Box Plots

Estimated time needed: **45** minutes

In this lab, you will focus on the visualization of data. The dataset will be provided through an RDBMS, and you will need to use SQL queries to extract the required data.

Objectives

In this lab you will perform the following:

- Visualize the distribution of data.
- Visualize the relationship between two features.
- Visualize data composition and comparisons using box plots.

Setup: Connecting to the Database

1. Download the Database File

In [1]:
```
!wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/QR9YeprUYhOoLafzlLspAw/sur
```

```
--2025-06-18 16:19:56--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/QR9YeprU
YhOoLafzlLspAw/survey-results-public.sqlite
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-ob
ject-storage.appdomain.cloud)... 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.clou
d-object-storage.appdomain.cloud)|169.63.118.104|:443... connected.
200 OKequest sent, awaiting response...
Length: 211415040 (202M) [application/octet-stream]
Saving to: ‘survey-results-public.sqlite.4’

survey-results-publ 100%[===================>] 201.62M  54.1MB/s    in 3.7s

2025-06-18 16:20:02 (54.1 MB/s) - ‘survey-results-public.sqlite.4’ saved [211415040/211415040]
```

2. Connect to the Database

Install the needed libraries

In [2]:
```
!pip install pandas
```

```
Requirement already satisfied: pandas in /opt/conda/lib/python3.12/site-packages (2.3.0)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-packages (from panda
s) (2.3.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.12/site-packages (fr
om pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-packages (from pandas)
(2024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-packages (from panda
s) (2025.2)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-packages (from python-date
util>=2.8.2->pandas) (1.17.0)
```

In [3]:
```
!pip install matplotlib
```

```
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.12/site-packages (3.10.3)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.12/site-packages (from mat
plotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-packages (from matplot
lib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.12/site-packages (from ma
tplotlib) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/conda/lib/python3.12/site-packages (from ma
tplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-packages (from matplotl
ib) (2.3.0)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.12/site-packages (from matp
lotlib) (24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-packages (from matplotli
b) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.12/site-packages (from mat
plotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.12/site-packages (from
matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-packages (from python-date
util>=2.7->matplotlib) (1.17.0)
```

In [4]:

```python
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt

# Connect to the SQLite database
conn = sqlite3.connect('survey-results-public.sqlite')
```

In [5]:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sqlite3
import numpy as np
import functools

# Decorator to warn only once for a given message
def warn_once(func):
    warnings = set()

    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        # Check if the first argument (message) is hashable
        message_to_check = args[0] if args else None

        is_hashable = False
        if message_to_check is not None:
            try:
                hash(message_to_check)
                is_hashable = True
            except TypeError:
                is_hashable = False

        if is_hashable and message_to_check not in warnings:
            warnings.add(message_to_check)
            return func(*args, **kwargs)
        elif not is_hashable:
            # If the message is not hashable (e.g., a DataFrame), just print it directly
            return func(*args, **kwargs)
        else:
            # If hashable and already warned, do nothing (skip printing)
            pass

    return wrapper

# Apply the decorator to the print function for warnings
original_print = print
@warn_once
def print_once(*args, **kwargs):
    original_print(*args, **kwargs)

print = print_once # Override print for warnings within this script
```

```python
# --- Setup: Connect to SQLite Database ---
print("--- Setup: Connecting to SQLite Database ---")

# The PDF indicates 'survey-results-public.sqlite' is downloaded via wget.
# We will attempt to connect to this local file.
db_file = 'survey-results-public.sqlite'
conn = None # Initialize conn to None

try:
    conn = sqlite3.connect(db_file)
    print(f"Successfully connected to database: '{db_file}'")

    # Load the entire 'main' table into a pandas DataFrame for easier processing
    # This assumes 'main' is the table name as used in the PDF examples (Demo 1, 3).
    QUERY_FULL_DATA = "SELECT * FROM main"
    df = pd.read_sql_query(QUERY_FULL_DATA, conn)
    print("DataFrame 'df' loaded from 'main' table.")
    print(f"Initial DataFrame shape: {df.shape}")
    print(f"Initial DataFrame columns: {df.columns.tolist()}")

