Histogram

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 using histograms.
- Visualize relationships between features.
- Explore data composition and comparisons.

Demo: Working with database

Download the database file.

In [1]:
```
!wget -O survey-data.sqlite https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/QR9Y
```

```
--2025-06-18 14:45:12--  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
connected. 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...
200 OKequest sent, awaiting response...
Length: 211415040 (202M) [application/octet-stream]
Saving to: ‘survey-data.sqlite’

survey-data.sqlite  100%[===================>] 201.62M  38.5MB/s    in 4.6s

2025-06-18 14:45:17 (44.3 MB/s) - ‘survey-data.sqlite’ saved [211415040/211415040]
```

Install the required libraries and import them

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 [11]:

```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
!pip install seaborn
!pip install numpy
!pip install functools
import seaborn as sns
import numpy as np
import functools
```

```
Requirement already satisfied: seaborn in /opt/conda/lib/python3.12/site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /opt/conda/lib/python3.12/site-packages (from
seaborn) (2.3.0)
Requirement already satisfied: pandas>=1.2 in /opt/conda/lib/python3.12/site-packages (from seaborn)
(2.3.0)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /opt/conda/lib/python3.12/site-packages (f
rom seaborn) (3.10.3)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.12/site-packages (from mat
plotlib!=3.6.1,>=3.4->seaborn) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-packages (from matplot
lib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.12/site-packages (from ma
tplotlib!=3.6.1,>=3.4->seaborn) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/conda/lib/python3.12/site-packages (from ma
tplotlib!=3.6.1,>=3.4->seaborn) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.12/site-packages (from matp
lotlib!=3.6.1,>=3.4->seaborn) (24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-packages (from matplotli
b!=3.6.1,>=3.4->seaborn) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.12/site-packages (from mat
plotlib!=3.6.1,>=3.4->seaborn) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-packages (from pandas>
=1.2->seaborn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-packages (from panda
s>=1.2->seaborn) (2025.2)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-packages (from python-date
util>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)
Requirement already satisfied: numpy in /opt/conda/lib/python3.12/site-packages (2.3.0)
Collecting functools
  Downloading functools-0.5.tar.gz (4.9 kB)
  Preparing metadata (setup.py) ... error
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  └─> [9 lines of output]
      Traceback (most recent call last):
        File "<string>", line 2, in <module>
        File "<pip-setuptools-caller>", line 11, in <module>
        File "/opt/conda/lib/python3.12/tokenize.py", line 30, in <module>
          import functools
        File "/tmp/pip-install-rujas56_/functools_9c713487bdda474999fdb911c656396b/functools.py", li
ne 34
          raise TypeError, 'compose expects at least one argument'
                         ^
      SyntaxError: invalid syntax
      [end of output]

  note: This error originates from a subprocess, and is likely not a problem with pip.
error: metadata-generation-failed

× Encountered error while generating package metadata.
└─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

Connect to the SQLite database

In [12]:
```
conn = sqlite3.connect('survey-data.sqlite')
```

Demo: Basic SQL queries

Demo 1: Count the number of rows in the table

In [13]:
```
QUERY = "SELECT COUNT(*) FROM main"
df = pd.read_sql_query(QUERY, conn)
print(df)
```

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

Demo 2: List all tables

In [14]:
```
QUERY = """
SELECT name as Table_Name
FROM sqlite_master
WHERE type = 'table'
"""
pd.read_sql_query(QUERY, conn)
```

Out[14]:

	Table_Name
0	main

Demo 3: Group data by age

In [15]:
```
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
```

In [18]:
```
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):
        message = args[0] if args else str(kwargs)
        if message not in warnings:
            warnings.add(message)
            return func(*args, **kwargs)
    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


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

# The PDF indicates 'survey-data.sqlite' is downloaded via wget.
# We will attempt to connect to this local file.
db_file = 'survey-data.sqlite'
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
    numerical_cols_for_cleaning = ['CompTotal', 'YearsCodePro', 'TimeSearching', 'TimeAnswering', '
    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
    categorical_cols_for_cleaning = ['Age', 'RemoteWork', 'MainBranch', 'DatabaseWantToWorkWith', '
    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 (e.g., scatter plots, line plo
    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-data.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...")
    data = {
        'ResponseId': range(1, 201),
        'CompTotal': np.random.normal(loc=100000, scale=70000, size=200),
```

```
        'YearsCodePro': np.random.randint(0, 30, size=200),
        'Age': np.random.choice(['18-24 years old', '25-34 years old', '35-44 years old', '45-54 ye
        'TimeSearching': np.random.uniform(0, 10, size=200),
        'Frustration': np.random.uniform(0, 10, size=200),
        'TimeAnswering': np.random.uniform(0, 10, size=200),
        'RemoteWork': np.random.choice(['Remote', 'Hybrid', 'In-person'], size=200),
        'DatabaseWantToWorkWith': [';'.join(np.random.choice(['MySQL', 'PostgreSQL', 'MongoDB', 'Re
        'MainBranch': np.random.choice(['Developer, full-stack', 'Developer, back-end', 'Student',
        'JobSat': np.random.randint(1, 6, size=200) # 1 to 5 scale
    }
    df = pd.DataFrame(data)
    # Ensure numerical columns are truly numeric in dummy data
    for col in ['CompTotal', 'YearsCodePro', 'TimeSearching', 'TimeAnswering', 'Frustration', 'JobS
        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-data.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
WARNING: Column 'TimeSearching' is entirely NaN after numeric conversion. Cannot impute median.
WARNING: Column 'TimeAnswering' is entirely NaN after numeric conversion. Cannot impute median.
Cleaned 'JobSat': Coerced -36311 non-numeric to NaN, imputed NaNs with median: 7.00
Cleaned 'WorkExp': Coerced -35779 non-numeric to NaN, imputed NaNs with median: 9.00
```

