Project_survey_data_documented

June 12, 2025

1 Exploratory Data Analysis of Developer Survey Results

Author: Arturo Alejandro Díaz Barbosa
Date: June, 2025

Data Source:
Stack Overflow Developer Survey 2024
(https://survey.stackoverflow.co/2024/)

1.1 Dataset Description

Stack Overflow, the go-to platform for developers, conducted a global survey to capture insights into the developer community. This survey includes various questions related to professional experience, coding activities, tools, technologies, and preferences. The dataset offers important information about the current state of software development worldwide and covers topics such as demographics, salary, education, career paths, and technology usage.

1.2 Objective

This project aims to explore the responses from the 2024 Stack Overflow Developer Survey to answer key questions such as:
1. How does average salary vary with years of experience?
2. Which programming languages are most used by region?
3. Is there a correlation between education level and annual salary?

1.3 Key Findings

- **Salary vs. Experience:** Median salary increases by 156% after 5 years of experience.

- **Top Languages:** JavaScript commands 11.6% usage among developers
- **Education Level and Annual Salary:** Formal education boosts earning potential by up to 60%.

1.4 *Import required libraries*

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.preprocessing import MinMaxScaler
```

1.5 Initial Data Loading and Cleaning

- Load raw survey data from CSV and preview its structure and size.
- Extract a subset of relevant columns for analysis.
- Explore the data with summary statistics and data types.
- Check for missing values and duplicated rows.
- Remove duplicates to ensure data quality before further analysis.

Load CSV data into DataFrame

[2]:
```
df_raw = pd.read_csv('survey_data.csv', low_memory=False) # Read CSV into␣
↪DataFrame
df_raw.head() # Show first 5 rows of DataFrame
```

[2]:

	ResponseId	MainBranch	Age	\
0	1	I am a developer by profession	Under 18 years old	
1	2	I am a developer by profession	35-44 years old	
2	3	I am a developer by profession	45-54 years old	
3	4	I am learning to code	18-24 years old	
4	5	I am a developer by profession	18-24 years old	

	Employment	RemoteWork	Check	\
0	Employed, full-time	Remote	Apples	
1	Employed, full-time	Remote	Apples	
2	Employed, full-time	Remote	Apples	
3	Student, full-time	NaN	Apples	
4	Student, full-time	NaN	Apples	

	CodingActivities	\
0	Hobby	
1	Hobby;Contribute to open-source projects;Other…	
2	Hobby;Contribute to open-source projects;Other…	
3	NaN	
4	NaN	

	EdLevel	\
0	Primary/elementary school	
1	Bachelor’s degree (B.A., B.S., B.Eng., etc.)	
2	Master’s degree (M.A., M.S., M.Eng., MBA, etc.)	
3	Some college/university study without earning …	

```
4  Secondary school (e.g. American high school, G…

                                          LearnCode  \
0                             Books / Physical media
1  Books / Physical media;Colleague;On the job tr…
2  Books / Physical media;Colleague;On the job tr…
3  Other online resources (e.g., videos, blogs, f…
4  Other online resources (e.g., videos, blogs, f…

                                    LearnCodeOnline  … JobSatPoints_6  \
0                                               NaN  …            NaN
1  Technical documentation;Blogs;Books;Written Tu…  …            0.0
2  Technical documentation;Blogs;Books;Written Tu…  …            NaN
3  Stack Overflow;How-to videos;Interactive tutorial  …            NaN
4  Technical documentation;Blogs;Written Tutorial…  …            NaN

  JobSatPoints_7 JobSatPoints_8 JobSatPoints_9 JobSatPoints_10  \
0            NaN            NaN            NaN             NaN
1            0.0            0.0            0.0             0.0
2            NaN            NaN            NaN             NaN
3            NaN            NaN            NaN             NaN
4            NaN            NaN            NaN             NaN

  JobSatPoints_11           SurveyLength SurveyEase ConvertedCompYearly JobSat
0             NaN                    NaN        NaN                 NaN    NaN
1             0.0                    NaN        NaN                 NaN    NaN
2             NaN  Appropriate in length       Easy                 NaN    NaN
3             NaN               Too long       Easy                 NaN    NaN
4             NaN             Too short        Easy                 NaN    NaN

[5 rows x 114 columns]
```

Returns the number of rows and columns in the DataFrame.

[3]:
```
df_raw.shape
```

[3]:
```
(65437, 114)
```

Returns a list of column names in the DataFrame.

[4]:
```
df_raw.columns.to_list()
```

[4]:
```
['ResponseId',
 'MainBranch',
 'Age',
 'Employment',
 'RemoteWork',
 'Check',
 'CodingActivities',
```

```
'EdLevel',
'LearnCode',
'LearnCodeOnline',
'TechDoc',
'YearsCode',
'YearsCodePro',
'DevType',
'OrgSize',
'PurchaseInfluence',
'BuyNewTool',
'BuildvsBuy',
'TechEndorse',
'Country',
'Currency',
'CompTotal',
'LanguageHaveWorkedWith',
'LanguageWantToWorkWith',
'LanguageAdmired',
'DatabaseHaveWorkedWith',
'DatabaseWantToWorkWith',
'DatabaseAdmired',
'PlatformHaveWorkedWith',
'PlatformWantToWorkWith',
'PlatformAdmired',
'WebframeHaveWorkedWith',
'WebframeWantToWorkWith',
'WebframeAdmired',
'EmbeddedHaveWorkedWith',
'EmbeddedWantToWorkWith',
'EmbeddedAdmired',
'MiscTechHaveWorkedWith',
'MiscTechWantToWorkWith',
'MiscTechAdmired',
'ToolsTechHaveWorkedWith',
'ToolsTechWantToWorkWith',
'ToolsTechAdmired',
'NEWCollabToolsHaveWorkedWith',
'NEWCollabToolsWantToWorkWith',
'NEWCollabToolsAdmired',
'OpSysPersonal use',
'OpSysProfessional use',
'OfficeStackAsyncHaveWorkedWith',
'OfficeStackAsyncWantToWorkWith',
'OfficeStackAsyncAdmired',
'OfficeStackSyncHaveWorkedWith',
'OfficeStackSyncWantToWorkWith',
'OfficeStackSyncAdmired',
```

```
'AISearchDevHaveWorkedWith',
'AISearchDevWantToWorkWith',
'AISearchDevAdmired',
'NEWSOSites',
'SOVisitFreq',
'SOAccount',
'SOPartFreq',
'SOHow',
'SOComm',
'AISelect',
'AISent',
'AIBen',
'AIAcc',
'AIComplex',
'AIToolCurrently Using',
'AIToolInterested in Using',
'AIToolNot interested in Using',
'AINextMuch more integrated',
'AINextNo change',
'AINextMore integrated',
'AINextLess integrated',
'AINextMuch less integrated',
'AIThreat',
'AIEthics',
'AIChallenges',
'TBranch',
'ICorPM',
'WorkExp',
'Knowledge_1',
'Knowledge_2',
'Knowledge_3',
'Knowledge_4',
'Knowledge_5',
'Knowledge_6',
'Knowledge_7',
'Knowledge_8',
'Knowledge_9',
'Frequency_1',
'Frequency_2',
'Frequency_3',
'TimeSearching',
'TimeAnswering',
'Frustration',
'ProfessionalTech',
'ProfessionalCloud',
'ProfessionalQuestion',
'Industry',
```

```
 'JobSatPoints_1',
 'JobSatPoints_4',
 'JobSatPoints_5',
 'JobSatPoints_6',
 'JobSatPoints_7',
 'JobSatPoints_8',
 'JobSatPoints_9',
 'JobSatPoints_10',
 'JobSatPoints_11',
 'SurveyLength',
 'SurveyEase',
 'ConvertedCompYearly',
 'JobSat']
```

Creates a new DataFrame df by selecting specific columns

[5]:
```
df = df_raw[['Age', 'Employment', 'WorkExp', 'RemoteWork', 'CodingActivities',␣
↪'EdLevel',
            'YearsCode', 'DevType', 'Country', 'ConvertedCompYearly',␣
↪'LanguageHaveWorkedWith', 'LanguageWantToWorkWith',
           'LanguageAdmired', 'DatabaseHaveWorkedWith',␣
↪'DatabaseWantToWorkWith', 'PlatformHaveWorkedWith', 'PlatformWantToWorkWith',
           'WebframeHaveWorkedWith', 'WebframeWantToWorkWith',␣
↪'OpSysProfessional use', 'Industry', 'JobSat']]
```

Display first rows of the new DataFrame

[6]:
```
df.head()  # Show first 5 rows of DataFrame
```

[6]:

	Age	Employment	WorkExp	RemoteWork	CodingActivities	EdLevel	YearsCode
0	Under 18 years old	Employed, full-time	NaN	Remote	Hobby	Primary/elementary school	NaN
1	35-44 years old	Employed, full-time	17.0	Remote	Hobby;Contribute to open-source projects;Other…	Bachelor’s degree (B.A., B.S., B.Eng., etc.)	20
2	45-54 years old	Employed, full-time	NaN	Remote	Hobby;Contribute to open-source projects;Other…	Master’s degree (M.A., M.S., M.Eng., MBA, etc.)	37
3	18-24 years old	Student, full-time	NaN	NaN	NaN	Some college/university study without earning …	4
4	18-24 years old	Student, full-time	NaN	NaN	NaN		