    # --- Initial Data Cleaning (Robust) ---
    # Apply robust cleaning to common numerical and categorical columns early
    # This helps prevent TypeErrors or NaN issues in later plotting steps.
    print("\n--- Initial Data Cleaning for All Lab Tasks ---")

    # Numerical columns that might have non-numeric entries or NaNs
    # IMPORTANT: Explicitly include 'JobSatPoints_6' as per PDF and your data's column names.
    numerical_cols_for_cleaning = ['CompTotal', 'YearsCodePro', 'WorkExp', 'JobSatPoints_6']
    if 'ConvertedCompYearly' in df.columns:
        numerical_cols_for_cleaning.append('ConvertedCompYearly')

    for col in numerical_cols_for_cleaning:
        if col in df.columns:
            # Attempt to convert to numeric, coercing errors to NaN
            original_null_count = df[col].isnull().sum()
            df[col] = pd.to_numeric(df[col], errors='coerce')

            # Impute NaNs if they exist after conversion
            if df[col].isnull().any():
                median_val = df[col].median()
                if pd.isna(median_val):
                    print(f"WARNING: Column '{col}' is entirely NaN after numeric conversion. Canno
                else:
                    df[col].fillna(median_val, inplace=True)
                    print(f"Cleaned '{col}': Coerced {df[col].isnull().sum() - original_null_count}
            else:
                print(f"Cleaned '{col}': No non-numeric or missing values found after conversion.")
        else:
            print(f"WARNING: Numerical column '{col}' not found in DataFrame for initial cleaning."

    # Categorical columns that might have problematic strings or NaNs
    # Include 'Employment', 'DevType', 'Country' as they are mentioned in tasks
    categorical_cols_for_cleaning = ['Age', 'Employment', 'DevType', 'Country']

    # The 'JobSat' column (generic name) might not exist, but JobSatPoints_X columns do.
    # The `JobSat` in numerical_cols_for_cleaning now takes care of `JobSatPoints_6`.
    # Remove the old `JobSat` specific categorical mapping if it's not needed/causing issues.

    for col in categorical_cols_for_cleaning:
        if col in df.columns:
            # Convert to string, strip whitespace, and replace common NaN representations
            df[col] = df[col].astype(str).str.strip()
            df[col].replace(['nan', 'NaN', 'N/A', 'None', '', ' '], np.nan, inplace=True)

            # Impute NaNs with mode if they exist
            if df[col].isnull().any():
                mode_val = df[col].mode()[0]
                df[col].fillna(mode_val, inplace=True)
                print(f"Cleaned '{col}': Imputed NaNs with mode: '{mode_val}'")
            else:
                print(f"Cleaned '{col}': No missing values found.")
        else:
            print(f"WARNING: Categorical column '{col}' not found in DataFrame for initial cleaning

    # Prepare 'Age_Numeric' for plotting where numeric age is useful
```

```
    age_numeric_mapping = {
        'Under 18 years old': 17, '18-24 years old': 21, '25-34 years old': 29,
        '35-44 years old': 39, '45-54 years old': 49, '55-64 years old': 59,
        '65 years or older': 65, 'Prefer not to say': np.nan
    }
    if 'Age' in df.columns:
        df['Age_Numeric'] = df['Age'].map(age_numeric_mapping)
        if df['Age_Numeric'].isnull().any():
            df['Age_Numeric'].fillna(df['Age_Numeric'].median(), inplace=True)
            print("Created and imputed 'Age_Numeric' column.")
    else:
        print("WARNING: 'Age' column not found, 'Age_Numeric' will not be created.")