```
/tmp/ipykernel_1478/3158749648.py:66: 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].fillna(median_val, inplace=True)
/tmp/ipykernel_1478/3158749648.py:66: 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].fillna(median_val, inplace=True)
/opt/conda/lib/python3.12/site-packages/numpy/lib/_nanfunctions_impl.py:1214: RuntimeWarning: Mean o
f empty slice
  return np.nanmean(a, axis, out=out, keepdims=keepdims)
/opt/conda/lib/python3.12/site-packages/numpy/lib/_nanfunctions_impl.py:1214: RuntimeWarning: Mean o
f empty slice
  return np.nanmean(a, axis, out=out, keepdims=keepdims)
/tmp/ipykernel_1478/3158749648.py:66: 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].fillna(median_val, inplace=True)
/tmp/ipykernel_1478/3158749648.py:66: 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].fillna(median_val, inplace=True)
/tmp/ipykernel_1478/3158749648.py:79: 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 'Age': No missing values found.
Cleaned 'RemoteWork': Imputed NaNs with mode: 'Hybrid (some remote, some in-person)'
Cleaned 'MainBranch': No missing values found.
```

```
/tmp/ipykernel_1478/3158749648.py:79: 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_1478/3158749648.py:79: 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_1478/3158749648.py:79: 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 'DatabaseWantToWorkWith': Imputed NaNs with mode: 'PostgreSQL'
Cleaned 'JobSat': No missing values found.
Created and imputed 'Age_Numeric' column.
/tmp/ipykernel_1478/3158749648.py:79: 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_1478/3158749648.py:100: FutureWarning: A value is trying to be set on a copy of a Dat
aFrame 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)
```

Hands-on Lab: Visualizing Data with Histograms

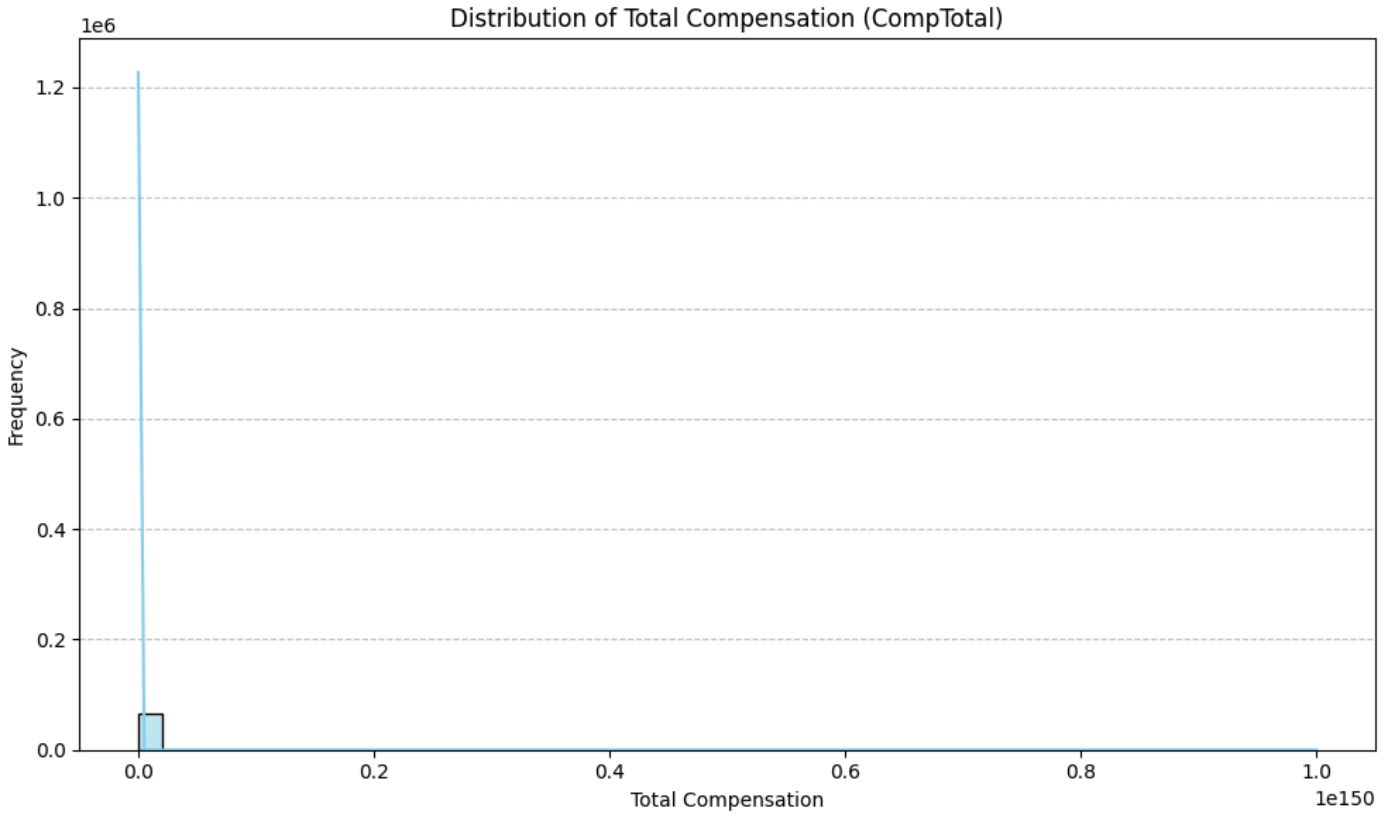