```
4  Secondary school (e.g. American high school, G…        9

               DevType                                               Country  \
0                  NaN                              United States of America
1  Developer, full-stack  United Kingdom of Great Britain and Northern I…
2   Developer Experience  United Kingdom of Great Britain and Northern I…
3  Developer, full-stack                                               Canada
4  Developer, full-stack                                               Norway

   ConvertedCompYearly  …  \
0                 NaN  …
1                 NaN  …
2                 NaN  …
3                 NaN  …
4                 NaN  …

                                  LanguageAdmired  \
0                                             NaN
1  Bash/Shell (all shells);Go;HTML/CSS;Java;JavaS…
2                                              C#
3  HTML/CSS;Java;JavaScript;PowerShell;Python;SQL…
4                C++;HTML/CSS;JavaScript;Lua;Python

         DatabaseHaveWorkedWith      DatabaseWantToWorkWith  \
0                           NaN                         NaN
1     Dynamodb;MongoDB;PostgreSQL                  PostgreSQL
2      Firebase Realtime Database  Firebase Realtime Database
3  MongoDB;MySQL;PostgreSQL;SQLite    MongoDB;MySQL;PostgreSQL
4               PostgreSQL;SQLite           PostgreSQL;SQLite

                  PlatformHaveWorkedWith  \
0                                    NaN
1  Amazon Web Services (AWS);Heroku;Netlify
2                           Google Cloud
3   Amazon Web Services (AWS);Fly.io;Heroku
4                                    NaN

                  PlatformWantToWorkWith  \
0                                    NaN
1  Amazon Web Services (AWS);Heroku;Netlify
2                           Google Cloud
3         Amazon Web Services (AWS);Vercel
4                                    NaN

                WebframeHaveWorkedWith            WebframeWantToWorkWith  \
0                                  NaN                               NaN
1         Express;Next.js;Node.js;React  Express;Htmx;Node.js;React;Remix
```

```
2                              ASP.NET CORE                        ASP.NET CORE
3  jQuery;Next.js;Node.js;React;WordPress      jQuery;Next.js;Node.js;React
4                                       NaN                                 NaN

  OpSysProfessional use Industry JobSat
0                    NaN      NaN    NaN
1                  MacOS      NaN    NaN
2                Windows      NaN    NaN
3                    NaN      NaN    NaN
4                    NaN      NaN    NaN

[5 rows x 22 columns]
```

Generates descriptive statistics for all columns

[7]:
```
df.describe(include='all')
```

[7]:
```
                    Age           Employment       WorkExp  \
count             65437                65437  29658.000000
unique                8                  110           NaN
top     25-34 years old  Employed, full-time           NaN
freq              23911                39041           NaN
mean                NaN                  NaN     11.466957
std                 NaN                  NaN      9.168709
min                 NaN                  NaN      0.000000
25%                 NaN                  NaN      4.000000
50%                 NaN                  NaN      9.000000
75%                 NaN                  NaN     16.000000
max                 NaN                  NaN     50.000000

                                  RemoteWork CodingActivities  \
count                                  54806            54466
unique                                     3              118
top     Hybrid (some remote, some in-person)           Hobby
freq                                   23015             9993
mean                                     NaN              NaN
std                                      NaN              NaN
min                                      NaN              NaN
25%                                      NaN              NaN
50%                                      NaN              NaN
75%                                      NaN              NaN
max                                      NaN              NaN

                                          EdLevel YearsCode  \
count                                       60784     59869
unique                                          8        52
top     Bachelor's degree (B.A., B.S., B.Eng., etc.)        10
freq                                        24942      4561
```

```
mean                                                NaN       NaN
std                                                 NaN       NaN
min                                                 NaN       NaN
25%                                                 NaN       NaN
50%                                                 NaN       NaN
75%                                                 NaN       NaN
max                                                 NaN       NaN

                      DevType                  Country  ConvertedCompYearly  \
count                   59445                    58930         2.343500e+04
unique                     34                      185                  NaN
top     Developer, full-stack  United States of America                  NaN
freq                    18260                    11095                  NaN
mean                      NaN                      NaN         8.615529e+04
std                       NaN                      NaN         1.867570e+05
min                       NaN                      NaN         1.000000e+00
25%                       NaN                      NaN         3.271200e+04
50%                       NaN                      NaN         6.500000e+04
75%                       NaN                      NaN         1.079715e+05
max                       NaN                      NaN         1.625660e+07

        … LanguageAdmired DatabaseHaveWorkedWith DatabaseWantToWorkWith  \
count   …           50872                 50254                  42558
unique  …           12335                  9050                   8478
top     …          Python            PostgreSQL             PostgreSQL
freq    …            1555                  3216                   3738
mean    …             NaN                   NaN                    NaN
std     …             NaN                   NaN                    NaN
min     …             NaN                   NaN                    NaN
25%     …             NaN                   NaN                    NaN
50%     …             NaN                   NaN                    NaN
75%     …             NaN                   NaN                    NaN
max     …             NaN                   NaN                    NaN

           PlatformHaveWorkedWith     PlatformWantToWorkWith  \
count                       42366                      34532
unique                       5467                       4784
top     Amazon Web Services (AWS)  Amazon Web Services (AWS)
freq                         6606                       4859
mean                          NaN                        NaN
std                           NaN                        NaN
min                           NaN                        NaN
25%                           NaN                        NaN
50%                           NaN                        NaN
75%                           NaN                        NaN
max                           NaN                        NaN
```

```
       WebframeHaveWorkedWith WebframeWantToWorkWith OpSysProfessional use  \
count                   45161                  38535                  52973
unique                  12235                  11654                   2032
top                     React                  React                Windows
freq                     1284                    997                  10472
mean                      NaN                    NaN                    NaN
std                       NaN                    NaN                    NaN
min                       NaN                    NaN                    NaN
25%                       NaN                    NaN                    NaN
50%                       NaN                    NaN                    NaN
75%                       NaN                    NaN                    NaN
max                       NaN                    NaN                    NaN

                     Industry        JobSat
count                   28858  29126.000000
unique                     15           NaN
top     Software Development            NaN
freq                    11918           NaN
mean                      NaN      6.935041
std                       NaN      2.088259
min                       NaN      0.000000
25%                       NaN      6.000000
50%                       NaN      7.000000
75%                       NaN      8.000000
max                       NaN     10.000000

[11 rows x 22 columns]
```

Returns the data type of each column in the DataFrame.

[8]:
```
df.dtypes
```

[8]:
```
Age                        object
Employment                 object
WorkExp                   float64
RemoteWork                 object
CodingActivities           object
EdLevel                    object
YearsCode                  object
DevType                    object
Country                    object
ConvertedCompYearly       float64
LanguageHaveWorkedWith     object
LanguageWantToWorkWith     object
LanguageAdmired            object
DatabaseHaveWorkedWith     object
DatabaseWantToWorkWith     object
PlatformHaveWorkedWith     object
```

```
PlatformWantToWorkWith     object
WebframeHaveWorkedWith     object
WebframeWantToWorkWith     object
OpSysProfessional use      object
Industry                   object
JobSat                    float64
dtype: object
```

Check for missing values in DataFrame

[9]:
```
df.isnull().sum()
```

[9]:
```
Age                           0
Employment                    0
WorkExp                   35779
RemoteWork                10631
CodingActivities          10971
EdLevel                    4653
YearsCode                  5568
DevType                    5992
Country                    6507
ConvertedCompYearly       42002
LanguageHaveWorkedWith     5692
LanguageWantToWorkWith     9685
LanguageAdmired           14565
DatabaseHaveWorkedWith    15183
DatabaseWantToWorkWith    22879
PlatformHaveWorkedWith    23071
PlatformWantToWorkWith    30905
WebframeHaveWorkedWith    20276
WebframeWantToWorkWith    26902
OpSysProfessional use     12464
Industry                  36579
JobSat                    36311
dtype: int64
```

Counts and prints the number of duplicate rows in the DataFrame, then removes them.

[10]:
```
print(df.duplicated().sum())  # Sum the duplicate rows in the DataFrame
df = df.drop_duplicates() #Remove the duplicate rows.
```