except sqlite3.Error as e:
    print(f"ERROR: Could not connect to database or load data: {e}")
    print("Please ensure 'survey-results-public.sqlite' is in the same directory as this script.")
    # Create a dummy DataFrame if DB connection fails, to allow code to run for demonstration
    print("Creating a dummy DataFrame for demonstration purposes...")
    num_rows_dummy = 200 # Using a larger dummy dataset
    data = {
        'ResponseId': range(1, num_rows_dummy + 1),
        'CompTotal': np.random.normal(loc=100000, scale=70000, size=num_rows_dummy),
        'YearsCodePro': np.random.randint(0, 30, size=num_rows_dummy),
        'Age': np.random.choice(['18-24 years old', '25-34 years old', '35-44 years old', '45-54 ye
        'Employment': np.random.choice(['Employed, full-time', 'Employed, part-time', 'Student', 'R
        'DevType': [';'.join(np.random.choice(['Developer, back-end', 'Developer, front-end', 'Deve
        'Country': np.random.choice(['USA', 'India', 'Germany', 'UK', 'Canada', 'France'], size=num
        'JobSatPoints_6': np.random.randint(1, 6, size=num_rows_dummy), # Using JobSatPoints_6 here
        'WorkExp': np.random.randint(0, 40, size=num_rows_dummy),
        'ConvertedCompYearly': np.random.normal(loc=120000, scale=80000, size=num_rows_dummy) # Add
    }
    df = pd.DataFrame(data)
    # Ensure numerical columns are truly numeric in dummy data
    for col in ['CompTotal', 'YearsCodePro', 'JobSatPoints_6', 'WorkExp', 'ConvertedCompYearly']:
        df[col] = pd.to_numeric(df[col], errors='coerce').fillna(df[col].median())
    # Ensure Age_Numeric is created for dummy data
    age_numeric_mapping = {
        'Under 18 years old': 17, '18-24 years old': 21, '25-34 years old': 29,
        '35-44 years old': 39, '45-54 years old': 49, '55-64 years old': 59,
        '65 years or older': 65, 'Prefer not to say': np.nan
    }
    df['Age_Numeric'] = df['Age'].map(age_numeric_mapping).fillna(df['Age_Numeric'].median())
    conn = None # Set conn to None to indicate no real DB connection
```

```
--- Setup: Connecting to SQLite Database ---
Successfully connected to database: 'survey-results-public.sqlite'
DataFrame 'df' loaded from 'main' table.
Initial DataFrame shape: (65437, 114)
Initial DataFrame columns: ['ResponseId', 'MainBranch', 'Age', 'Employment', 'RemoteWork', 'Check',
'CodingActivities', 'EdLevel', 'LearnCode', 'LearnCodeOnline', 'TechDoc', 'YearsCode', 'YearsCodePr
o', 'DevType', 'OrgSize', 'PurchaseInfluence', 'BuyNewTool', 'BuildvsBuy', 'TechEndorse', 'Country',
'Currency', 'CompTotal', 'LanguageHaveWorkedWith', 'LanguageWantToWorkWith', 'LanguageAdmired', 'Dat
abaseHaveWorkedWith', 'DatabaseWantToWorkWith', 'DatabaseAdmired', 'PlatformHaveWorkedWith', 'Platfo
rmWantToWorkWith', 'PlatformAdmired', 'WebframeHaveWorkedWith', 'WebframeWantToWorkWith', 'WebframeA
dmired', 'EmbeddedHaveWorkedWith', 'EmbeddedWantToWorkWith', 'EmbeddedAdmired', 'MiscTechHaveWorkedW
ith', 'MiscTechWantToWorkWith', 'MiscTechAdmired', 'ToolsTechHaveWorkedWith', 'ToolsTechWantToWorkWi
th', 'ToolsTechAdmired', 'NEWCollabToolsHaveWorkedWith', 'NEWCollabToolsWantToWorkWith', 'NEWCollabT
oolsAdmired', 'OpSysPersonal use', 'OpSysProfessional use', 'OfficeStackAsyncHaveWorkedWith', 'Offic
eStackAsyncWantToWorkWith', 'OfficeStackAsyncAdmired', 'OfficeStackSyncHaveWorkedWith', 'OfficeStack
SyncWantToWorkWith', 'OfficeStackSyncAdmired', 'AISearchDevHaveWorkedWith', 'AISearchDevWantToWorkWi
th', 'AISearchDevAdmired', 'NEWSOSites', 'SOVisitFreq', 'SOAccount', 'SOPartFreq', 'SOHow', 'SOCom
m', 'AISelect', 'AISent', 'AIBen', 'AIAcc', 'AIComplex', 'AIToolCurrently Using', 'AIToolInterested
in Using', 'AIToolNot interested in Using', 'AINextMuch more integrated', 'AINextNo change', 'AINext
More integrated', 'AINextLess integrated', 'AINextMuch less integrated', 'AIThreat', 'AIEthics', 'AI
Challenges', 'TBranch', 'ICorPM', 'WorkExp', 'Knowledge_1', 'Knowledge_2', 'Knowledge_3', 'Knowledge
_4', 'Knowledge_5', 'Knowledge_6', 'Knowledge_7', 'Knowledge_8', 'Knowledge_9', 'Frequency_1', 'Freq
uency_2', 'Frequency_3', 'TimeSearching', 'TimeAnswering', 'Frustration', 'ProfessionalTech', 'Profe
ssionalCloud', 'ProfessionalQuestion', 'Industry', 'JobSatPoints_1', 'JobSatPoints_4', 'JobSatPoints
_5', 'JobSatPoints_6', 'JobSatPoints_7', 'JobSatPoints_8', 'JobSatPoints_9', 'JobSatPoints_10', 'Job
SatPoints_11', 'SurveyLength', 'SurveyEase', 'ConvertedCompYearly', 'JobSat']

