Stacked Charts

Estimated time needed: **45** minutes

In this lab, you will focus on visualizing data specifically using stacked charts. You will use SQL queries to extract the necessary data and apply stacked charts to analyze the composition and comparison within the data.

Objectives

In this lab, you will perform the following:

- Visualize the composition of data using stacked charts.
- Compare multiple variables across different categories using stacked charts.
- Analyze trends within stacked chart visualizations.

Setup: Downloading and Loading the Data

Install the libraries

In [1]:
```
!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 [2]:
```
!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 [3]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np # For NaN handling and dummy data

# --- Setup: Download and Load the Data ---
print("--- Setup: Downloading and Loading the Data ---")

# The PDF specifies downloading 'survey-data.csv'
file_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/n01PQ9pSmiRX6520fluj
local_file_name = 'survey-data.csv'

try:
    df = pd.read_csv(file_url, na_values=['NA', 'N/A', 'nan', 'NaN', 'null', 'Null', '', ' ', '-'])
    print(f"Dataset loaded successfully from: {file_url}")
    print(f"Initial DataFrame shape: {df.shape}")
    print(f"Initial DataFrame columns: {df.columns.tolist()}")
except Exception as e:
    print(f"ERROR: Could not load dataset from URL: {e}")
    print(f"Creating a dummy DataFrame for demonstration purposes as '{local_file_name}' was not fo
    # Create a dummy DataFrame if download fails
    num_rows_dummy = 200
    data = {
        'ResponseId': range(1, num_rows_dummy + 1),
        'Age': np.random.choice(['Under 18 years old', '18-24 years old', '25-34 years old', '35-44
        'JobSatPoints_6': np.random.randint(1, 6, size=num_rows_dummy), # 1-5 scale for satisfactio
        'JobSatPoints_7': np.random.randint(1, 6, size=num_rows_dummy), # Another JobSat score
        'Employment': np.random.choice(['Employed, full-time', 'Employed, part-time', 'Student', 'I
        'ConvertedCompYearly': np.random.normal(loc=90000, scale=40000, size=num_rows_dummy),
        'DatabaseWantToWorkWith': [';'.join(np.random.choice(['MySQL', 'PostgreSQL', 'MongoDB', 'Re
        'LanguageAdmired': [';'.join(np.random.choice(['Python', 'Rust', 'Go', 'TypeScript', 'C++',
        'PlatformAdmired': [';'.join(np.random.choice(['AWS', 'Azure', 'Google Cloud', 'Heroku', 'D
    }
    df = pd.DataFrame(data)
    print("Dummy DataFrame created and populated with sample data.")

# --- Data Cleaning and Preprocessing ---
print("\n--- Data Cleaning and Preprocessing ---")

# Convert relevant numerical columns, coercing errors to NaN and filling with median
numeric_cols = ['ConvertedCompYearly', 'JobSatPoints_6', 'JobSatPoints_7']
for col in numeric_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
        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. Cannot i
            else:
                df[col].fillna(median_val, inplace=True)
                print(f"Cleaned '{col}': Imputed NaNs with median: {median_val:.2f}")
    else:
        print(f"WARNING: Numerical column '{col}' not found in DataFrame.")

# Map 'Age' to a numeric approximation for grouping/ordering
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():
        median_age_numeric = df['Age_Numeric'].median()
        if pd.isna(median_age_numeric):
            print("WARNING: 'Age_Numeric' column is entirely NaN. Cannot impute median.")
        else:
            df['Age_Numeric'].fillna(median_age_numeric, inplace=True)
            print("Created and imputed 'Age_Numeric' column.")
else:
    print("WARNING: 'Age' column not found, 'Age_Numeric' will not be created.")

# Define a custom sorting key function for age categories
def get_age_sort_key(age_str):
```

```python
    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

# Clean categorical columns (strip whitespace, replace common NaN strings)
categorical_cols_to_clean = [
    'Employment', 'DatabaseWantToWorkWith', 'LanguageAdmired', 'PlatformAdmired'
]
for col in categorical_cols_to_clean:
    if col in df.columns:
        df[col] = df[col].astype(str).str.strip()
        df[col].replace(['nan', 'NaN', 'N/A', 'None', '', ' '], np.nan, inplace=True)
    else:
        print(f"WARNING: Categorical column '{col}' not found in DataFrame for cleaning.")
```

```
--- Setup: Downloading and Loading the Data ---
Dataset loaded successfully from: https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clou
d/n01PQ9pSmiRX6520flujwQ/survey-data.csv
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']

--- Data Cleaning and Preprocessing ---
Cleaned 'ConvertedCompYearly': Imputed NaNs with median: 65000.00
Cleaned 'JobSatPoints_6': Imputed NaNs with median: 20.00
Cleaned 'JobSatPoints_7': Imputed NaNs with median: 15.00
Created and imputed 'Age_Numeric' column.
```

```
/tmp/ipykernel_953/2545220165.py:50: FutureWarning: A value is trying to be set on a copy of a DataFrame 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}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].fillna(median_val, inplace=True)
/tmp/ipykernel_953/2545220165.py:50: FutureWarning: A value is trying to be set on a copy of a DataFrame 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}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].fillna(median_val, inplace=True)
/tmp/ipykernel_953/2545220165.py:50: FutureWarning: A value is trying to be set on a copy of a DataFrame 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}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].fillna(median_val, inplace=True)
/tmp/ipykernel_953/2545220165.py:68: FutureWarning: A value is trying to be set on a copy of a DataFrame 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}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['Age_Numeric'].fillna(median_age_numeric, inplace=True)
/tmp/ipykernel_953/2545220165.py:92: FutureWarning: A value is trying to be set on a copy of a DataFrame 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}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].replace(['nan', 'NaN', 'N/A', 'None', '', ' '], np.nan, inplace=True)
/tmp/ipykernel_953/2545220165.py:92: FutureWarning: A value is trying to be set on a copy of a DataFrame 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}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].replace(['nan', 'NaN', 'N/A', 'None', '', ' '], np.nan, inplace=True)
/tmp/ipykernel_953/2545220165.py:92: FutureWarning: A value is trying to be set on a copy of a DataFrame 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}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.
```

```
  df[col].replace(['nan', 'NaN', 'N/A', 'None', '', ' '], np.nan, inplace=True)
/tmp/ipykernel_953/2545220165.py:92: 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].replace(['nan', 'NaN', 'N/A', 'None', '', ' '], np.nan, inplace=True)
```

Download and Load the Data

To start, download and load the dataset into a `pandas` DataFrame.

Step 1: Download the dataset

In [4]:
```
!wget -O survey-data.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/n01PQ9p
```