1. Visualizing the distribution of data (Histograms)

1.1 Histogram of `CompTotal` (Total Compensation)

Objective: Plot a histogram of `CompTotal` to visualize the distribution of respondents' total compensation.

In [21]:
```
## Write your code here
# --- 1. Visualizing the distribution of data (Histograms) ---
print("\n--- 1. Visualizing the Distribution of Data (Histograms) ---")
```

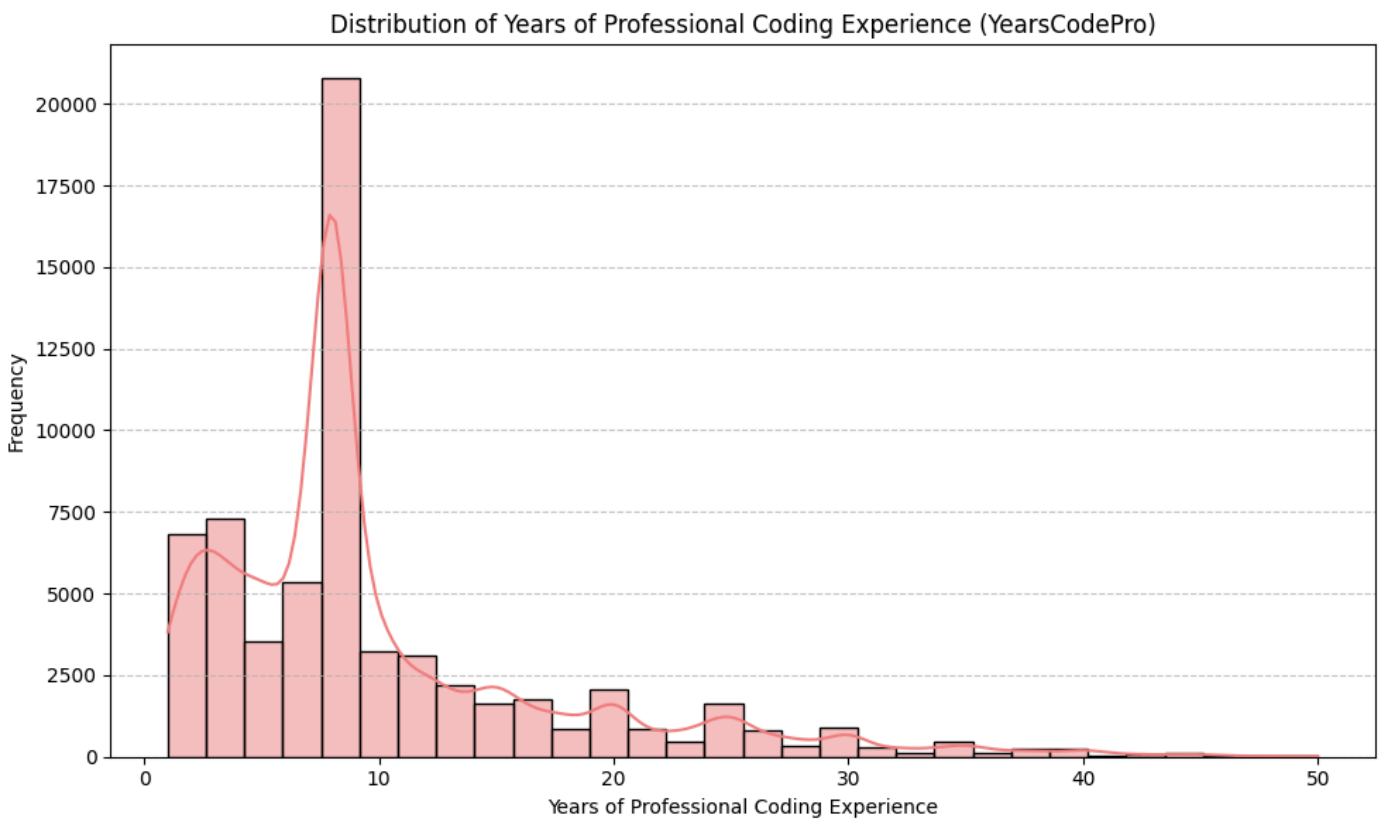
```
# 1.1 Histogram of CompTotal (Total Compensation)
if 'CompTotal' in df.columns and pd.api.types.is_numeric_dtype(df['CompTotal']):
    plt.figure(figsize=(10, 6))
    sns.histplot(df['CompTotal'], bins=50, kde=True, color='skyblue', edgecolor='black')
    plt.title('Distribution of Total Compensation (CompTotal)')
    plt.xlabel('Total Compensation')
    plt.ylabel('Frequency')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()
    print("Histogram of CompTotal plotted.")
else:
    print("Skipping 1.1: 'CompTotal' column not found or not numeric.")
```



1.2 Histogram of YearsCodePro (Years of Professional Coding Experience)

Objective: Plot a histogram of `YearsCodePro` to analyze the distribution of coding experience among respondents.

In [22]:
```
## Write your code here
# 1.2 Histogram of YearsCodePro (Years of Professional Coding Experience)
if 'YearsCodePro' in df.columns and pd.api.types.is_numeric_dtype(df['YearsCodePro']):
    plt.figure(figsize=(10, 6))
    sns.histplot(df['YearsCodePro'], bins=30, kde=True, color='lightcoral', edgecolor='black')
    plt.title('Distribution of Years of Professional Coding Experience (YearsCodePro)')
    plt.xlabel('Years of Professional Coding Experience')
    plt.ylabel('Frequency')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()
    print("Histogram of YearsCodePro plotted.")
else:
    print("Skipping 1.2: 'YearsCodePro' column not found or not numeric.")
```