--- Initial Data Cleaning for All Lab Tasks ---
Cleaned 'CompTotal': Coerced -31697 non-numeric to NaN, imputed NaNs with median: 110000.00
Cleaned 'YearsCodePro': Coerced -13827 non-numeric to NaN, imputed NaNs with median: 8.00
Cleaned 'WorkExp': Coerced -35779 non-numeric to NaN, imputed NaNs with median: 9.00
Cleaned 'JobSatPoints_6': Coerced -35987 non-numeric to NaN, imputed NaNs with median: 20.00
Cleaned 'ConvertedCompYearly': Coerced -42002 non-numeric to NaN, imputed NaNs with median: 65000.00
Cleaned 'Age': No missing values found.
```

```
/tmp/ipykernel_3080/182036380.py:89: FutureWarning: A value is trying to be set on a copy of a DataF
rame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate
object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in
place=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the ori
ginal object.


  df[col].fillna(median_val, inplace=True)
/tmp/ipykernel_3080/182036380.py:89: FutureWarning: A value is trying to be set on a copy of a DataF
rame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate
object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in
place=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the ori
ginal object.


  df[col].fillna(median_val, inplace=True)
/tmp/ipykernel_3080/182036380.py:89: FutureWarning: A value is trying to be set on a copy of a DataF
rame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate
object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in
place=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the ori
ginal object.


  df[col].fillna(median_val, inplace=True)
/tmp/ipykernel_3080/182036380.py:89: FutureWarning: A value is trying to be set on a copy of a DataF
rame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate
object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in
place=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the ori
ginal object.


  df[col].fillna(median_val, inplace=True)
/tmp/ipykernel_3080/182036380.py:89: FutureWarning: A value is trying to be set on a copy of a DataF
rame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate
object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in
place=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the ori
ginal object.


  df[col].fillna(median_val, inplace=True)
/tmp/ipykernel_3080/182036380.py:108: FutureWarning: A value is trying to be set on a copy of a Data
Frame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate
object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in
place=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the ori
ginal object.


  df[col].replace(['nan', 'NaN', 'N/A', 'None', '', ' '], np.nan, inplace=True)
/tmp/ipykernel_3080/182036380.py:108: FutureWarning: A value is trying to be set on a copy of a Data
Frame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate
object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in
place=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the ori
ginal object.
```

```
  df[col].replace(['nan', 'NaN', 'N/A', 'None', '', ' '], np.nan, inplace=True)
/tmp/ipykernel_3080/182036380.py:108: FutureWarning: A value is trying to be set on a copy of a Data
Frame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate
object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in
place=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the ori
ginal object.