```
--2025-06-18 17:58:39--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/n01PQ9pS
miRX6520flujwQ/survey-data.csv
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.
HTTP request sent, awaiting response... 200 OK
Length: 159525875 (152M) [text/csv]
Saving to: ‘survey-data.csv’

survey-data.csv     100%[===================>] 152.13M  51.1MB/s    in 3.0s

2025-06-18 17:58:43 (51.1 MB/s) - ‘survey-data.csv’ saved [159525875/159525875]
```

Step 2: Import necessary libraries and load the dataset

In [5]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

Load the data

In [6]:
```
df = pd.read_csv("survey-data.csv")
```

Display the first few rows of the data to understand its structure

In [7]:
```
df.head()
```

Out[7]:

	ResponseId	MainBranch	Age	Employment	RemoteWork	Check	CodingActivities	EdLevel	
0	1	I am a developer by profession	Under 18 years old	Employed, full-time	Remote	Apples	Hobby	Primary/elementary school	E
1	2	I am a developer by profession	35-44 years old	Employed, full-time	Remote	Apples	Hobby;Contribute to open-source projects;Other...	Bachelor's degree (B.A., B.S., B.Eng., etc.)	E medi
2	3	I am a developer by profession	45-54 years old	Employed, full-time	Remote	Apples	Hobby;Contribute to open-source projects;Other...	Master's degree (M.A., M.S., M.Eng., MBA, etc.)	E medi
3	4	I am learning to code	18-24 years old	Student, full-time	NaN	Apples	NaN	Some college/university study without earning ...	vi
4	5	I am a developer by profession	18-24 years old	Student, full-time	NaN	Apples	NaN	Secondary school (e.g. American high school, G...	vi

5 rows × 114 columns

Task 1: Stacked Chart for Composition of Job Satisfaction Across Age Groups

1. Stacked Chart of Median `JobSatPoints_6` and `JobSatPoints_7` for Different Age Groups

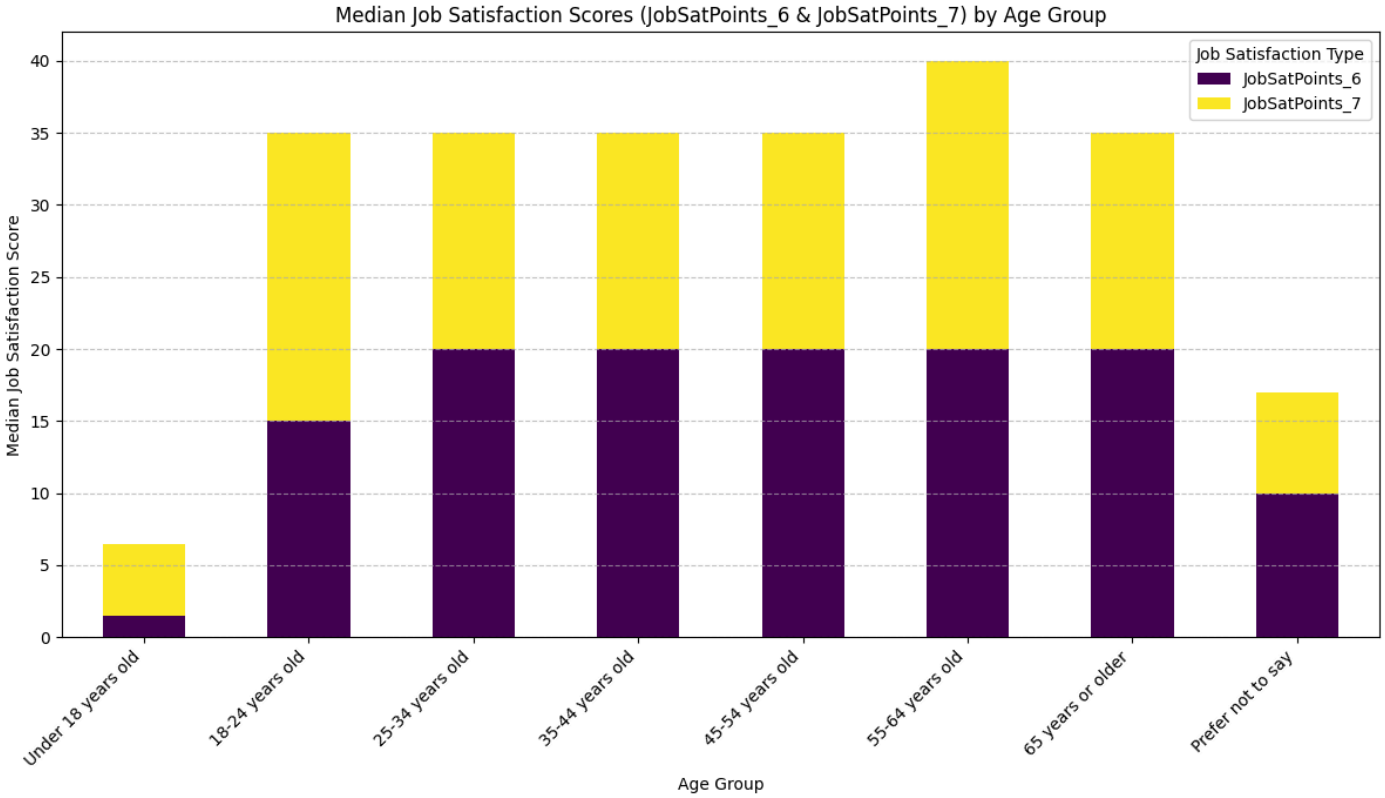
Visualize the composition of job satisfaction scores (`JobSatPoints_6` and `JobSatPoints_7`) across various age groups. This will help in understanding the breakdown of satisfaction levels across different demographics.

In [8]:

```
##Write your code here
# --- Task 1: Stacked Chart for Composition of Job Satisfaction Across Age Groups ---
print("\n--- Task 1: Stacked Chart for Composition of Job Satisfaction Across Age Groups ---")

# 1. Stacked Chart of Median JobSatPoints_6 and JobSatPoints_7 for Different Age Groups
if all(col in df.columns for col in ['JobSatPoints_6', 'JobSatPoints_7', 'Age']):
    df_task1_1 = df.dropna(subset=['JobSatPoints_6', 'JobSatPoints_7', 'Age']).copy()

    if not df_task1_1.empty:
        # Group by Age and calculate median for both JobSat columns
        # Filter for age groups that actually have data
        valid_age_groups = df_task1_1['Age'].unique().tolist()
        age_order = sorted(valid_age_groups, key=get_age_sort_key)