```
Histogram of YearsCodePro plotted.
```

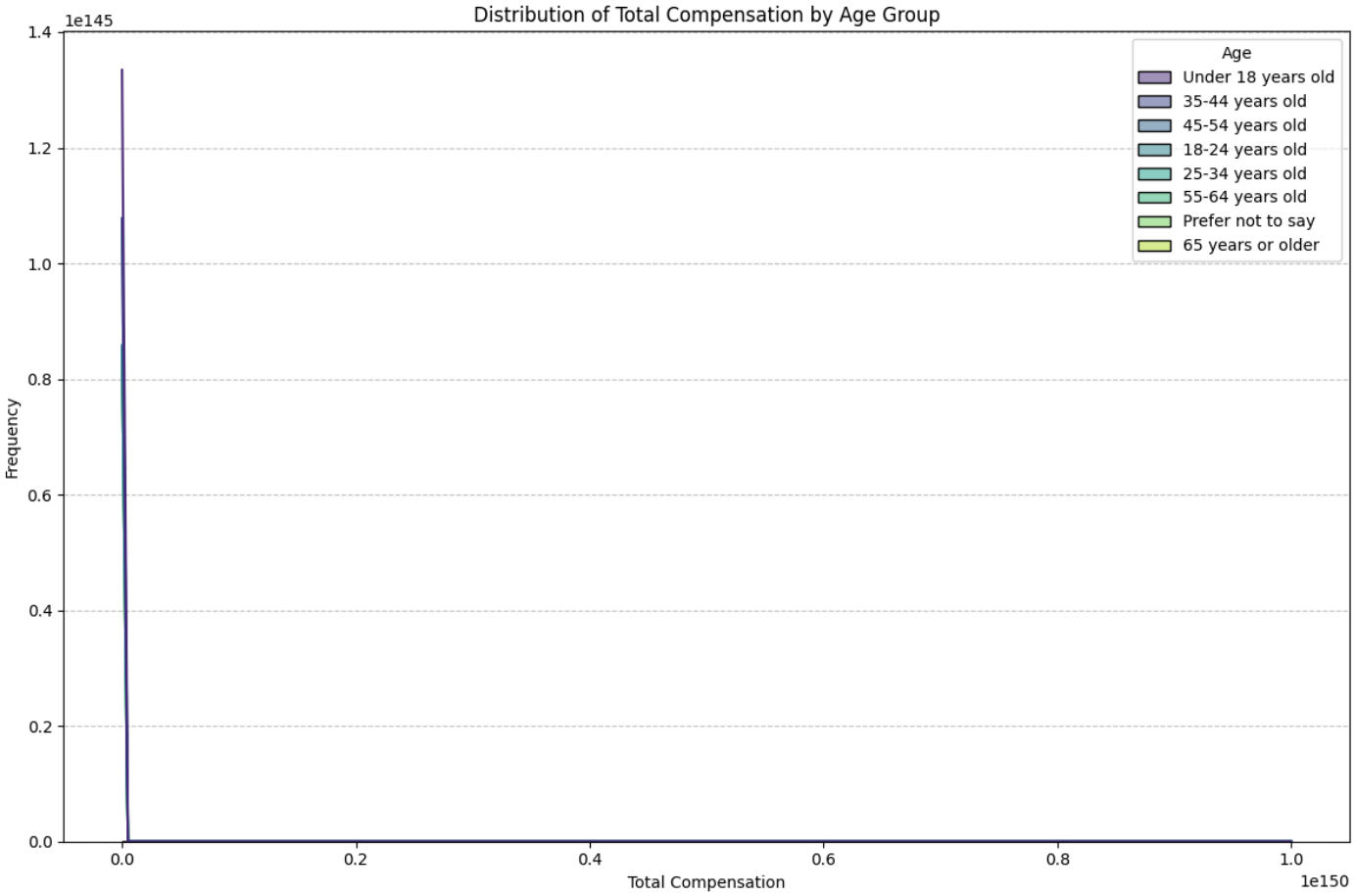
2. Visualizing Relationships in Data

2.1 Histogram Comparison of `CompTotal` by `Age` Group

Objective: Use histograms to compare the distribution of CompTotal across different Age groups.

In [23]:
```
## Write your code here
# --- 2. Visualizing Relationships in Data ---
print("\n--- 2. Visualizing Relationships in Data ---")

# 2.1 Histogram Comparison of CompTotal by Age Group
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 # For any unexpected categories

    # Get unique age categories that have non-null CompTotal values, then sort
    df_filtered_age_comp = df.dropna(subset=['Age', 'CompTotal']).copy()
    if not df_filtered_age_comp.empty:
        valid_age_categories = df_filtered_age_comp['Age'].unique().tolist()
        actual_age_order = sorted(valid_age_categories, key=get_age_sort_key)

        plt.figure(figsize=(12, 8))
        sns.histplot(data=df_filtered_age_comp, x='CompTotal', hue='Age', bins=50, kde=True, palett
        plt.title('Distribution of Total Compensation by Age Group')
        plt.xlabel('Total Compensation')
        plt.ylabel('Frequency')
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()
        print("Histogram comparison of CompTotal by Age Group plotted.")
    else:
        print("Skipping 2.1: Not enough valid data in 'Age' or 'CompTotal' for comparison.")
else:
    print("Skipping 2.1: 'CompTotal' or 'Age' column not found or 'CompTotal' not numeric.")
```