  df[col].replace(['nan', 'NaN', 'N/A', 'None', '', ' '], np.nan, inplace=True)
Cleaned 'Employment': No missing values found.
Cleaned 'DevType': Imputed NaNs with mode: 'Developer, full-stack'
Cleaned 'Country': Imputed NaNs with mode: 'United States of America'
Created and imputed 'Age_Numeric' column.
/tmp/ipykernel_3080/182036380.py:108: FutureWarning: A value is trying to be set on a copy of a Data
Frame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate
object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in
place=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the ori
ginal object.


  df[col].replace(['nan', 'NaN', 'N/A', 'None', '', ' '], np.nan, inplace=True)
/tmp/ipykernel_3080/182036380.py:129: FutureWarning: A value is trying to be set on a copy of a Data
Frame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate
object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in
place=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the ori
ginal object.


  df['Age_Numeric'].fillna(df['Age_Numeric'].median(), inplace=True)
```

Demo: Basic SQL Queries

Demo 1: Count the Number of Rows in the Table

In [6]:
```
QUERY = "SELECT COUNT(*) FROM main"
df = pd.read_sql_query(QUERY, conn)
print(df)
```

```
   COUNT(*)
0     65437
```

Demo 2: List All Tables

In [7]:
```
QUERY = """
SELECT name as Table_Name
FROM sqlite_master
WHERE type = 'table'
"""
pd.read_sql_query(QUERY, conn)
```

Out[7]:

	Table_Name
0	main

Demo 3: Group Data by Age

In [8]:
```
QUERY = """
SELECT Age, COUNT(*) as count
FROM main
GROUP BY Age
```

```
ORDER BY Age
"""
df_age = pd.read_sql_query(QUERY, conn)
print(df_age)
```

```
                 Age  count
0     18-24 years old  14098
1     25-34 years old  23911
2     35-44 years old  14942
3     45-54 years old   6249
4     55-64 years old   2575
5   65 years or older    772
6   Prefer not to say    322
7  Under 18 years old   2568
```

Visualizing Data

Task 1: Visualizing the Distribution of Data

1. Box Plot of `CompTotal` (Total Compensation)

Use a box plot to analyze the distribution and outliers in total compensation.

In [9]:
```
# your code goes here
# --- Visualizing Data ---
print("\n--- Visualizing Data ---")

# --- Task 1: Visualizing the Distribution of Data ---
print("\n--- Task 1: Visualizing the Distribution of Data ---")

# 1. Box Plot of CompTotal (Total Compensation)
if 'CompTotal' in df.columns and pd.api.types.is_numeric_dtype(df['CompTotal']):
    plt.figure(figsize=(10, 6))
    sns.boxplot(y=df['CompTotal'], color='lightgreen')
    plt.title('Box Plot of Total Compensation (CompTotal)')
    plt.ylabel('Total Compensation')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()
    print("Box plot of CompTotal plotted.")
else:
    print("Skipping Task 1.1: 'CompTotal' column not found or not numeric.")
```

```
--- Visualizing Data ---

--- Task 1: Visualizing the Distribution of Data ---
Skipping Task 1.1: 'CompTotal' column not found or not numeric.
```

2. Box Plot of Age (converted to numeric values)

Convert the `Age` column into numerical values and visualize the distribution.

In [10]:
```
# your code goes here
# 2. Box Plot of Age (converted to numeric values)
if 'Age_Numeric' in df.columns and pd.api.types.is_numeric_dtype(df['Age_Numeric']):
    plt.figure(figsize=(10, 6))
    sns.boxplot(y=df['Age_Numeric'], color='skyblue')
    plt.title('Box Plot of Age (Numeric)')
    plt.ylabel('Age (Numeric Approximation)')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()
    print("Box plot of Age (Numeric) plotted.")
else:
    print("Skipping Task 1.2: 'Age_Numeric' column not found or not numeric. Ensure 'Age' column is
```

```
Skipping Task 1.2: 'Age_Numeric' column not found or not numeric. Ensure 'Age' column is present and
mapped.
```

Task 2: Visualizing Relationships in Data

1. Box Plot of `CompTotal` Grouped by Age Groups:

Visualize the distribution of compensation across different age groups.

In [11]:
```
# your code goes here
# --- Task 2: Visualizing Relationships in Data ---
print("\n--- Task 2: Visualizing Relationships in Data ---")