        # Melt the DataFrame to have JobSat scores in one column
        df_melted_jobsat = df_task1_1.melt(
            id_vars=['Age'],
            value_vars=['JobSatPoints_6', 'JobSatPoints_7'],
            var_name='JobSatisfactionType',
            value_name='JobSatisfactionScore'
        )

        # Calculate median for plotting
        median_jobsat = df_melted_jobsat.groupby(['Age', 'JobSatisfactionType'])['JobSatisfactionSc

        if not median_jobsat.empty:
            median_jobsat = median_jobsat.reindex(age_order) # Ensure correct age order

            # Plotting the stacked bar chart of medians
            ax = median_jobsat.plot(kind='bar', stacked=True, figsize=(12, 7), colormap='viridis')
            plt.title('Median Job Satisfaction Scores (JobSatPoints_6 & JobSatPoints_7) by Age Grou
            plt.xlabel('Age Group')
            plt.ylabel('Median Job Satisfaction Score')
            plt.xticks(rotation=45, ha='right')
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.legend(title='Job Satisfaction Type')
            plt.tight_layout()
```

```
            plt.show()
            print("Task 1.1: Stacked chart of median JobSatPoints_6 and JobSatPoints_7 by Age Group
        else:
            print("Skipping Task 1.1: Not enough valid grouped data for Job Satisfaction by Age Gro
    else:
        print("Skipping Task 1.1: Not enough valid data in 'JobSatPoints_6', 'JobSatPoints_7', or '
else:
    print("Skipping Task 1.1: Required columns missing or not ready ('JobSatPoints_6', 'JobSatPoint
```

```
--- Task 1: Stacked Chart for Composition of Job Satisfaction Across Age Groups ---
```




```
Task 1.1: Stacked chart of median JobSatPoints_6 and JobSatPoints_7 by Age Group plotted.
```

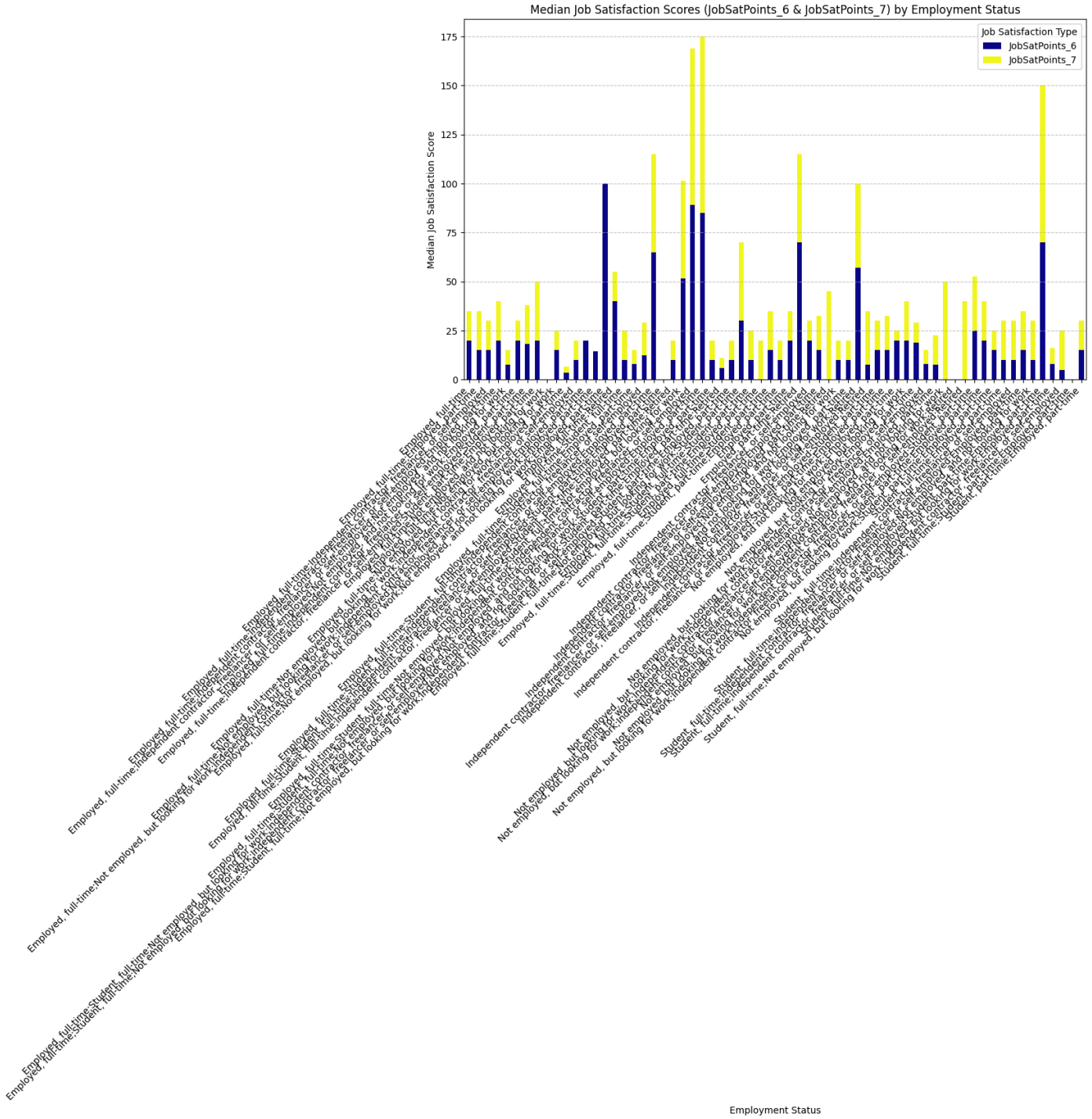
Stacked Chart of `JobSatPoints_6` and `JobSatPoints_7` for Employment Status

Create a stacked chart to compare job satisfaction (`JobSatPoints_6` and `JobSatPoints_7`) across different employment statuses. This will show how satisfaction varies by employment type.

In [9]:
```
##Write your code here
# 2. Stacked Chart of JobSatPoints_6 and JobSatPoints_7 for Employment Status
if all(col in df.columns for col in ['JobSatPoints_6', 'JobSatPoints_7', 'Employment']):
    df_task1_2 = df.dropna(subset=['JobSatPoints_6', 'JobSatPoints_7', 'Employment']).copy()

    if not df_task1_2.empty:
        # Get employment order for consistency (alphabetical or custom if needed)
        employment_order = sorted(df_task1_2['Employment'].unique().tolist())