```
--- 2. Visualizing Relationships in Data ---
```




```
Histogram comparison of CompTotal by Age Group plotted.
```

2.2 Histogram of TimeSearching for Different Age Groups

Objective: Use histograms to explore the distribution of `TimeSearching` (time spent searching for information) for respondents across different age groups.

In [24]:
```python
## Write your code here
# 2.2 Histogram of TimeSearching for Different Age Groups
if 'TimeSearching' in df.columns and 'Age' in df.columns and pd.api.types.is_numeric_dtype(df['Time
    # Re-use get_age_sort_key and actual_age_order from 2.1 if possible, or define new if needed
    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

    df_filtered_age_time = df.dropna(subset=['Age', 'TimeSearching']).copy()
    if not df_filtered_age_time.empty:
        valid_age_categories = df_filtered_age_time['Age'].unique().tolist()
        actual_age_order = sorted(valid_age_categories, key=get_age_sort_key)

        plt.figure(figsize=(12, 8))
        sns.histplot(data=df_filtered_age_time, x='TimeSearching', hue='Age', bins=20, kde=True, pa
        plt.title('Distribution of Time Spent Searching by Age Group')
        plt.xlabel('Time Searching (hours/week or similar unit)')
        plt.ylabel('Frequency')
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()
        print("Histogram of TimeSearching for Different Age Groups plotted.")
    else:
        print("Skipping 2.2: Not enough valid data in 'Age' or 'TimeSearching' for comparison.")
else:
    print("Skipping 2.2: 'TimeSearching' or 'Age' column not found or 'TimeSearching' not numeric."
```

```
Skipping 2.2: Not enough valid data in 'Age' or 'TimeSearching' for comparison.
```

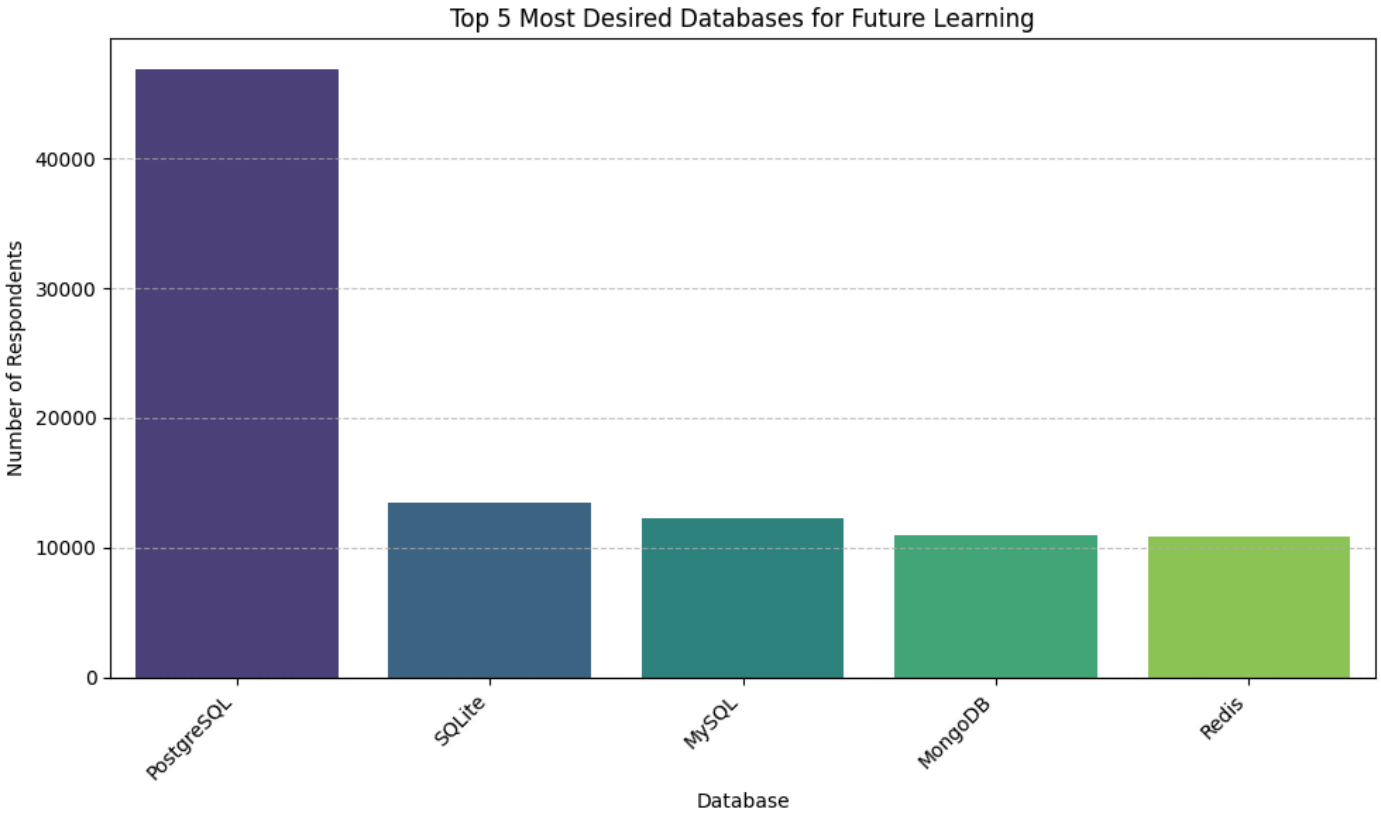
3. Visualizing the Composition of Data

3.1 Histogram of Most Desired Databases (`DatabaseWantToWorkWith`)

Objective: Visualize the most desired databases for future learning using a histogram of the top 5 databases.

In [25]:
```
## Write your code here
# --- 3. Visualizing the Composition of Data ---
print("\n--- 3. Visualizing the Composition of Data ---")