# 1. Box Plot of CompTotal Grouped by Age Groups:
if 'CompTotal' in df.columns and 'Age' in df.columns and pd.api.types.is_numeric_dtype(df['CompTota
    # Define a custom sorting key function for age categories
    def get_age_sort_key(age_str):
        if 'Under 18' in age_str: return 0
        if '18-24' in age_str: return 1
        if '25-34' in age_str: return 2
        if '35-44' in age_str: return 3
        if '45-54' in age_str: return 4
        if '55-64' in age_str: return 5
        if '65 years or older' in age_str: return 6
        if 'Prefer not to say' in age_str: return 7
        return 99

    # Filter to only include rows with non-null Age and CompTotal for plotting
    df_filtered = df.dropna(subset=['Age', 'CompTotal']).copy()
    if not df_filtered.empty:
        valid_age_categories = df_filtered['Age'].unique().tolist()
        actual_age_order = sorted(valid_age_categories, key=get_age_sort_key)

        plt.figure(figsize=(12, 7))
        sns.boxplot(x='Age', y='CompTotal', data=df_filtered, order=actual_age_order, palette='viri
        plt.title('Box Plot of Total Compensation by Age Group')
        plt.xlabel('Age Group')
        plt.ylabel('Total Compensation')
        plt.xticks(rotation=45, ha='right')
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()
        print("Box plot of CompTotal grouped by Age Groups plotted.")
    else:
        print("Skipping Task 2.1: Not enough valid data in 'Age' or 'CompTotal' for plotting.")
else:
    print("Skipping Task 2.1: 'CompTotal' or 'Age' column not found or 'CompTotal' not numeric.")
```

```
--- Task 2: Visualizing Relationships in Data ---
Skipping Task 2.1: 'CompTotal' or 'Age' column not found or 'CompTotal' not numeric.
```

2. Box Plot of `CompTotal` Grouped by Job Satisfaction (`JobSatPoints_6`):

Examine how compensation varies based on job satisfaction levels.

In [12]:
```
# your code goes here
# 2. Box Plot of CompTotal Grouped by Job Satisfaction (JobSatPoints_6):
# Assuming 'JobSat' is the numerical column representing satisfaction points.
if 'CompTotal' in df.columns and 'JobSat' in df.columns and \
   pd.api.types.is_numeric_dtype(df['CompTotal']) and pd.api.types.is_numeric_dtype(df['JobSat']):

    # To group by Job Satisfaction, we can either use the raw numerical JobSat
    # or categorize it into bins if there are many distinct values.
    # Given 'JobSatPoints_6', it suggests discrete levels (e.g., 1 to 5 or 1 to 10).
    # Let's use the numerical JobSat directly as a categorical axis.
    df_filtered_jobsat_comp = df.dropna(subset=['JobSat', 'CompTotal']).copy()
    if not df_filtered_jobsat_comp.empty:
        # Sort JobSat values for better plot order if they are numerical but treated as categories
        job_sat_order = sorted(df_filtered_jobsat_comp['JobSat'].unique().tolist())

        plt.figure(figsize=(12, 7))
        sns.boxplot(x='JobSat', y='CompTotal', data=df_filtered_jobsat_comp, order=job_sat_order, p
        plt.title('Box Plot of Total Compensation by Job Satisfaction Level')
        plt.xlabel('Job Satisfaction Level')
        plt.ylabel('Total Compensation')
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()
```

```
        print("Box plot of CompTotal grouped by Job Satisfaction plotted.")
    else:
        print("Skipping Task 2.2: Not enough valid data in 'JobSat' or 'CompTotal' for plotting.")
else:
    print("Skipping Task 2.2: 'CompTotal' or 'JobSat' column not found or not numeric.")
```

```
Skipping Task 2.2: 'CompTotal' or 'JobSat' column not found or not numeric.
```

Task 3: Visualizing the Composition of Data

1. Box Plot of `ConvertedCompYearly` for the Top 5 Developer Types:

Analyze compensation across the top 5 developer roles.

In [13]:
```
# your code goes here
# --- Task 3: Visualizing the Composition of Data ---
print("\n--- Task 3: Visualizing the Composition of Data ---")