```
2190
```

1.6 Processing Database Experience by Age

- Select `Age` and `DatabaseHaveWorkedWith` columns.
- Split database strings into lists.
- Explode lists into individual rows.
- Group by age and database, counting entries.

[11]:
```
dfExp_Age = df[['Age', 'DatabaseHaveWorkedWith']] # Select relevant columns: Age and the databases people have worked with
dfExp_Age.loc[:,'DatabaseHaveWorkedWith'] = dfExp_Age['DatabaseHaveWorkedWith'].str.split(';') # Split the 'DatabaseHaveWorkedWith' into a list (separated by semicolons)
dfExp_Age = dfExp_Age.explode('DatabaseHaveWorkedWith') # Expand the list of databases into separate rows
Exp_Age_grouped = dfExp_Age.groupby(['Age', 'DatabaseHaveWorkedWith']).size().reset_index(name='count') # Group by Age and Database, then count the number of occurrences
```

1.6.1 Pivot Table: Age vs. Database Experience

We create a pivot table to analyze the relationship between respondents ages and the databases they have worked with.

The table uses: - **Rows** for `Age` - **Columns** for `DatabaseHaveWorkedWith` - **Values** as the count of respondents

Missing values (`NaN`) are replaced with `0`, as they represent age-database combinations with no responses. This ensures smooth visualization and avoids errors due to null values in the plot.

[12]:
```
Exp_Age_pivot_table = Exp_Age_grouped.pivot(index='Age', columns ='DatabaseHaveWorkedWith', values='count').fillna(0) #Create a pivot table of Age and DatabaseHaveWorkedWith
filt = Exp_Age_pivot_table.sum().sort_values(ascending=False).head().index #Gets the top 5 columns with the highest total values from the pivot table.
Exp_Age_pivot_table = Exp_Age_pivot_table[filt]#Filters the pivot table to keep only the top 5 columns.
Exp_Age_pivot_table #Show the results
```

[12]:
```
DatabaseHaveWorkedWith  PostgreSQL   MySQL  SQLite  Microsoft SQL Server  \
Age
18-24 years old              5262.0  5204.0  4061.0                2137.0
25-34 years old             10120.0  7707.0  6140.0                4470.0
35-44 years old              6279.0  4507.0  3952.0                3559.0
45-54 years old              2458.0  2049.0  1684.0                1987.0
55-64 years old               780.0   757.0   630.0                 807.0
65 years or older             127.0   219.0   177.0                 160.0
Prefer not to say              61.0    73.0    73.0                  43.0
Under 18 years old            448.0   583.0   648.0                 112.0

DatabaseHaveWorkedWith  MongoDB
Age
18-24 years old          3615.0
25-34 years old          5123.0
35-44 years old          2546.0
45-54 years old           939.0
```

```
55-64 years old          266.0
65 years or older         46.0
Prefer not to say         35.0
Under 18 years old       437.0
```

1.6.2 Bar Chart: Top 5 Databases Used by Age

We now plot a bar chart showing the distribution of the top 5 most commonly used databases across different age groups. The chart helps visualize usage trends by age range.

[13]:
```
Exp_Age_pivot_table.plot(kind='bar', stacked=False, figsize=(10, 6),␣
↪colormap='tab20c', edgecolor = 'black')#Plots a bar chart with custom style.

plt.title('Databases used by age ranges')  # Set plot title
plt.ylabel('Proportion')  # Set y-axis label
plt.xlabel('Age') # Set x-axis label
plt.legend(title='Database') #Adds a legend with the title 'Database'.
plt.tight_layout()  # Adjust layout spacing
plt.xticks(rotation=70)  # Rotate x-axis labels
plt.figtext(0.5, -0.05,'Figure 1. Databases used by age ranges', # Adds␣
↪centered caption below the plot
            wrap=True, horizontalalignment='center', fontsize=10)
plt.show()  # Display the plot
```

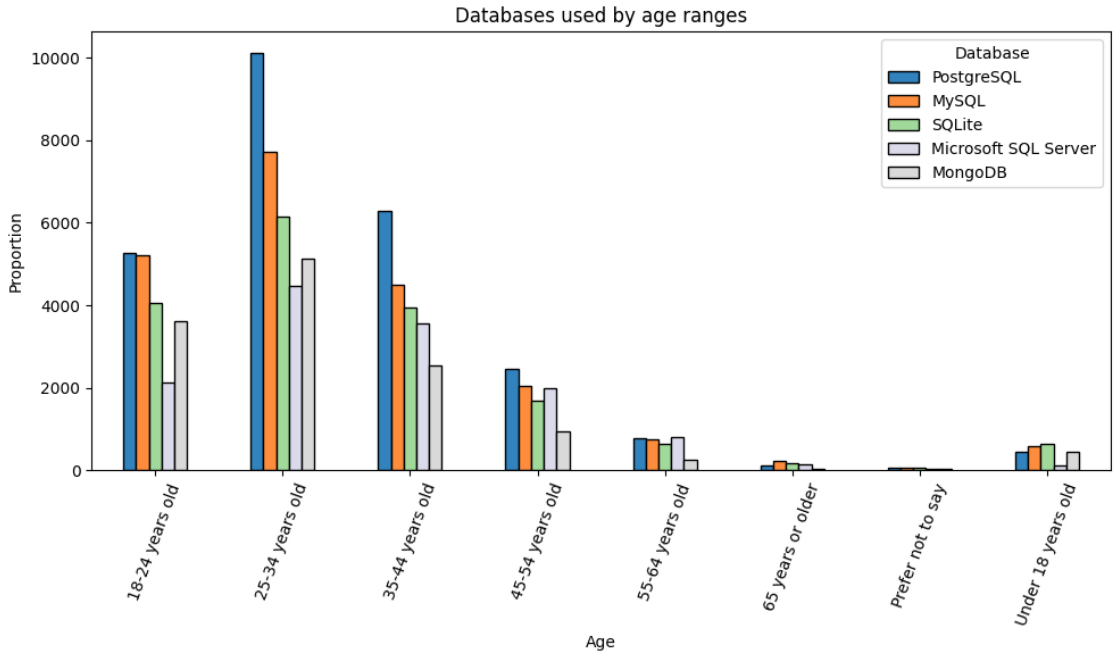

Figure 1. Databases used by age ranges

Conclusions: Database Usage by Age Group

- **PostgreSQL** is the most widely used database among all age groups, especially prominent in the **25–34** age range.
- **MySQL** follows PostgreSQL closely, with its usage peaking in the **25–34** age group and gradually declining in older age brackets.
- **SQLite** and **Microsoft SQL Server** show moderate usage across all age ranges, with a noticeable drop-off after the **35–44** age group.
- **MongoDB** usage is relatively lower overall, but it's more evenly distributed across younger and middle age groups.
- There is a clear decline in the number of respondents as age increases, especially after the **45–54** range, suggesting lower representation or participation from older developers.
- Younger respondents (**18–24** and **under 18**) already show significant usage of PostgreSQL and MySQL, indicating these technologies are common even among early-career or student developers.

1.7 Choropleth Map: Number of Developers by Country

We calculate the number of survey respondents (developers) from each country and visualize the results using a choropleth map.

- The `Country` column is used to identify geographic locations.
- The `Developer_Count` column represents the number of developers per country.
- A `plotly.express` choropleth is used for an interactive world map visualization, with a continuous Viridis color scale to represent density.

This map helps identify where most developers in the dataset are located geographically.

[32]:
```
country_counts = df['Country'].value_counts().reset_index()  # Get counts of␣
↪unique values
country_counts.columns = ['Country', 'Developer_Count']#Renames columns to␣
↪'Country' and 'Developer_Count'.

fig = px.choropleth( #Creates a choropleth map of developers by country.
    country_counts,
    locations='Country',
    locationmode='country names',
    color='Developer_Count',
    color_continuous_scale='Viridis',
    title='Number of developers by country'
)
fig.add_annotation( # Adds centered caption below the plot
    text="Figure 2. Number of developers by country based on survey data.",
    xref="paper", yref="paper",
    x=0.5, y=-0.15,
    showarrow=False,
    font=dict(size=12),
    align="center"
)
```

```
fig.show()
```

1.7.1 Conclusions: Number of Developers by Country

- The **United States** has the highest number of developers, clearly leading with over **10,000** respondents.
- Other countries with a significant developer presence include **India**, **Germany**, **United Kingdom**, and **Canada**.
- European countries show moderate developer activity, especially in **Western and Central Europe**.
- **South America**, **Africa**, and most parts of **Asia** have lower representation, with smaller clusters in countries like **Brazil**, **Nigeria**, and **Indonesia**.
- The heatmap highlights a digital divide, with **North America**, parts of **Europe**, and **India** dominating in developer count.
- These figures may reflect both population size and internet access, as well as participation in the survey source (likely Stack Overflow or similar).

1.8 Analyzes annual compensation in the top 10 countries with the most developers

We focus on the top 10 countries with the highest number of developers.

- Filter the dataset to include only these countries.
- Calculate the interquartile range (IQR) of the yearly salary (`ConvertedCompYearly`) to detect outliers.
- Remove outliers by keeping salaries within 1.5 * IQR above and below the quartiles.
- Replace long country names with shorter versions for better visualization and readability (e.g., "United Kingdom of Great Britain and Northern Ireland" to "UK").

This cleaning step ensures that extreme salary values do not skew the analysis.

[15]:
```
topCountry = df['Country'].value_counts().head(10).index  # Show first 10 rows␣
↪of DataFrame
topCountry
filtroCtryCCY = df[df['Country'].isin(topCountry)]  # Access DataFrame column


def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)  # Calculate specified quantile
    Q3 = df[column].quantile(0.75)  # Calculate specified quantile
    IQR = Q3 - Q1           #Calculates the interquartile range (IQR).
    lower = Q1 - 1.5 * IQR  #Calculates the lower bound for outliers.
    upper = Q3 + 1.5 * IQR  #Calculates the upper bound for outliers.