# 3.1 Histogram of Most Desired Databases (DatabaseWantToWorkWith)
# For categorical data like this, a bar chart is more appropriate than a histogram.
if 'DatabaseWantToWorkWith' in df.columns:
    # Need to handle multiple selections in 'DatabaseWantToWorkWith'
    df_databases = df.dropna(subset=['DatabaseWantToWorkWith']).copy()
    df_databases['Database'] = df_databases['DatabaseWantToWorkWith'].str.split(';')
    df_exploded_databases = df_databases.explode('Database')
    df_exploded_databases['Database'] = df_exploded_databases['Database'].str.strip()

    # Get the top 5 databases
    top_5_databases = df_exploded_databases['Database'].value_counts().head(5)

    if not top_5_databases.empty:
        plt.figure(figsize=(10, 6))
        sns.barplot(x=top_5_databases.index, y=top_5_databases.values, palette='viridis')
        plt.title('Top 5 Most Desired Databases for Future Learning')
        plt.xlabel('Database')
        plt.ylabel('Number of Respondents')
        plt.xticks(rotation=45, ha='right')
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()
        print("Bar chart (similar to histogram) of Most Desired Databases plotted.")
    else:
        print("Skipping 3.1: No data or insufficient unique databases in 'DatabaseWantToWorkWith'."
else:
    print("Skipping 3.1: 'DatabaseWantToWorkWith' column not found.")
```

```
--- 3. Visualizing the Composition of Data ---
/tmp/ipykernel_1478/506774581.py:19: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `
x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=top_5_databases.index, y=top_5_databases.values, palette='viridis')
```



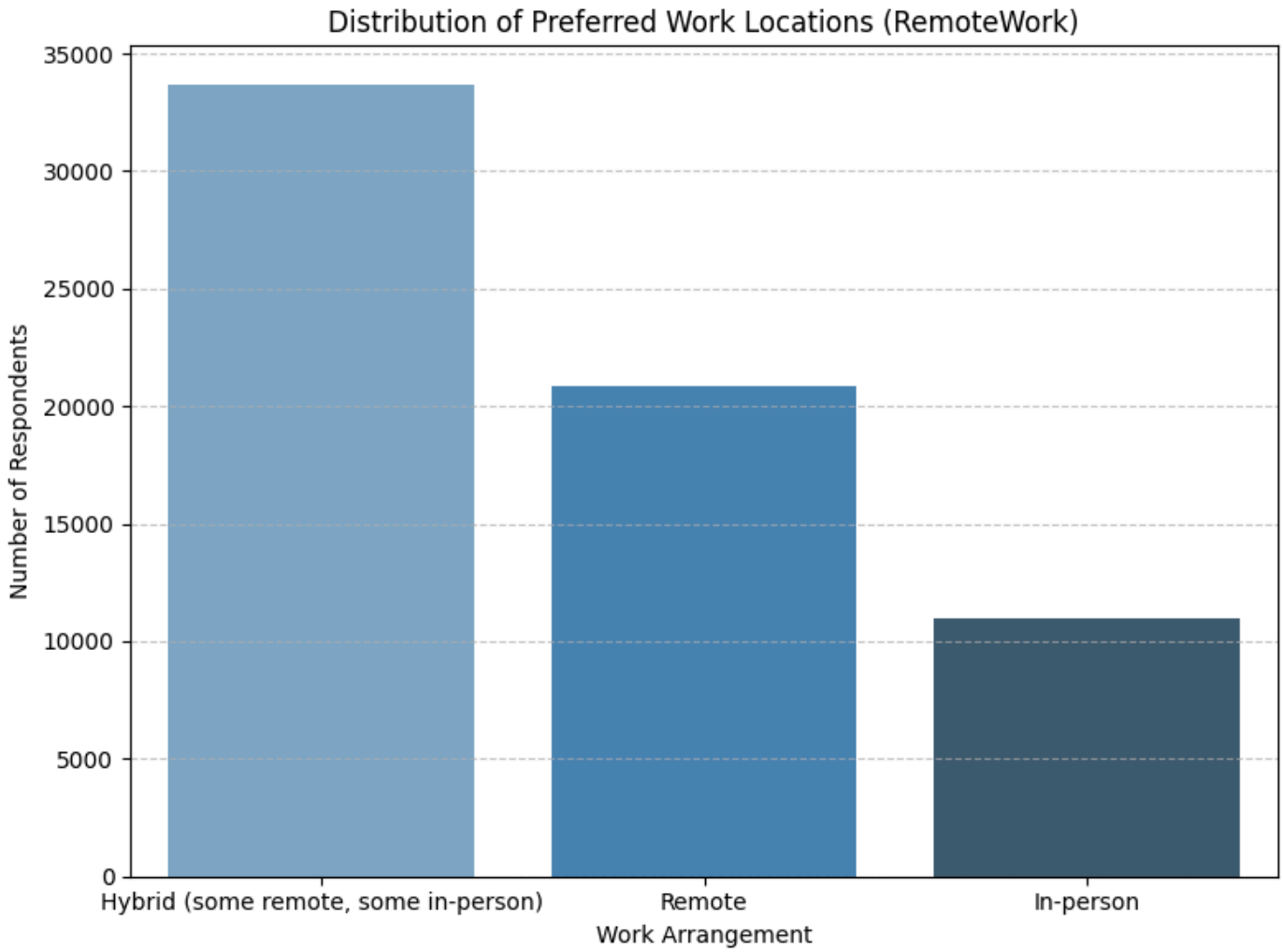