        # Melt the DataFrame to have JobSat scores in one column
        df_melted_jobsat_emp = df_task1_2.melt(
            id_vars=['Employment'],
            value_vars=['JobSatPoints_6', 'JobSatPoints_7'],
            var_name='JobSatisfactionType',
            value_name='JobSatisfactionScore'
        )

        # Calculate median for plotting
        median_jobsat_emp = df_melted_jobsat_emp.groupby(['Employment', 'JobSatisfactionType'])['Jo

        if not median_jobsat_emp.empty:
            median_jobsat_emp = median_jobsat_emp.reindex(employment_order) # Ensure correct order

            # Plotting the stacked bar chart of medians
            ax = median_jobsat_emp.plot(kind='bar', stacked=True, figsize=(12, 7), colormap='plasma
            plt.title('Median Job Satisfaction Scores (JobSatPoints_6 & JobSatPoints_7) by Employme
            plt.xlabel('Employment Status')
            plt.ylabel('Median Job Satisfaction Score')
            plt.xticks(rotation=45, ha='right')
            plt.grid(axis='y', linestyle='--', alpha=0.7)
```

```
            plt.legend(title='Job Satisfaction Type')
            plt.tight_layout()
            plt.show()
            print("Task 1.2: Stacked chart of median JobSatPoints_6 and JobSatPoints_7 by Employmen
        else:
            print("Skipping Task 1.2: Not enough valid grouped data for Job Satisfaction by Employm
    else:
        print("Skipping Task 1.2: Not enough valid data in 'JobSatPoints_6', 'JobSatPoints_7', or '
else:
    print("Skipping Task 1.2: Required columns missing or not ready ('JobSatPoints_6', 'JobSatPoint
```

```
/tmp/ipykernel_953/1185071150.py:32: UserWarning: Tight layout not applied. The bottom and top margi
ns cannot be made large enough to accommodate all Axes decorations.
  plt.tight_layout()
```




```
Task 1.2: Stacked chart of median JobSatPoints_6 and JobSatPoints_7 by Employment Status plotted.
```

Task 2: Stacked Chart for Compensation and Job Satisfaction by Age Group

This stacked chart visualizes the composition of compensation (`ConvertedCompYearly`) and job satisfaction (`JobSatPoints_6`) specifically for respondents aged 30-35.

In [10]:
```
##Write your code here
# --- Task 2: Stacked Chart for Compensation and Job Satisfaction by Age Group ---
print("\n--- Task 2: Stacked Chart for Compensation and Job Satisfaction by Age Group ---")
```

```
# 1. This implies a combined view for a specific age range (30-35).
if all(col in df.columns for col in ['ConvertedCompYearly', 'JobSatPoints_6', 'Age_Numeric', 'Age']
    df_task2_1_specific_age = df[(df['Age_Numeric'] >= 30) & (df['Age_Numeric'] <= 35)].dropna(
        subset=['ConvertedCompYearly', 'JobSatPoints_6', 'Age']
    ).copy()

    if not df_task2_1_specific_age.empty:
        # Group by actual 'Age' string if there are variations within 30-35, and get medians
        grouped_data = df_task2_1_specific_age.groupby('Age').agg(
            Median_Compensation=('ConvertedCompYearly', 'median'),
            Median_JobSat=('JobSatPoints_6', 'median')
        ).unstack() # Use unstack to prepare for stacked bar chart

        if not grouped_data.empty:
            # Re-order columns for consistent stacking if necessary
            grouped_data = grouped_data[['Median_Compensation', 'Median_JobSat']]

            ax = grouped_data.plot(kind='bar', stacked=True, figsize=(10, 6), colormap='coolwarm')
            plt.title('Median Compensation and Job Satisfaction (JobSatPoints_6) for Ages 30-35')
            plt.xlabel('Age Group (30-35 range)')
            plt.ylabel('Median Value')
            plt.xticks(rotation=45, ha='right')
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.legend(title='Metric')
            plt.tight_layout()
            plt.show()
            print("Task 2.1: Stacked chart for Median Compensation and Job Satisfaction for Ages 30
        else:
            print("Skipping Task 2.1: Not enough valid grouped data for specific age range (30-35).
    else:
        print("Skipping Task 2.1: Not enough valid data in 'ConvertedCompYearly', 'JobSatPoints_6',
else:
    print("Skipping Task 2.1: Required columns missing or not ready ('ConvertedCompYearly', 'JobSat
```

```
--- Task 2: Stacked Chart for Compensation and Job Satisfaction by Age Group ---
Skipping Task 2.1: Required columns missing or not ready ('ConvertedCompYearly', 'JobSatPoints_6',
'Age_Numeric', 'Age').
```

Stacked Chart of Median Compensation and Job Satisfaction Across Age Group

Compare the median compensation and job satisfaction metrics across different age groups. This helps visualize how compensation and satisfaction levels differ by age.

In [11]:

```
##Write your code here
# 2. Stacked Chart of Median Compensation and Job Satisfaction Across All Age Groups
if all(col in df.columns for col in ['ConvertedCompYearly', 'JobSatPoints_6', 'Age']):
    df_task2_2 = df.dropna(subset=['ConvertedCompYearly', 'JobSatPoints_6', 'Age']).copy()

    if not df_task2_2.empty:
        # Filter for age groups that actually have data
        valid_age_groups = df_task2_2['Age'].unique().tolist()
        age_order = sorted(valid_age_groups, key=get_age_sort_key)

        # Group by Age and calculate median for both compensation and JobSat
        grouped_data_all_ages = df_task2_2.groupby('Age').agg(
            Median_Compensation=('ConvertedCompYearly', 'median'),
            Median_JobSat=('JobSatPoints_6', 'median')
        ).unstack() # Unstack to prepare for stacked bar chart

        if not grouped_data_all_ages.empty:
            # Flatten multi-index columns for easier plotting
            grouped_data_all_ages.columns = ['_'.join(col).strip() for col in grouped_data_all_ages

            # Reindex to ensure correct age order
            grouped_data_all_ages = grouped_data_all_ages.reindex(age_order)

            ax = grouped_data_all_ages.plot(kind='bar', stacked=True, figsize=(12, 7), colormap='Sp
            plt.title('Median Compensation and Job Satisfaction (JobSatPoints_6) Across All Age Gro
            plt.xlabel('Age Group')
            plt.ylabel('Median Value')
            plt.xticks(rotation=45, ha='right')
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.legend(title='Metric')
            plt.tight_layout()
```