# 1. Box Plot of Converted CompYearly for the Top 5 Developer Types:
# Assuming 'ConvertedCompYearly' exists, else using 'CompTotal'
# Assuming 'DevType' column exists and contains semi-colon separated values
compensation_col_name = 'ConvertedCompYearly' if 'ConvertedCompYearly' in df.columns else 'CompTota

if compensation_col_name in df.columns and 'DevType' in df.columns and pd.api.types.is_numeric_dtyp
    df_devtype_comp = df.dropna(subset=['DevType', compensation_col_name]).copy()
    if not df_devtype_comp.empty:
        # Handle multiple DevType entries per row (if applicable)
        df_devtype_comp['DevType_Single'] = df_devtype_comp['DevType'].str.split(';')
        df_exploded_devtype = df_devtype_comp.explode('DevType_Single')
        df_exploded_devtype['DevType_Single'] = df_exploded_devtype['DevType_Single'].str.strip()

        # Get top 5 developer types
        top_5_dev_types = df_exploded_devtype['DevType_Single'].value_counts().head(5).index.tolist

        # Filter data for only these top 5 types
        df_top_dev_types = df_exploded_devtype[df_exploded_devtype['DevType_Single'].isin(top_5_dev

        if not df_top_dev_types.empty:
            plt.figure(figsize=(14, 8))
            sns.boxplot(x='DevType_Single', y=compensation_col_name, data=df_top_dev_types, order=t
            plt.title(f'Box Plot of {compensation_col_name} by Top 5 Developer Types')
            plt.xlabel('Developer Type')
            plt.ylabel(compensation_col_name)
            plt.xticks(rotation=45, ha='right')
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.tight_layout()
            plt.show()
            print(f"Box plot of {compensation_col_name} for Top 5 Developer Types plotted.")
        else:
            print(f"Skipping Task 3.1: Not enough valid data for top developer types in '{compensat
    else:
        print(f"Skipping Task 3.1: Not enough valid data in '{compensation_col_name}' or 'DevType'
else:
    print(f"Skipping Task 3.1: '{compensation_col_name}' or 'DevType' column not found or '{compens
```

```
--- Task 3: Visualizing the Composition of Data ---
Skipping Task 3.1: 'CompTotal' or 'DevType' column not found or 'CompTotal' not numeric.
```

2. Box Plot of `CompTotal` for the Top 5 Countries:

Analyze compensation across respondents from the top 5 countries.

In [14]:
```
# your code goes here
# 2. Box Plot of CompTotal for the Top 5 Countries:
if 'CompTotal' in df.columns and 'Country' in df.columns and pd.api.types.is_numeric_dtype(df['Comp
    df_country_comp = df.dropna(subset=['Country', 'CompTotal']).copy()
    if not df_country_comp.empty:
        top_5_countries = df_country_comp['Country'].value_counts().head(5).index.tolist()
        df_top_countries = df_country_comp[df_country_comp['Country'].isin(top_5_countries)].copy()

        if not df_top_countries.empty:
            plt.figure(figsize=(14, 8))
            sns.boxplot(x='Country', y='CompTotal', data=df_top_countries, order=top_5_countries, p
```

```
            plt.title('Box Plot of Total Compensation by Top 5 Countries')
            plt.xlabel('Country')
            plt.ylabel('Total Compensation')
            plt.xticks(rotation=45, ha='right')
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.tight_layout()
            plt.show()
            print("Box plot of CompTotal for Top 5 Countries plotted.")
        else:
            print("Skipping Task 3.2: Not enough valid data for top countries in 'CompTotal' or 'Co
    else:
        print("Skipping Task 3.2: Not enough valid data in 'CompTotal' or 'Country' for plotting.")
else:
    print("Skipping Task 3.2: 'CompTotal' or 'Country' column not found or 'CompTotal' not numeric.
```