    df = df[(df[column] >= lower)
    & (df[column] <= upper)]#Filters data within the salary bounds to remove␣
↪outliers.

    return df
```

```
filtroCtryCCY = remove_outliers(filtroCtryCCY, 'ConvertedCompYearly')  #Filters␣
 ↪data within the salary bounds to remove outliers.


namereplace = {"United Kingdom of Great Britain and Northern Ireland":"UK",␣
 ↪"United States of America":"USA"}
filtroCtryCCY.loc[:, 'Country'] = filtroCtryCCY['Country'].replace(namereplace)␣
 ↪ # Replace DataFrame names
```

1.8.1 Boxplot of Annual Compensation in the Top 10 Countries

This boxplot visualizes the distribution of yearly compensation (`ConvertedCompYearly`) for developers in the top 10 countries with the most respondents.

- The x-axis shows the countries.
- The y-axis shows the annual salary, filtered to remove outliers.
- The boxplot highlights the median, quartiles, and potential remaining spread of salaries by country.
- X-axis labels are rotated for better readability.

This visualization helps compare salary distributions across countries while minimizing the effect of extreme values.

[16]:
```
plt.figure(figsize=(12,7))  # Initialize new matplotlib figure

plt.title('Annual compensation in the top 10 countries with the most␣
 ↪developers')  # Set plot title
sns.boxplot(x='Country', y='ConvertedCompYearly', data=filtroCtryCCY)  # Draw␣
 ↪boxplot
sns.set(style='darkgrid')
plt.ylabel('Converted Compensation (Yearly)')  # Set y-axis label
plt.xticks(rotation=90)  # Rotate x-axis labels
plt.tight_layout()  # Adjust layout spacing
plt.figtext(0.5, -0.05,'Figure 3. Annual compensation in the top 10 countries␣
 ↪with the most developers', # Adds centered caption below the plot
            wrap=True, horizontalalignment='center', fontsize=10)
plt.show()  # Display the plot
```

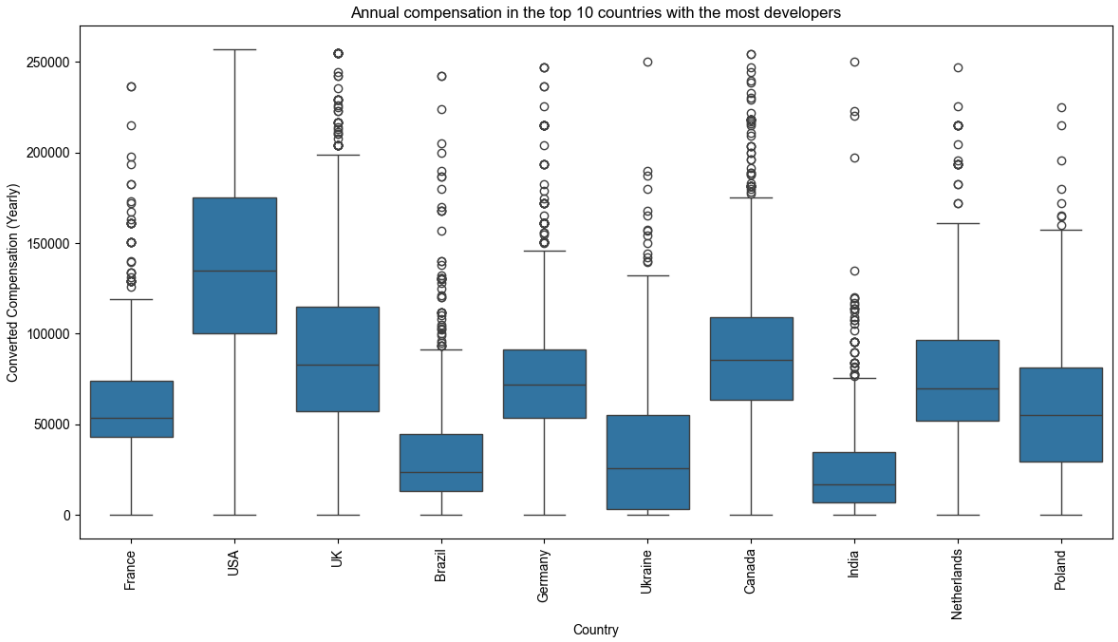

Figure 3. Annual compensation in the top 10 countries with the most developers

1.8.2 Conclusions: Annual Compensation in the Top 10 Countries with the Most Developers

- The **United States** shows the **highest median compensation**, with a wide spread and many high-end outliers exceeding $250,000 USD.
- **Canada** and the **UK** also show relatively high median salaries compared to other countries.
- **India** and **Brazil** have the **lowest median compensations**, though they still have many high-end outliers, suggesting income inequality or strong variation in experience/roles.
- **France**, **Germany**, and the **Netherlands** have more concentrated salary distributions, with fewer extreme outliers compared to the US.
- **Ukraine** and **Poland** show modest median compensation, but a wider spread of data suggests significant variation within those countries.
- Overall, compensation tends to correlate with the economic strength and cost of living of the country, but outliers highlight opportunities for high-earning individuals across all regions.

1.9 Bar Plot: Annual Compensation by Work Modality

We analyze how yearly compensation (`ConvertedCompYearly`) varies depending on whether developers work remotely or not.

- The dataset is filtered to remove outliers in compensation.
- A bar plot shows the average salary for each category of work modality (`RemoteWork`).
- Bars are annotated with their exact average salary values for clarity.
- The y-axis limit is set slightly above the highest average to improve visualization.

This plot helps understand if and how remote work status affects developer compensation.

[17]:
```
dfRemConv = remove_outliers(df, 'ConvertedCompYearly')  #Filters data within␣
 ↪the salary bounds to remove outliers.

plt.figure(figsize=(10,5))  # Initialize new matplotlib figure
plt.title('Annual compensation by work modality')  # Set plot title
ax = sns.barplot(x='RemoteWork', y='ConvertedCompYearly', data=dfRemConv,␣
 ↪color='brown', edgecolor='black')#Creates a brown barplot of salary by␣
 ↪remote work status.
plt.xlabel('Work Modality') #Set x-axis label
plt.ylabel('Converted Compensation (Yearly)')  # Set y-axis label
max_y = dfRemConv.groupby('RemoteWork')['ConvertedCompYearly'].mean().max()␣
 ↪#Finds the highest average salary by remote work group.
ax.set_ylim(0, max_y * 1.2) #Sets y-axis limit slightly above the max average␣
 ↪salary.

for bar in ax.patches:
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 4500, f'{bar.
 ↪get_height():.0f}', ha='center', va='bottom') #Adds value labels above each␣
 ↪bar in the plot.