```
            plt.show()
            print("Task 2.2: Stacked chart of Median Compensation and Job Satisfaction Across All A
        else:
            print("Skipping Task 2.2: Not enough valid grouped data for Compensation and Job Satisf
    else:
        print("Skipping Task 2.2: Not enough valid data in 'ConvertedCompYearly', 'JobSatPoints_6',
else:
    print("Skipping Task 2.2: Required columns missing or not ready ('ConvertedCompYearly', 'JobSat
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipykernel_953/360306759.py in ?()
     15         ).unstack() # Unstack to prepare for stacked bar chart
     16
     17         if not grouped_data_all_ages.empty:
     18             # Flatten multi-index columns for easier plotting
---> 19             grouped_data_all_ages.columns = ['_'.join(col).strip() for col in grouped_data_a
ll_ages.columns.values]
     20
     21             # Reindex to ensure correct age order
     22             grouped_data_all_ages = grouped_data_all_ages.reindex(age_order)

/opt/conda/lib/python3.12/site-packages/pandas/core/generic.py in ?(self, name)
   6314             and name not in self._accessors
   6315             and self._info_axis._can_hold_identifiers_and_holds_name(name)
   6316         ):
   6317             return self[name]
-> 6318         return object.__getattribute__(self, name)

AttributeError: 'Series' object has no attribute 'columns'
```

Task 3: Comparing Data Using Stacked Charts

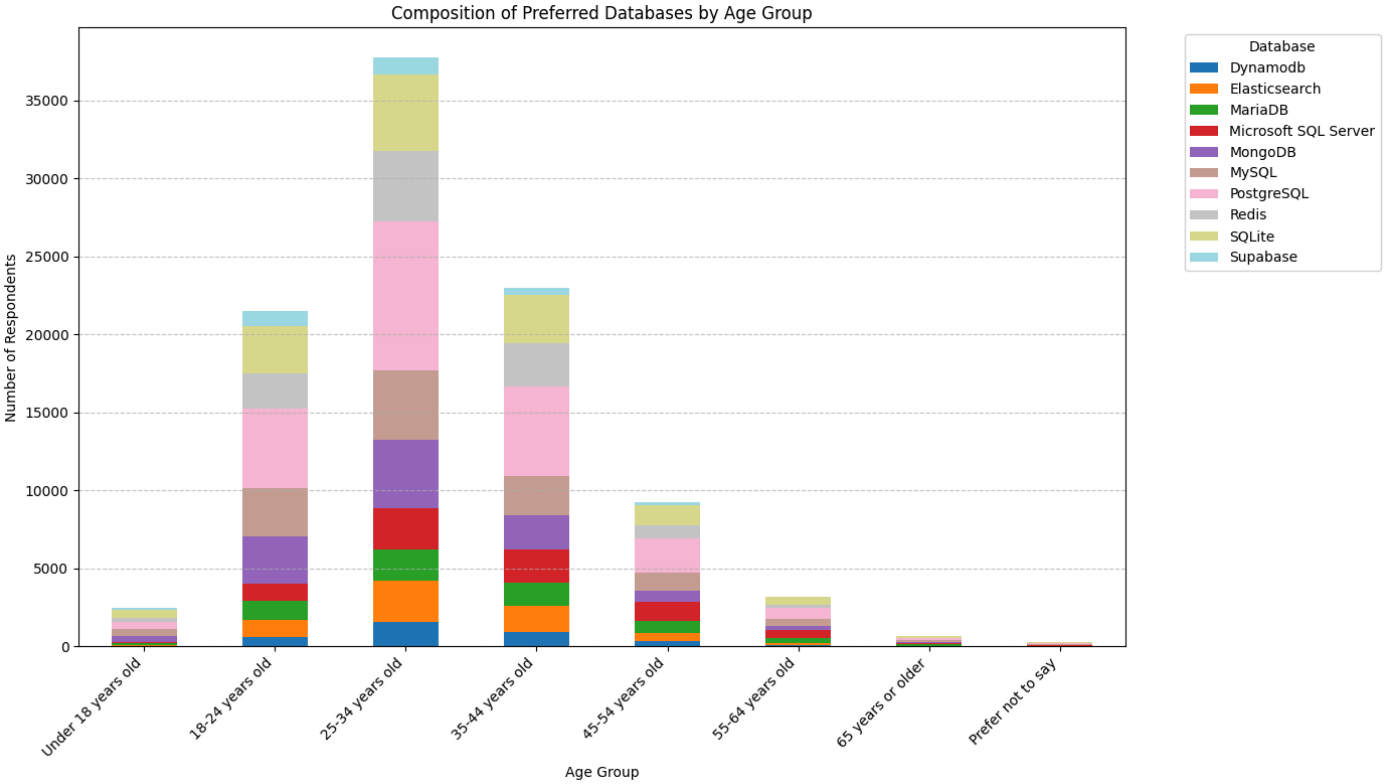
1. Stacked Chart of Preferred Databases by Age Group

Visualize the top databases that respondents from different age groups wish to learn. Create a stacked chart to show the proportion of each database in each age group.

In [12]:
```
##Write your code here
# --- Task 3: Comparing Data Using Stacked Charts ---
print("\n--- Task 3: Comparing Data Using Stacked Charts ---")

# 1. Stacked Chart of Preferred Databases by Age Group
if 'DatabaseWantToWorkWith' in df.columns and 'Age' in df.columns:
    df_task3_1 = df.dropna(subset=['DatabaseWantToWorkWith', 'Age']).copy()
    if not df_task3_1.empty:
        # Explode databases
        df_task3_1['Database'] = df_task3_1['DatabaseWantToWorkWith'].str.split(';')
        df_exploded_databases = df_task3_1.explode('Database')
        df_exploded_databases['Database'] = df_exploded_databases['Database'].str.strip()

        # Get top N databases for readability
        top_databases = df_exploded_databases['Database'].value_counts().head(10).index.tolist()
        df_top_databases = df_exploded_databases[df_exploded_databases['Database'].isin(top_databas

        # Group by age and database, then unstack to prepare for stacked bar chart
        database_composition = df_top_databases.groupby(['Age', 'Database']).size().unstack(fill_va

        if not database_composition.empty:
            valid_age_groups = database_composition.index.unique().tolist()
            age_order = sorted(valid_age_groups, key=get_age_sort_key)
            database_composition = database_composition.reindex(age_order)

            ax = database_composition.plot(kind='bar', stacked=True, figsize=(14, 8), colormap='tab
            plt.title('Composition of Preferred Databases by Age Group')
            plt.xlabel('Age Group')
            plt.ylabel('Number of Respondents')
            plt.xticks(rotation=45, ha='right')
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.legend(title='Database', bbox_to_anchor=(1.05, 1), loc='upper left')
            plt.tight_layout()
            plt.show()
            print("Task 3.1: Stacked chart of Preferred Databases by Age Group plotted.")
```

