     Input('date-range', 'start_date'),
     Input('date-range', 'end_date')]
)
def update_daily_distribution(selected_column, start_date, end_date):
    filtered_data = data.copy()
    if start_date and end_date:
        filtered_data = filtered_data[
            (filtered_data['date'] >= start_date) &
            (filtered_data['date'] <= end_date)
        ]

    return px.box(filtered_data, x='date', y=selected_column,
                  title=" Daily Sales Distribution")
```

Abbildung 20: Callback-Funktion zur Aktualisierung der täglichen Verkaufsverteilung im Boxplot basierend auf Nutzereingaben.

Das vorliegende Diagramm erweist sich als ein besonders nützliches Instrument zur Analyse regelmäßiger Verkaufsmuster sowie plötzlicher Schwankungen.
Mit dem Amazon Sales Dashboard haben Sie eine umfassende Analyse-Tool für Ihre Verkaufsdaten. Die Kombination aus Datenbankanbindung, interaktiven Filtern und dynamischen Diagrammen gibt Ihnen wertvolle Einblicke in Ihre Verkaufsleistung. Die automatische Aktualisierung aller Metriken und Diagramme ermöglicht eine präzise und flexible Datenanalyse.

7 Business Case

Das Ziel dieses Projekts besteht in der Optimierung des gesamten Prozesses der Datenverarbeitung und Analyse von Verkaufsdaten. In vielen Unternehmen erfolgt die Verarbeitung von Verkaufsdaten manuell oder mit ineffizienten Methoden, was häufig zu Fehlern, redundanten Arbeitsschritten und einer verzögerten Entscheidungsfindung führt. Durch die Automatisierung der Datenerfassung, Bereinigung und Speicherung können Unternehmen schneller und präziser auf Veränderungen am Markt reagieren. Ein essenzieller Anwendungsfall dieser Lösung liegt in der Analyse des Umsatzes sowie der Ermittlung von Trends. Unternehmen können durch die Analyse historischer Verkaufsdaten Muster erkennen, beispielsweise saisonale Schwankungen oder den Einfluss spezifischer Rabattaktionen.Diese Erkenntnisse ermöglichen eine gezielte Anpassung der Marketingstrategien, eine effiziente Verwaltung der Lagerbestände und eine Optimierung der Kundenansprache.

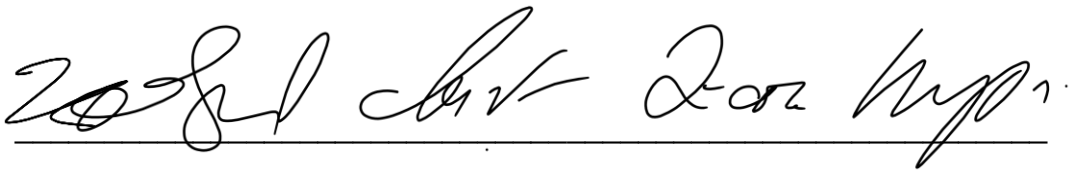
Die Skalierbarkeit des Systems stellt einen weiteren geschäftlichen Vorteil dar. Die entwickelte Lösung kann an die spezifischen Bedürfnisse eines Unternehmens angepasst werden und es besteht die Möglichkeit, sie mit zusätzlichen Funktionen zu erweitern. So können beispielsweise weitere Datenquellen, wie Social-Media-Interaktionen oder Markttrends, integriert werden, um noch detailliertere Analysen zu ermöglichen.
Zusammenfassend lässt sich festhalten, dass das Projekt zur Beschleunigung, Standardisierung und effizienteren Gestaltung des gesamten Datenverarbeitungsprozesses beiträgt. Die Reduktion operativer Kosten sowie die Steigerung der Qualität strategischer Entscheidungen auf einer soliden, datenbasierten Grundlage sind dabei wesentliche Vorteile, die sich aus der Umsetzung ergeben. Langfristig resultiert dies in einer signifikant verbesserten Unternehmensperformance.

Nachricht an den Prüfer: Alle Aufgaben und Anforderungen wurden als Team in mehreren Gruppensitzungen geplant, abgearbeitet und implementiert. Eine strikte Aufgabenteilung ist nicht erfolgt.

8 Eidesstattliche Erklärung

Hiermit versichern wir, dass wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind,sind als solche kenntlich gemacht. Die Arbeit hat in gleicher Form noch keiner anderen Prüfbehörde vorgelegen.

Frankfurt am Main, den 09.02.2025



Unterschrift