plt.figtext(0.5, -0.05,'Figure 4. Annual compensation by work modality', # Adds␣
 ↪centered caption below the plot
            wrap=True, horizontalalignment='center', fontsize=10)
plt.tight_layout()  # Adjust layout spacing
plt.show()  # Display the plot
```

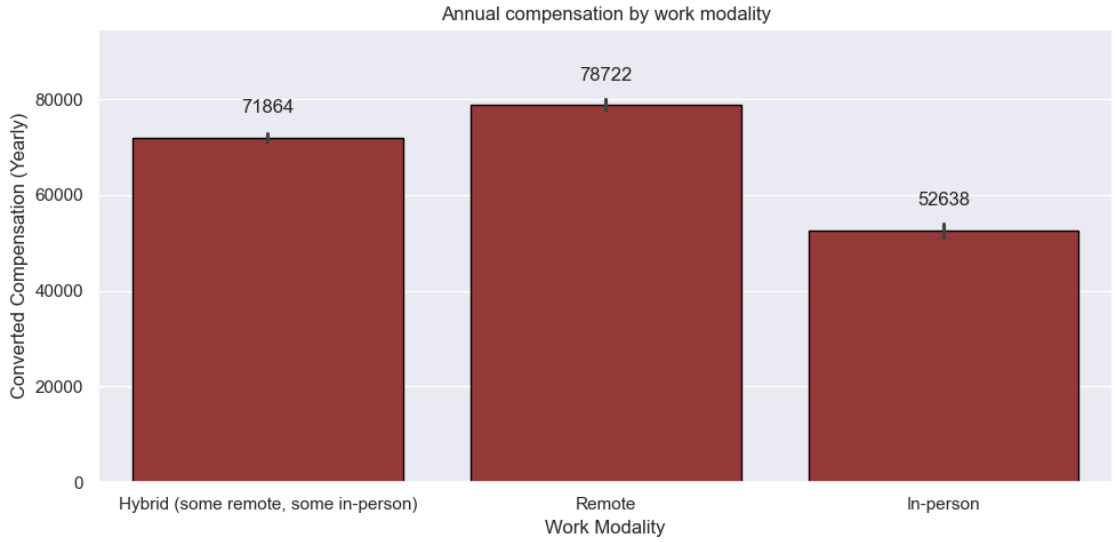

Figure 4. Annual compensation by work modality

1.9.1 Conclusions: Annual Compensation by Work Modality

- **Remote workers** receive the highest average annual compensation at **$78,722**, highlighting the financial advantage of fully remote positions.
- **Hybrid workers** (those with a mix of remote and in-person work) earn slightly less, with an average salary of **$71,864**, indicating some flexibility may still yield competitive pay.
- **In-person workers** report the lowest average annual compensation at **$52,638**, showing a significant gap compared to remote roles — roughly **33% less** than remote counterparts.
- The data suggests a clear trend: **greater flexibility in work location is associated with higher compensation**.
- These results may reflect broader market demand, talent distribution, or cost-of-living adjustments that favor remote work arrangements.

1.10 Analysis and Visualization of Programming Languages Used Across Countries

- We select respondents programming languages (`LanguageHaveWorkedWith`) and their countries.
- Missing values are dropped to ensure clean data processing and accurate visualization:
  - Null values in either column would interfere with text splitting, row expansion, grouping, and plotting.
  - Dropping them prevents errors and avoids introducing meaningless entries (e.g., blank countries or languages).
- Languages are split into individual entries for each developer.
- We focus on the top 10 most used programming languages and the top 20 countries with the most responses.
- Data is grouped by country and language to count occurrences.
- We calculate and print:
  - The most popular language and its percentage of total responses.
  - The combined percentage of the top 10 languages.
- A bubble plot visualizes the frequency of languages used per country:
  - The size and color of each bubble represent how many developers in that country use the language.
  - X-axis shows programming languages, y-axis shows countries.

This plot provides an intuitive overview of programming language popularity and distribution across different countries.

[18]:
```
dfProgram = df[['LanguageHaveWorkedWith', 'Country']].dropna()  #Selects␣
↪language and country columns, drops missing values.
dfProgram['LanguageHaveWorkedWith'] = dfProgram['LanguageHaveWorkedWith'].str.
↪split(';') #Splits languages into lists by semicolon.
dfProgram = dfProgram.explode('LanguageHaveWorkedWith') #Expands list of␣
↪languages into separate rows.

filtro = dfProgram['LanguageHaveWorkedWith'].value_counts().
↪sort_values(ascending=False).head(10).index #Gets top 10 most used languages.
```

```
dfProgram_filter = dfProgram[dfProgram['LanguageHaveWorkedWith'].isin(filtro)]␣
 ↪#Filters to keep only the top 10 languages
filtro2 = dfProgram_filter['Country'].value_counts().
 ↪sort_values(ascending=False).head(20).index  #Gets top 20 countries by count.
dfProgram_filter = dfProgram_filter[dfProgram_filter['Country'].isin(filtro2)]␣
 ↪#Filters to keep only the top 20 countries.

grupo = dfProgram_filter.groupby(['Country', 'LanguageHaveWorkedWith']).size().
 ↪reset_index(name='count') #Groups by country and language, counts occurrences
grupo = grupo.replace(namereplace)  # Replace DataFrame names


LangPorcentage = dfProgram.groupby(['Country', 'LanguageHaveWorkedWith']).
 ↪size().reset_index(name='count')
LangPorcentage = LangPorcentage[['LanguageHaveWorkedWith', 'count']].
 ↪groupby('LanguageHaveWorkedWith').sum() \
.sort_values(['count'], ascending=False).reset_index()
TotalP = LangPorcentage['count'].sum()
Top1 = LangPorcentage.iloc[0,1]/TotalP * 100
Top10_total = (LangPorcentage['count'].iloc[:10].sum() / TotalP) * 100
print(f'\033[1mThe top 1 language is {LangPorcentage.iloc[0, 0]} with {Top1:.
 ↪1f}% \
and the sum of the top 10 most used languages is {Top10_total:.1f}%\033[0m')
print('')

plt.figure(figsize=(15,10))  # Initialize new matplotlib figure
plt.scatter(
    grupo['LanguageHaveWorkedWith'],
    grupo['Country'],
    s=grupo['count'],
    c=grupo['count'],
    alpha = 0.5
)
plt.title('Bubble Plot for Languages Worked With Across Countries')  # Set plot␣
 ↪title
plt.xlabel('Languages') # Set x-axis label
plt.ylabel('Country')  # Set y-axis label
plt.xticks(rotation = 45)  # Rotate x-axis labels
plt.tight_layout()  # Adjust layout spacing
plt.colorbar(label='Frequence') #Adds a colorbar labeled 'Frequence'.
plt.figtext(0.5, -0.05,'Figure 5. Bubble Plot for Languages Worked With Across␣
 ↪Countries', # Adds centered caption below the plot
            wrap=True, horizontalalignment='center', fontsize=10)
plt.show()  # Display the plot
```

The top 1 language is JavaScript with 11.6% and the sum of the top 10 most used languages is 72.9%

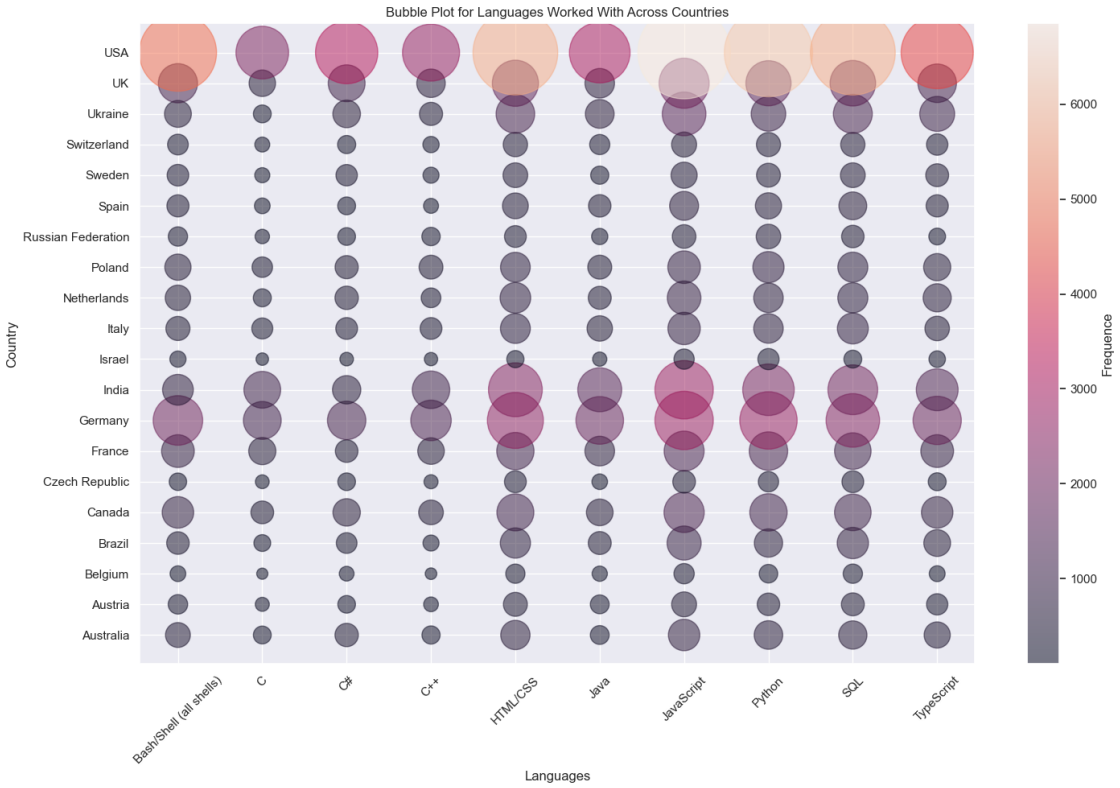

Figure 5. Bubble Plot for Languages Worked With Across Countries

1.10.1 Conclusions: Languages Worked With Across Countries

- **JavaScript** and **HTML/CSS** are the most widely used programming languages across nearly all countries, particularly dominant in the **USA**, **UK**, and **India**.
- **Python** shows consistently high usage worldwide, with notable representation in countries like **USA**, **India**, and **Germany**, indicating its global versatility across roles and industries.
- **SQL** also ranks among the most frequently used languages, highlighting the importance of data handling skills across regions.
- **C**, **C++**, and **C#** have more moderate usage, with relatively higher representation in countries such as **Germany**, **France**, and **India**, suggesting stronger ties to traditional software development or embedded systems.
- **TypeScript** shows strong adoption in **USA**, **UK**, and **India**, reflecting growing trends in modern frontend and full-stack development.
- Smaller European countries (e.g., **Sweden**, **Switzerland**, **Austria**) show generally lower absolute usage across all languages, likely due to smaller sample sizes rather than lack of developer engagement.
- The **USA** consistently reports the highest frequencies across nearly all languages, underscoring its large and diverse developer population.

1.11 Analysis of Desired Programming Languages Across Countries

- We extract respondents desired programming languages to work with (`LanguageWantToWorkWith`) along with their countries.
- Missing values are dropped to ensure reliable data transformation and visualization:
  - Null values would break text processing (e.g., splitting) and cause issues in the plot (e.g., blank axes labels).
- Languages are split into individual entries.
- The focus is on the top 10 most desired languages and the top 20 countries with the highest number of responses.
- Data is grouped by country and desired language to count how many developers want to work with each language.

This data preparation sets the stage for visualizing developer preferences for programming languages by country.

[19]:
```
dfLW = df[['LanguageWantToWorkWith', 'Country']].dropna() #Selects desired␣
 ↪language and country columns, drops missing values.
dfLW['LanguageWantToWorkWith'] = dfLW['LanguageWantToWorkWith'].str.split(';')␣
 ↪#Splits desired languages into lists by semicolon.
dfLW = dfLW.explode('LanguageWantToWorkWith') #Expands desired languages into␣
 ↪separate rows.