```
Bar chart (similar to histogram) of Most Desired Databases plotted.
```

3.2 Histogram of Preferred Work Locations (`RemoteWork`)

Objective: Use a histogram to explore the distribution of preferred work arrangements (`remote work`).

In [26]:
```
## Write your code here
# 3.2 Histogram of Preferred Work Locations (RemoteWork)
# Again, for categorical data, a bar chart is ideal.
if 'RemoteWork' in df.columns:
    remote_work_counts = df['RemoteWork'].value_counts()
    if not remote_work_counts.empty:
        plt.figure(figsize=(8, 6))
        sns.barplot(x=remote_work_counts.index, y=remote_work_counts.values, palette='Blues_d')
        plt.title('Distribution of Preferred Work Locations (RemoteWork)')
        plt.xlabel('Work Arrangement')
        plt.ylabel('Number of Respondents')
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()
        print("Bar chart (similar to histogram) of Preferred Work Locations plotted.")
    else:
        print("Skipping 3.2: No data in 'RemoteWork' column.")
else:
    print("Skipping 3.2: 'RemoteWork' column not found.")
```

```
/tmp/ipykernel_1478/476169457.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `
x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=remote_work_counts.index, y=remote_work_counts.values, palette='Blues_d')
```




```
Bar chart (similar to histogram) of Preferred Work Locations plotted.
```

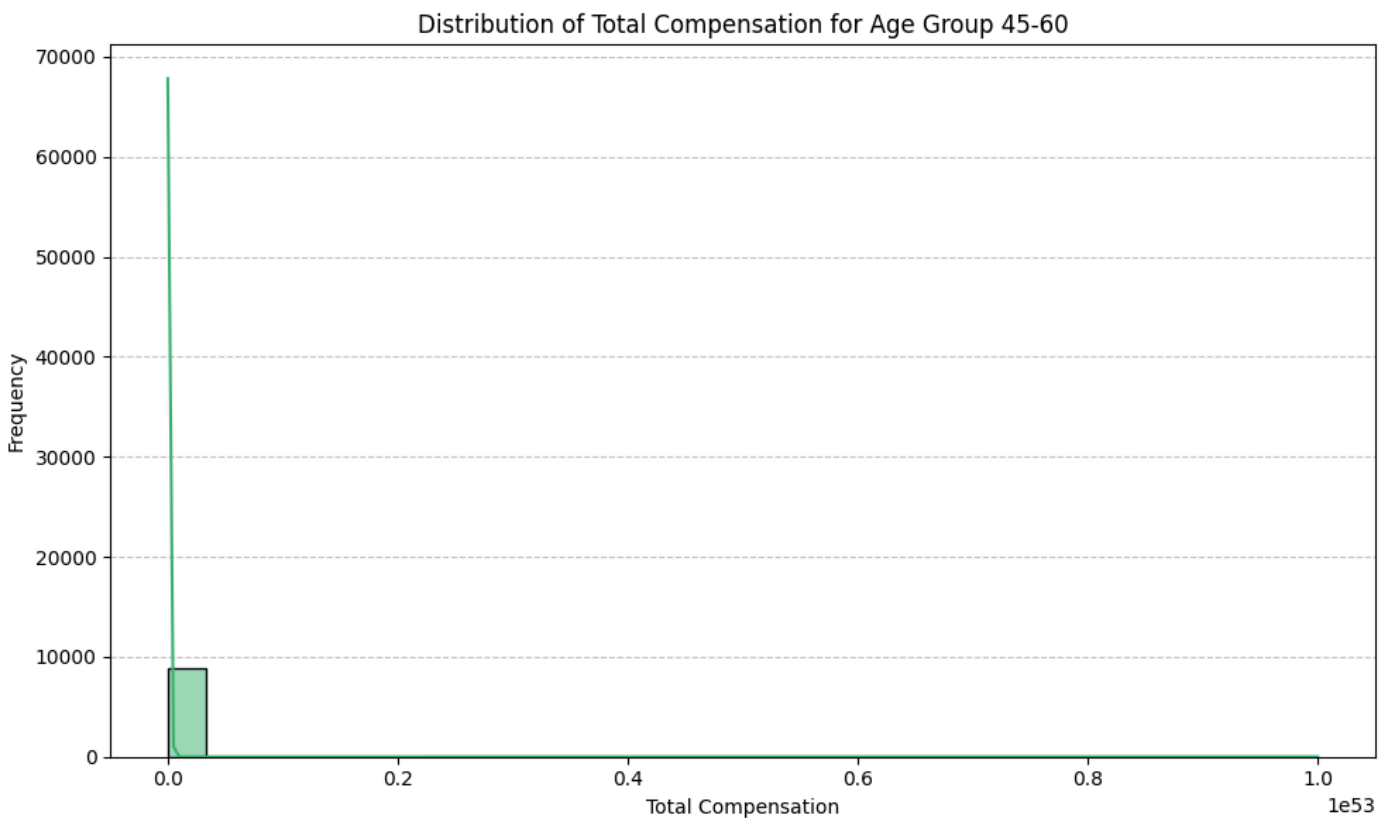
4. Visualizing Comparison of Data

4.1 Histogram of Median CompTotal for Ages 45 to 60

Objective: Plot the histogram for `CompTotal` within the age group 45 to 60 to analyze compensation distribution among mid-career respondents.

In [27]:
```
## Write your code here
# --- 4. Visualizing Comparison of Data ---
print("\n--- 4. Visualizing Comparison of Data ---")