```
        else:
            print("Skipping Task 3.1: Not enough valid grouped data for Preferred Databases by Age
    else:
        print("Skipping Task 3.1: Not enough valid data in 'DatabaseWantToWorkWith' or 'Age' for pl
else:
    print("Skipping Task 3.1: Required columns missing or not ready ('DatabaseWantToWorkWith', 'Age
```

```
--- Task 3: Comparing Data Using Stacked Charts ---
```



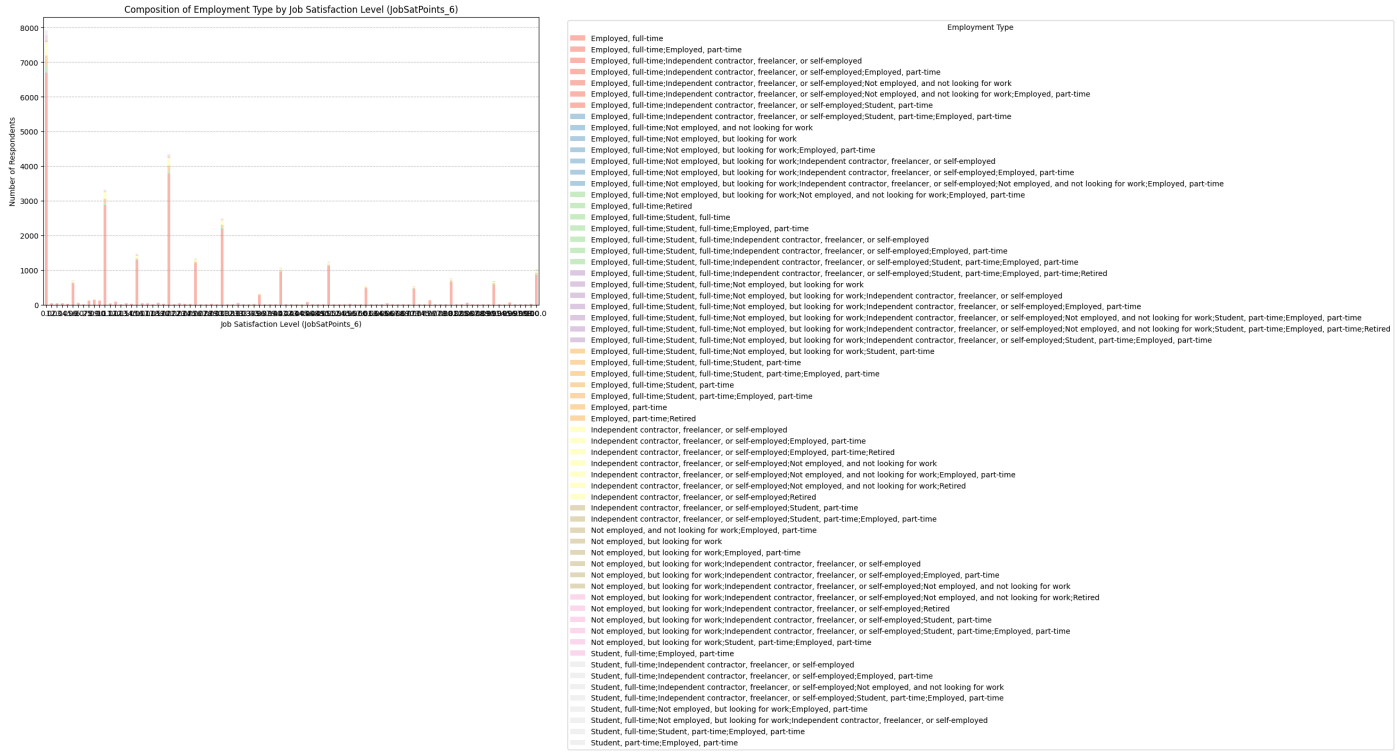

```
Task 3.1: Stacked chart of Preferred Databases by Age Group plotted.
```

2. Stacked Chart of Employment Type by Job Satisfaction

Analyze the distribution of employment types within each job satisfaction level using a stacked chart. This will provide insights into how employment types are distributed across various satisfaction ratings.

In [13]:
```
##Write your code here
# 2. Stacked Chart of Employment Type by Job Satisfaction
if 'Employment' in df.columns and 'JobSatPoints_6' in df.columns:
    df_task3_2 = df.dropna(subset=['Employment', 'JobSatPoints_6']).copy()
    if not df_task3_2.empty:
        # Group by Job Satisfaction and Employment Type
        employment_composition = df_task3_2.groupby(['JobSatPoints_6', 'Employment']).size().unstac

        if not employment_composition.empty:
            # Sort JobSat values for consistent order
            jobsat_order = sorted(employment_composition.index.unique().tolist())
            employment_composition = employment_composition.reindex(jobsat_order)

            ax = employment_composition.plot(kind='bar', stacked=True, figsize=(12, 7), colormap='P
            plt.title('Composition of Employment Type by Job Satisfaction Level (JobSatPoints_6)')
            plt.xlabel('Job Satisfaction Level (JobSatPoints_6)')
            plt.ylabel('Number of Respondents')
            plt.xticks(rotation=0) # JobSat is typically numerical or ordered, so no rotation neede
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.legend(title='Employment Type', bbox_to_anchor=(1.05, 1), loc='upper left')
            plt.tight_layout()
            plt.show()
            print("Task 3.2: Stacked chart of Employment Type by Job Satisfaction plotted.")
        else:
            print("Skipping Task 3.2: Not enough valid grouped data for Employment Type by Job Sati
    else:
        print("Skipping Task 3.2: Not enough valid data in 'Employment' or 'JobSatPoints_6' for plo
else:
    print("Skipping Task 3.2: Required columns missing or not ready ('Employment', 'JobSatPoints_6'
```