filtroLanguages = dfLW['LanguageWantToWorkWith'].value_counts().
 ↪sort_values(ascending=False).head(10).index #Gets top 10 desired languages.

dfLW = dfLW[dfLW['LanguageWantToWorkWith'].isin(filtroLanguages)] #Filters to␣
 ↪keep only the top 10 desired languages.
dfLW = dfLW[dfLW['Country'].isin(filtro2)] #Filters to keep only the top 20␣
 ↪countries.

group = dfLW.groupby(['Country', 'LanguageWantToWorkWith']).size().
 ↪reset_index(name='count').replace(namereplace) #Groups by country and␣
 ↪desired language, counts, and replaces names.
```

1.11.1 Bubble Plot: Desired Programming Languages Across Countries

This bubble plot visualizes the distribution of the top 10 programming languages developers want to work with, across the top 20 countries.

- The size and color of each bubble represent the number of developers in a country interested in a particular language.
- The x-axis shows the programming languages.
- The y-axis shows the countries.
- X-axis labels are rotated for better readability.
- A colorbar indicates the frequency scale.

This plot helps reveal trends and preferences in programming languages developers aspire to work with worldwide.

[20]:
```
plt.figure(figsize=(15,10))  # Initialize new matplotlib figure
plt.scatter(
    group['LanguageWantToWorkWith'],
    group['Country'],
    s=group['count'],
    c=group['count'],
    alpha = 0.5
)
plt.title('Bubble Plot for Databases Wanted Across Countries', fontsize=16)  #␣
 ↪Set plot title
plt.xlabel('Languages') #Set x-axis label
plt.ylabel('Country')  # Set y-axis label
plt.xticks(rotation = 45)  # Rotate x-axis labels
plt.tight_layout()  # Adjust layout spacing
plt.colorbar(label='Frequence')
plt.figtext(0.5, -0.05,'Figure 6. Bubble Plot for Databases Wanted Across␣
 ↪Countries', # Adds centered caption below the plot
            wrap=True, horizontalalignment='center', fontsize=13)
plt.show()  # Display the plot
```

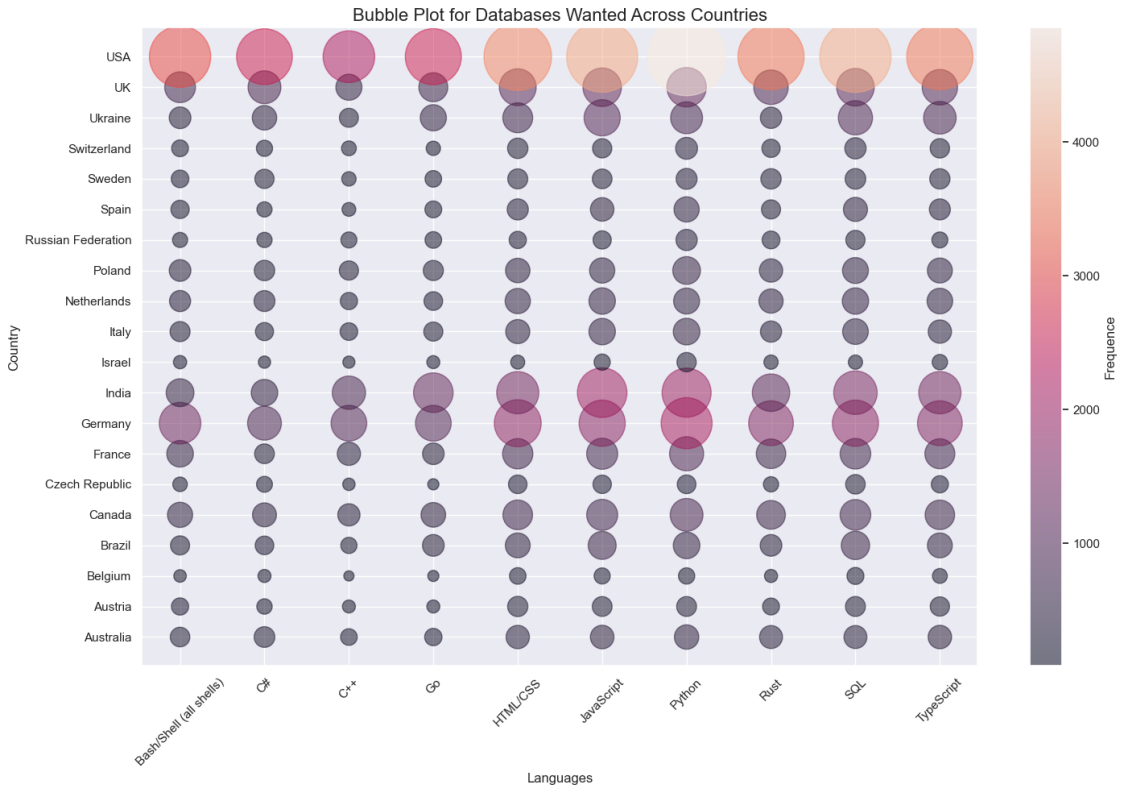

Figure 6. Bubble Plot for Databases Wanted Across Countries

1.11.2 Conclusions: Databases Wanted Across Countries

- **JavaScript**, **Python**, and **SQL** are the most *wanted* technologies across nearly all countries, with the **USA**, **India**, and **UK** showing the highest interest.
- **TypeScript** also appears in high demand, especially in English-speaking countries, suggesting strong momentum for modern frontend and full-stack development.
- Demand for **HTML/CSS** remains consistently strong, likely due to its foundational role in web development.
- Interest in lower-level languages like **C**, **C++**, and **C#** is notably lower across most regions, with relatively stronger demand in **Germany**, **France**, and **India**.
- The **USA** continues to show the highest overall demand for nearly every language, reinforcing its large, diverse, and fast-evolving tech market.
- Emerging markets like **India** and **Ukraine** demonstrate comparable demand patterns to developed countries, particularly in high-growth technologies such as **Python** and **JavaScript**.
- Compared to actual usage (from the previous bubble chart), the *wanted* technologies reveal similar trends, suggesting that developers want to work with the same languages they already use — especially in popular ecosystems like JavaScript/TypeScript and Python.

1.12 Relationship Between Work Experience and Annual Salary

- We select work experience (`WorkExp`) and yearly compensation (`ConvertedCompYearly`), dropping missing values:
  - Null values are removed to ensure valid numerical operations such as averaging and percentage calculations.
  - Keeping them would interfere with statistical calculations and plot accuracy.
- Outliers in salary are removed using previously calculated salary bounds (`lower` and `upper`).
- The average annual salary is calculated for each year of experience.
- The data is sorted by years of experience.
- We compute and print the total percentage increase in salary from year 0 to year 5.
- A scatter plot visualizes the relationship between years of experience and average annual salary.

This analysis helps illustrate how salary tends to increase with work experience.

[21]:
```
dfExpSalarie = df[['WorkExp', 'ConvertedCompYearly']]
dfExpSalarie = dfExpSalarie.dropna()
dfExpSalarie_outliers = remove_outliers(dfExpSalarie, 'ConvertedCompYearly') ␣
 ↪#Filters data within the salary bounds to remove outliers.
groupExpSalarie = dfExpSalarie_outliers.
 ↪groupby('WorkExp')['ConvertedCompYearly'].mean().reset_index()

groupExpSalarie = groupExpSalarie.sort_values('WorkExp')

initial_salary = groupExpSalarie.loc[groupExpSalarie['WorkExp'] == 0,␣
 ↪'ConvertedCompYearly'].values[0]
salary_year_5 = groupExpSalarie.loc[groupExpSalarie['WorkExp'] == 5,␣
 ↪'ConvertedCompYearly'].values[0]
```