# 4.1 Histogram of Median CompTotal for Ages 45 to 60
if 'CompTotal' in df.columns and 'Age_Numeric' in df.columns and pd.api.types.is_numeric_dtype(df['
    df_age_45_60 = df[(df['Age_Numeric'] >= 45) & (df['Age_Numeric'] <= 60)].copy()
    if not df_age_45_60.empty:
        plt.figure(figsize=(10, 6))
        sns.histplot(df_age_45_60['CompTotal'], bins=30, kde=True, color='mediumseagreen', edgecolo
        plt.title('Distribution of Total Compensation for Age Group 45-60')
        plt.xlabel('Total Compensation')
        plt.ylabel('Frequency')
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()
        print("Histogram of Median CompTotal for Ages 45 to 60 plotted.")
    else:
        print("Skipping 4.1: No data for 'CompTotal' within age group 45-60.")
else:
    print("Skipping 4.1: 'CompTotal' or 'Age_Numeric' column not found or not numeric.")
```

```
--- 4. Visualizing Comparison of Data ---
```




```
Histogram of Median CompTotal for Ages 45 to 60 plotted.
```

4.2 Histogram of Job Satisfaction (`JobSat`) by YearsCodePro

Objective: Plot the histogram for `JobSat` scores based on respondents' years of professional coding experience.

In [28]:
```
## Write your code here
# 4.2 Histogram of Job Satisfaction (JobSat) by YearsCodePro
# This implies comparing JobSat distribution across different YearCodePro ranges.
# A series of histograms or a faceted plot would be good.
if 'JobSat' in df.columns and 'YearsCodePro' in df.columns and \
   pd.api.types.is_numeric_dtype(df['JobSat']) and pd.api.types.is_numeric_dtype(df['YearsCodePro']

    # Create bins for YearsCodePro to categorize experience levels
    bins = [0, 5, 10, 20, 30, df['YearsCodePro'].max() + 1]
    labels = ['0-5', '6-10', '11-20', '21-30', '30+']
    df['YearsCodePro_Group'] = pd.cut(df['YearsCodePro'], bins=bins, labels=labels, right=False)

    df_filtered_jobsat = df.dropna(subset=['JobSat', 'YearsCodePro_Group']).copy()

    if not df_filtered_jobsat.empty:
        plt.figure(figsize=(14, 8))
        sns.histplot(data=df_filtered_jobsat, x='JobSat', hue='YearsCodePro_Group',
                     multiple='dodge', shrink=.8, bins=5, palette='Spectral')
        plt.title('Distribution of Job Satisfaction by Years of Professional Coding Experience')
        plt.xlabel('Job Satisfaction Score')
        plt.ylabel('Frequency')
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()
        print("Histogram of Job Satisfaction by YearsCodePro plotted.")
    else:
        print("Skipping 4.2: Not enough valid data in 'JobSat' or 'YearsCodePro' for comparison.")
else:
    print("Skipping 4.2: 'JobSat' or 'YearsCodePro' column not found or not numeric.")
```

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

Final step: Close the database connection

Once you've completed the lab, make sure to close the connection to the SQLite database:

In [29]:
```
print("\n--- Final Step: Close the Database Connection ---")
if conn:
    try:
```

```
        conn.close()
        print("Database connection closed successfully.")
    except Exception as e:
        print(f"Error closing database connection: {e}")
else:
    print("No active database connection to close.")

# --- Summary ---
print("\n--- Lab Summary ---")
print("In this lab, you used histograms and appropriate bar charts to visualize various aspects of
print("• Distribution of compensation, coding experience, and work hours.")
print("• Relationships in compensation across age groups and work status.")
print("• Composition of data by desired databases and work environments.")
print("• Comparisons of job satisfaction across years of experience.")
print("Histograms and bar charts helped reveal patterns and distributions in the data, enhancing yo
```

```
--- Final Step: Close the Database Connection ---
Database connection closed successfully.

--- Lab Summary ---
In this lab, you used histograms and appropriate bar charts to visualize various aspects of the data
set, focusing on:
• Distribution of compensation, coding experience, and work hours.
• Relationships in compensation across age groups and work status.
• Composition of data by desired databases and work environments.
• Comparisons of job satisfaction across years of experience.
Histograms and bar charts helped reveal patterns and distributions in the data, enhancing your under
standing of developer demographics and preferences.
```

Summary

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

- Distribution of compensation, coding experience, and work hours.
- Relationships in compensation across age groups and work status.
- Composition of data by desired databases and work environments.
- Comparisons of job satisfaction across years of experience.

Histograms helped reveal patterns and distributions in the data, enhancing your understanding of developer demographics and preferences.

Authors:

Ayushi Jain

Other Contributors:

- Rav Ahuja
- Lakshmi Holla
- Malika