```
/tmp/ipykernel_953/2448813710.py:21: UserWarning: Tight layout not applied. The left and right margi
ns cannot be made large enough to accommodate all Axes decorations.
  plt.tight_layout()
```



Task 3.2: Stacked chart of Employment Type by Job Satisfaction plotted.

Task 4: Exploring Technology Preferences Using Stacked Charts

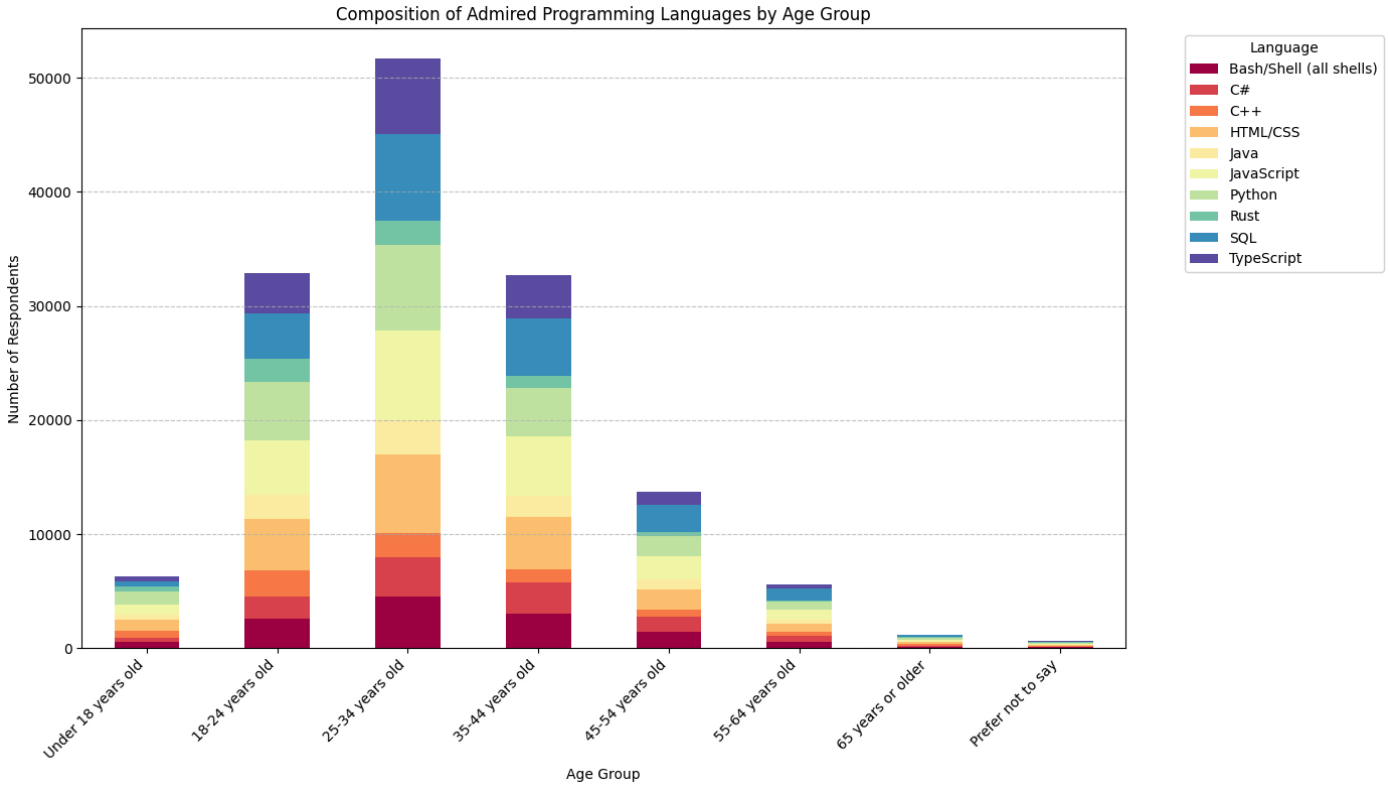
1. Stacked Chart for Preferred Programming Languages by Age Group

Analyze how programming language preferences (`LanguageAdmired`) vary across age groups.

In [14]:
```
##Write your code here
# --- Task 4: Exploring Technology Preferences Using Stacked Charts ---
print("\n--- Task 4: Exploring Technology Preferences Using Stacked Charts ---")

# 1. Stacked Chart for Preferred Programming Languages (LanguageAdmired) by Age Group
if 'LanguageAdmired' in df.columns and 'Age' in df.columns:
    df_task4_1 = df.dropna(subset=['LanguageAdmired', 'Age']).copy()
    if not df_task4_1.empty:
        # Explode admired languages
        df_task4_1['Language'] = df_task4_1['LanguageAdmired'].str.split(';')
        df_exploded_languages = df_task4_1.explode('Language')
        df_exploded_languages['Language'] = df_exploded_languages['Language'].str.strip()

        # Get top N languages
        top_languages = df_exploded_languages['Language'].value_counts().head(10).index.tolist()
        df_top_languages = df_exploded_languages[df_exploded_languages['Language'].isin(top_languag

        # Group by Age and Language, then unstack for plotting
        language_composition = df_top_languages.groupby(['Age', 'Language']).size().unstack(fill_va

        if not language_composition.empty:
            valid_age_groups = language_composition.index.unique().tolist()
            age_order = sorted(valid_age_groups, key=get_age_sort_key)
            language_composition = language_composition.reindex(age_order)

            ax = language_composition.plot(kind='bar', stacked=True, figsize=(14, 8), colormap='Spe
            plt.title('Composition of Admired Programming Languages by Age Group')
            plt.xlabel('Age Group')
            plt.ylabel('Number of Respondents')
            plt.xticks(rotation=45, ha='right')
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.legend(title='Language', bbox_to_anchor=(1.05, 1), loc='upper left')
            plt.tight_layout()
            plt.show()
            print("Task 4.1: Stacked chart for Preferred Programming Languages by Age Group plotted
        else:
```

```
            print("Skipping Task 4.1: Not enough valid grouped data for Preferred Programming Langu
    else:
        print("Skipping Task 4.1: Not enough valid data in 'LanguageAdmired' or 'Age' for plotting.
else:
    print("Skipping Task 4.1: Required columns missing or not ready ('LanguageAdmired', 'Age').")
```