```
total_pct_increase = ((salary_year_5 - initial_salary) / initial_salary) * 100

print(f"Total salary increase from year 1 to year 5: {total_pct_increase:.2f}%")

sns.scatterplot(data=groupExpSalarie, x='WorkExp', y='ConvertedCompYearly')
plt.title('Work Experience vs Annual Salary')
plt.xlabel('Years of Experience')
plt.ylabel('Annual Salary (USD)')
plt.figtext(0.5, -0.05,'Figure 7. Work Experience vs Annual Salary', # Adds␣
 ↪centered caption below the plot
            wrap=True, horizontalalignment='center', fontsize=9)
plt.show()
```

```
Total salary increase from year 1 to year 5: 152.03%
```

Figure 7. Work Experience vs Annual Salary

1.12.1 Conclusions: Work Experience vs Annual Salary

- **Annual salary increases sharply during the first 5–10 years** of work experience, with compensation nearly doubling compared to entry-level salaries.

- After around **10–15 years of experience**, salary growth begins to **slow down**, indicating a plateau effect in mid to late career stages.
- **Maximum average salaries** appear between **30 and 40 years of experience**, reaching over **$120,000 USD**.
- Beyond **40 years**, salaries tend to **stabilize or slightly decline**, possibly due to role changes (e.g., moving into advisory, part-time, or lower-pressure positions).
- The chart clearly supports the idea that **early career growth is the most financially impactful period**, while gains in later years are less pronounced.
- This insight can be useful for career planning, salary negotiations, and understanding long-term earning trajectories.

1.13 Analysis of Desired Web Frameworks

- We extract the `WebframeWantToWorkWith` column, which lists the web frameworks developers want to work with.
- Missing values are dropped to ensure proper data processing and accurate results:
  - Null values would cause errors during string splitting and row expansion, and would distort framework counts.
- Frameworks are split into individual entries.
- The data is exploded so each framework appears in its own row.
- We count the occurrences of each framework and identify the top 10 most desired ones.
- The counts of all other frameworks beyond the top 10 are summed into an "Others" category.
- Labels and sizes for a potential pie or bar chart are prepared, including the "Others" category.

This preparation helps visualize the popularity of web frameworks among developers.

[22]:
```
dfWeb_Frame = df['WebframeWantToWorkWith'].dropna().str.split(';') #Splits␣
 ↪desired web frameworks into lists, drops missing values.
dfWeb_Frame = dfWeb_Frame.explode() #Expands desired web frameworks into␣
 ↪separate rows.

top = dfWeb_Frame.value_counts() #Counts occurrences of each web framework.
top5 = top.head(10) #Selects the top 10 most wanted web frameworks.
others = top[10:].sum() #Sums counts of web frameworks outside the top 10.

labels = top5.index.tolist() + ['Others'] #Creates labels list with top 10 and␣
 ↪'Others'.
sizes = top5.values.tolist() + [others] #Creates sizes list with top 10 counts␣
 ↪plus 'Others' sum.
```

1.13.1 Pie Chart: Preferred Web Frameworks (Top 10 + Others)

- This pie chart visualizes the distribution of the top 10 most desired web frameworks among developers.
- The "Others" category aggregates all less popular frameworks.
- Percentages are displayed inside each slice for clarity.
- The chart uses the 'Solarize_Light2' style with black edges on each wedge for better contrast.
- A legend lists all categories in the upper right corner.
- The aspect ratio is set to equal to ensure the pie is circular.

This chart provides a clear overview of web framework preferences in the developer community.

[23]:
```
plt.figure(figsize=(10, 6))  # Initialize new matplotlib figure
patches, texts, autotexts = plt.pie(
    sizes,
    labels=labels,
    autopct='%1.1f%%',
    startangle=140,
    wedgeprops={'edgecolor': 'black'}
)

for text in texts:
    text.set_fontsize(11)
for autotext in autotexts:
    autotext.set_fontsize(11)
    autotext.set_color('white')

plt.title('Pie Chart of Preferred Web Frameworks (Top 5 + Others)',␣
 ↪fontsize=14)  # Set plot title
plt.axis('equal')  #Sets plot aspect ratio to be equal
plt.tight_layout()  # Adjust layout spacing
plt.legend(labels, loc='upper right') #Adds a legend with labels in the upper␣
 ↪right corner.
plt.style.use('Solarize_Light2') #Applies the 'Solarize_Light2' plot style.
plt.figtext(0.5, -0.05,'Figure 8. Pie Chart of Preferred Web Frameworks', #␣
 ↪Adds centered caption below the plot
            wrap=True, horizontalalignment='center', fontsize=10)
plt.show()  # Display the plot
```

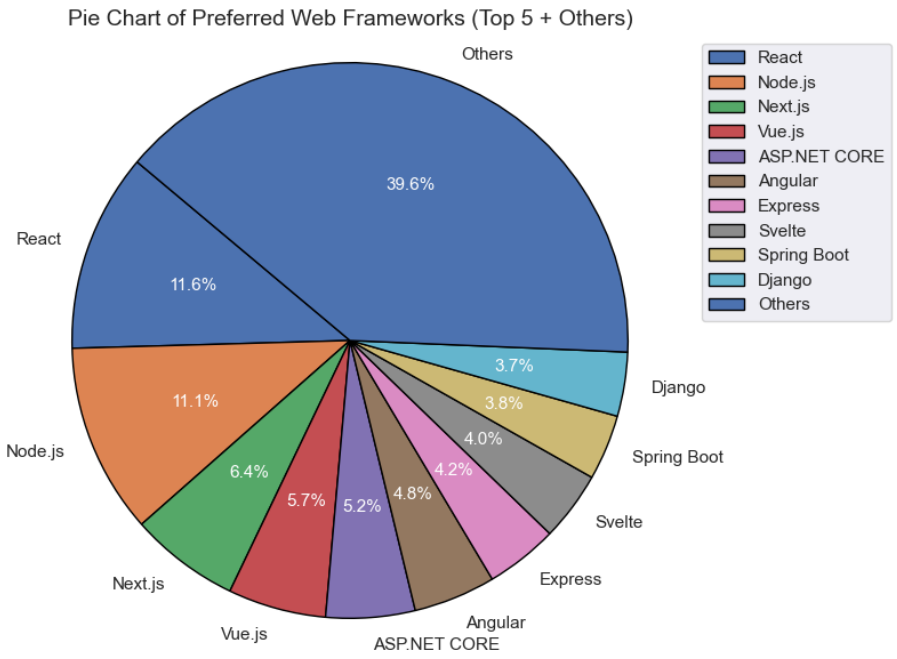

Figure 8. Pie Chart of Preferred Web Frameworks

1.13.2 Conclusions: Preferred Web Frameworks

- **React** is the most preferred web framework, accounting for **11.6%** of respondents' preferences among the top listed frameworks.
- **Node.js** follows closely with **11.1%**, showing strong popularity for backend JavaScript development.
- **Next.js** and **Vue.js** maintain solid positions with **6.4%** and **5.7%** respectively, indicating rising interest in modern, flexible frontend frameworks.
- **ASP.NET Core** (5.2%) and **Angular** (4.8%) continue to hold relevance, especially among enterprise developers.
- Newer or niche frameworks such as **Svelte** (4.0%), **Spring Boot** (3.8%), and **Django** (3.7%) also show steady adoption, suggesting diversified ecosystem preferences.
- The **"Others"** category, comprising **39.6%**, indicates a broad variety of frameworks in use beyond the top 10, reflecting fragmentation and specialization in web development technologies.
- Overall, JavaScript-based frameworks dominate the top spots, emphasizing its central role in modern web development across both frontend and backend.

1.14 Line Plot: Average Annual Compensation by Age Group

- We filter the dataset to include only respondents with yearly compensation (`ConvertedCompYearly`) within the previously calculated salary bounds to remove outliers.
- After filtering, we retrieve the unique ages present in this cleaned dataset.

This step ensures subsequent analyses consider only valid salary data and identifies the range of

respondent ages available for further study.

[24]:
```
dfAC = remove_outliers(df, 'ConvertedCompYearly')  #Filters data within the␣
↪salary bounds to remove outliers.