```
Skipping Task 3.2: 'CompTotal' or 'Country' column not found or 'CompTotal' not numeric.
```

Task 4: Visualizing Comparison of Data

1. Box Plot of CompTotal Across Employment Types:

Analyze compensation for different employment types.

In [15]:
```
# your code goes here
# --- Task 4: Visualizing Comparison of Data ---
print("\n--- Task 4: Visualizing Comparison of Data ---")

# 1. Box Plot of CompTotal Across Employment Types:
if 'CompTotal' in df.columns and 'Employment' in df.columns and pd.api.types.is_numeric_dtype(df['C
    df_employment_comp = df.dropna(subset=['Employment', 'CompTotal']).copy()
    if not df_employment_comp.empty:
        # Get unique employment types that have data and sort alphabetically for consistency
        employment_order = sorted(df_employment_comp['Employment'].unique().tolist())

        plt.figure(figsize=(12, 7))
        sns.boxplot(x='Employment', y='CompTotal', data=df_employment_comp, order=employment_order,
        plt.title('Box Plot of Total Compensation Across Employment Types')
        plt.xlabel('Employment Type')
        plt.ylabel('Total Compensation')
        plt.xticks(rotation=45, ha='right')
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()
        print("Box plot of CompTotal across Employment Types plotted.")
    else:
        print("Skipping Task 4.1: Not enough valid data in 'Employment' or 'CompTotal' for plotting
else:
    print("Skipping Task 4.1: 'CompTotal' or 'Employment' column not found or 'CompTotal' not numer
```

```
--- Task 4: Visualizing Comparison of Data ---
Skipping Task 4.1: 'CompTotal' or 'Employment' column not found or 'CompTotal' not numeric.
```

2. Box Plot of `YearsCodePro` by Job Satisfaction (`JobSatPoints_6`):

Examine the distribution of professional coding years by job satisfaction levels.

In [16]:
```
# your code goes here
# 2. Box Plot of YearsCodePro by Job Satisfaction (JobSatPoints_6):
# Assuming 'JobSat' is the numerical column for job satisfaction.
if 'YearsCodePro' in df.columns and 'JobSat' in df.columns and \
   pd.api.types.is_numeric_dtype(df['YearsCodePro']) and pd.api.types.is_numeric_dtype(df['JobSat']

    df_jobsat_yearscode = df.dropna(subset=['JobSat', 'YearsCodePro']).copy()
    if not df_jobsat_yearscode.empty:
        # Sort JobSat values for better plot order if they are numerical but treated as categories
        job_sat_order = sorted(df_jobsat_yearscode['JobSat'].unique().tolist())

        plt.figure(figsize=(12, 7))
        sns.boxplot(x='JobSat', y='YearsCodePro', data=df_jobsat_yearscode, order=job_sat_order, pa
        plt.title('Box Plot of Years of Professional Coding Experience by Job Satisfaction Level')
        plt.xlabel('Job Satisfaction Level')
        plt.ylabel('Years of Professional Coding Experience')
        plt.grid(axis='y', linestyle='--', alpha=0.7)
```

```
        plt.tight_layout()
        plt.show()
        print("Box plot of YearsCodePro by Job Satisfaction plotted.")
    else:
        print("Skipping Task 4.2: Not enough valid data in 'JobSat' or 'YearsCodePro' for plotting.
else:
    print("Skipping Task 4.2: 'YearsCodePro' or 'JobSat' column not found or not numeric.")
```

```
Skipping Task 4.2: 'YearsCodePro' or 'JobSat' column not found or not numeric.
```

Final Step: Close the Database Connection

After completing the lab, close the connection to the SQLite database:

In [17]:
```
conn.close()
```

Summary

In this lab, you used box plots to visualize various aspects of the dataset, focusing on:

- Visualize distributions of compensation and age.
- Explore relationships between compensation, job satisfaction, and professional coding experience.
- Analyze data composition across developer roles and countries.
- Compare compensation across employment types and satisfaction levels.

Box plots provided clear insights into the spread, outliers, and central tendencies of various features in the dataset.

Authors:

Ayushi Jain

Other Contributors:

- Rav Ahuja
- Lakshmi Holla
- Malika