```
--- Task 4: Exploring Technology Preferences Using Stacked Charts ---
```




```
Task 4.1: Stacked chart for Preferred Programming Languages by Age Group plotted.
```

2. Stacked Chart for Technology Adoption by Employment Type

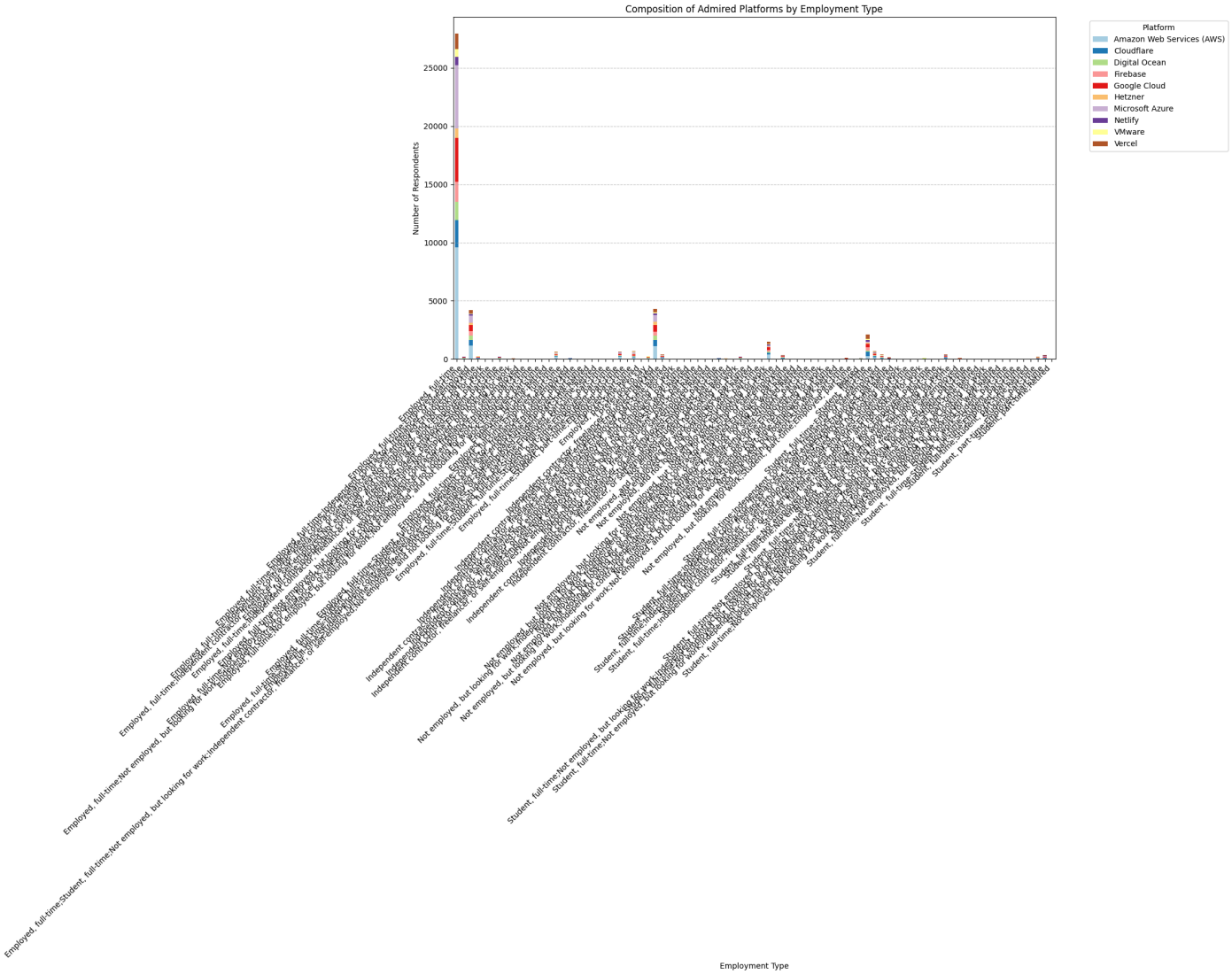
Explore how admired platforms (`PlatformAdmired`) differ across employment types (e.g., full-time, freelance)

In [15]:

```
##Write your code here
# 2. Stacked Chart for Technology Adoption (PlatformAdmired) by Employment Type
if 'PlatformAdmired' in df.columns and 'Employment' in df.columns:
    df_task4_2 = df.dropna(subset=['PlatformAdmired', 'Employment']).copy()
    if not df_task4_2.empty:
        # Explode admired platforms
        df_task4_2['Platform'] = df_task4_2['PlatformAdmired'].str.split(';')
        df_exploded_platforms = df_task4_2.explode('Platform')
        df_exploded_platforms['Platform'] = df_exploded_platforms['Platform'].str.strip()

        # Get top N platforms
        top_platforms = df_exploded_platforms['Platform'].value_counts().head(10).index.tolist()
        df_top_platforms = df_exploded_platforms[df_exploded_platforms['Platform'].isin(top_platfor

        # Group by Employment and Platform, then unstack for plotting
        platform_composition = df_top_platforms.groupby(['Employment', 'Platform']).size().unstack(

        if not platform_composition.empty:
            employment_order = sorted(platform_composition.index.unique().tolist())
            platform_composition = platform_composition.reindex(employment_order)

            ax = platform_composition.plot(kind='bar', stacked=True, figsize=(14, 8), colormap='Pai
            plt.title('Composition of Admired Platforms by Employment Type')
            plt.xlabel('Employment Type')
            plt.ylabel('Number of Respondents')
            plt.xticks(rotation=45, ha='right')
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.legend(title='Platform', bbox_to_anchor=(1.05, 1), loc='upper left')
            plt.tight_layout()
            plt.show()
            print("Task 4.2: Stacked chart for Technology Adoption by Employment Type plotted.")
        else:
            print("Skipping Task 4.2: Not enough valid grouped data for Technology Adoption by Empl
    else:
        print("Skipping Task 4.2: Not enough valid data in 'PlatformAdmired' or 'Employment' for pl
```

```
else:
    print("Skipping Task 4.2: Required columns missing or not ready ('PlatformAdmired', 'Employment

print("\n--- Lab Completion Summary ---")
print("All stacked chart tasks have been attempted. Please review the plots and console output for
```

```
/tmp/ipykernel_953/4149106411.py:29: UserWarning: Tight layout not applied. The bottom and top margi
ns cannot be made large enough to accommodate all Axes decorations.
  plt.tight_layout()
```




```
Task 4.2: Stacked chart for Technology Adoption by Employment Type plotted.

--- Lab Completion Summary ---
All stacked chart tasks have been attempted. Please review the plots and console output for any warn
ings or skipped tasks.
```

Final Step: Review

In this lab, you focused on using stacked charts to understand the composition and comparison within the dataset. Stacked charts provided insights into job satisfaction, compensation, and preferred databases across age groups and employment types.

Summary

After completing this lab, you will be able to:

- Use stacked charts to analyze the composition of data across categories, such as job satisfaction and compensation by age group.
- Compare data across different dimensions using stacked charts, enhancing your ability to communicate complex relationships in the data.

- Visualize distributions across multiple categories, such as employment type by satisfaction, to gain a deeper understanding of patterns within the dataset.

Author:

Ayushi Jain

Other Contributors:

- Rav Ahuja
- Lakshmi Holla
- Malika

<!-- ## Change Log |Date (YYYY-MM-DD)|Version|Changed By|Change Description| |-|-|-|-| |2024-10-28|1.2|Madhusudhan Moole|Updated lab| |2024-10-16|1.1|Madhusudhan Moole|Updated lab| |2024-10-15|1.0|Raghul Ramesh|Created lab| --!>