dfAC['Age'].unique() #Gets unique ages in the filtered data.
```

[24]:
```
array(['18-24 years old', '25-34 years old', '35-44 years old',
       '45-54 years old', '55-64 years old', '65 years or older',
       'Under 18 years old', 'Prefer not to say'], dtype=object)
```

1.14.1 Cleaning and Transforming Age Data for Salary Analysis

- Remove respondents who chose "Prefer not to say" or are under 18 years old.
- Clean the `Age` column by removing text such as "years old" or "years or older".
- Split age ranges (e.g., "25-29") into lists and calculate the average age for each range.
- Drop rows with missing values in `Age` or `ConvertedCompYearly` to ensure accurate calculations:
  - Null values in these columns would lead to incorrect mean salary calculations and potentially cause missing or misleading points in the visualization.
- Group the cleaned data by average age and calculate the mean yearly compensation for each age group.

This transformation allows a more precise analysis of salary trends across continuous age values rather than categorical ranges.

[25]:
```
dfAC = dfAC[~dfAC['Age'].isin(['Prefer not to say', 'Under 18 years old'])].
↪copy() #Removes entries with 'Prefer not to say' or 'Under 18 years old'␣
↪ages.
dfAC['Age'] = dfAC['Age'].str.replace(r' years old| years or older', '',␣
↪regex=True) #Removes text like 'years old' from the Age values.
dfAC['Age'] = dfAC['Age'].str.split('-') #Splits Age values into lists by␣
↪hyphen.
dfAC['Age'] = dfAC['Age'].apply(lambda x: sum([float(i) for i in x]) / len(x)␣
↪if isinstance(x, list) else np.nan) #Calculates average age from age ranges.

dfAC = dfAC.dropna(subset=['Age', 'ConvertedCompYearly'])
group2 = dfAC.groupby('Age')['ConvertedCompYearly'].mean().reset_index()␣
↪#Calculates average salary by age group.
```

1.14.2 Line Plot: Relationship Between Age and Annual Compensation

- This line plot shows the average yearly compensation (`ConvertedCompYearly`) across different ages.
- Markers highlight individual data points for each age.
- The plot includes descriptive axis labels and a clear title.
- Layout settings enhance readability with adjusted size and font sizes.

This visualization helps observe trends and patterns in how salary varies with age.

[26]:
```
fig = px.line(group2,        #Creates a line plot of average salary by age with␣
 ↪markers and labels.
              x='Age',
              y='ConvertedCompYearly',
              markers=True,
              title='Relationship Between Age and Converted Yearly␣
 ↪Compensation',
              labels={
                  'Age': 'Age',
                  'ConvertedCompYearly': 'Converted Compensation (Yearly)'
              })

fig.update_layout(       #Updates plot layout with size and font settings.
    width = 900,
    height = 400,
    title_font_size=18,
    xaxis_title_font_size=14,
    yaxis_title_font_size=14,
    margin=dict(t=80, b=80)

)

fig.add_annotation(
    text="Figure 9. Relationship Between Age and Converted Yearly␣
 ↪Compensation", # Adds centered caption below the plot
    xref="paper", yref="paper",
    x=0.5, y=-.43,
    font=dict(size=12),
    align="center"
)
fig.show()
```

1.14.3 Conclusions: Age vs Annual Compensation

- **Annual compensation increases steadily with age**, particularly from **20 to 50 years old**, suggesting consistent salary growth throughout early and mid-career stages.
- The most **significant salary jump** appears between the **20–30 age range**, indicating rapid early-career progression.
- Salaries appear to **peak around age 60**, reaching just above **$110,000 USD** on average.
- After **age 60**, there is a **slight decline** in compensation, possibly due to transitions into part-time roles, retirement plans, or advisory positions.
- Overall, age shows a **positive correlation with earnings** up to a point, reinforcing the value of experience but also hinting at a plateau in later years.
- These trends align closely with the patterns observed in the **Work Experience vs Salary** plot, emphasizing long-term career growth followed by stabilization.

1.15 Average Salary by Education Level

- Filter the dataset to include only salaries within the predefined bounds to remove outliers.
- Drop rows with missing values in `EdLevel` or `ConvertedCompYearly` to ensure accurate average salary calculations and avoid errors in grouping and plotting.
- Calculate the average yearly compensation (`ConvertedCompYearly`) for each education level (`EdLevel`).
- Replace long or complex education level names with simplified labels for better readability in plots.

This preparation facilitates a clear comparison of how education level correlates with average salary.

[27]:
```
dfEd = remove_outliers(df, 'ConvertedCompYearly')  #Filters data within the␣
 ↪salary bounds to remove outliers.
dfEd = dfEd.dropna(subset=['EdLevel', 'ConvertedCompYearly'])
group3 = dfEd.groupby('EdLevel')['ConvertedCompYearly'].mean().reset_index()␣
 ↪#Calculates average salary by education level.

Edlvlnames = {
    'Associate degree (A.A., A.S., etc.)': 'Associate degree',
    'Bachelor's degree (B.A., B.S., B.Eng., etc.)': 'Bachelor's degree',
    'Master's degree (M.A., M.S., M.Eng., MBA, etc.)': 'Master's degree',
    'Some college/university study without earning a degree': 'Some college, no␣
 ↪degree',
    'Secondary school (e.g. American high school, German Realschule or␣
 ↪Gymnasium, etc.)':'Secondary school',
    'Professional degree (JD, MD, Ph.D, Ed.D, etc.)':'Professional degree'

}
group3 = group3.replace(Edlvlnames)  # Replace DataFrame values
```

1.15.1 Line Plot: Education Level vs Annual Compensation

- This line plot visualizes the average yearly compensation by education level with markers for each data point.
- Axis labels and a descriptive title improve clarity.
- The x-axis labels are rotated for better readability due to long category names.
- The percentage difference in salary between holders of a professional degree and those with only secondary education is calculated and printed.
- The plot layout is adjusted for size and font for better presentation.

This analysis highlights the impact of education level on annual compensation, showing significant salary differences across education tiers.

[28]:
```
fig = px.line(group3,       #Creates a line plot of average salary by education␣
 ↪level with markers and labels.
              x='EdLevel',
              y='ConvertedCompYearly',
              markers=True,
```

```
            title='Relationship Between Education Level and Converted Yearly␣
 ↪Compensation',
            labels={
                'EdLevel': 'Education Level',
                'ConvertedCompYearly': 'Converted Compensation (Yearly)'
            })

high = group3[group3['EdLevel'] == 'Professional␣
 ↪degree']['ConvertedCompYearly'].values[0]
low = group3[group3['EdLevel'] == 'Secondary school']['ConvertedCompYearly'].
 ↪values[0]

# Calcular el porcentaje
diff_percent = ((high - low) / low) * 100

print(f"Professional degree holders earn {diff_percent:.1f}% more than those␣
 ↪with only secondary education.")


fig.update_layout(   #Updates plot layout with size, fonts, and rotated x-axis␣
 ↪labels.
    width=1300,
    height=500,
    title_font_size=18,
    xaxis_title_font_size=14,
    yaxis_title_font_size=14,
    xaxis_tickangle=270,
    margin=dict(t=80, b=210)
)
fig.add_annotation( # Adds centered caption below the plot
    text="Figure 10. Relationship Between Education Level and Converted Yearly␣
 ↪Compensation",
    xref="paper", yref="paper",
    x=0.5, y=-1.1,
    font=dict(size=12),
    align="center"
)
fig.show()
```

```
Professional degree holders earn 57.4% more than those with only secondary
education.
```

1.15.2 Conclusions: Education Level vs Annual Compensation

- **Professional degree holders** report the **highest average annual salary**, surpassing **$80,000 USD**, indicating a strong return on advanced credentials in specific fields.
- **Bachelor's and Master's degree holders** earn similarly, both averaging around **$73,000**

USD, showing that a Master's degree may not drastically increase salary in all cases.

- Individuals with only **primary or secondary education** report significantly **lower salaries** (around **$52,000 – $54,000 USD**), highlighting the economic value of higher education.
- Those with **some college but no degree** earn about **$67,000 USD**, suggesting partial post-secondary education still has a positive impact on earnings.
- Interestingly, **Associate degree** holders earn nearly as much as Bachelor's graduates, with average compensation near **$68,000 USD**.
- The **"Something else"** category has one of the lowest reported salaries (~**$58,000 USD**), possibly due to unconventional or unrecognized education pathways.
- Overall, while **formal education generally correlates with higher compensation**, there are exceptions that may reflect job market realities, specific professional credentials, or industry variation.

1.16 Scatter Plot: Normalized Compensation vs. Coding Experience Colored by Job Satisfaction

- Select relevant columns: `YearsCode`, `ConvertedCompYearly`, and `JobSat`.
- Filter rows to include only salaries within the predefined bounds and drop rows with missing values in any of these columns. This ensures that subsequent data transformations (like extracting numeric values) and visualizations are accurate and not affected by incomplete or missing data, which could cause errors or bias.
- Extract numeric values from the `YearsCode` column, which may contain text, to get the number of years of coding experience.
- Retrieve unique values in the `YearsCode` column to understand the range of coding experience reported.

This step prepares the dataset for analyzing relationships between coding experience, salary, and job satisfaction.

[29]:
```
dfC = df[['YearsCode', 'ConvertedCompYearly', 'JobSat']] #Selects years coding, salary, and job satisfaction columns.
dfC = remove_outliers(dfC, 'ConvertedCompYearly')  #Filters data within the salary bounds to remove outliers.
dfC = dfC.dropna(subset=['YearsCode', 'ConvertedCompYearly', 'JobSat'])
print(dfC['YearsCode'].unique()) #Gets unique values in the YearsCode column.
dfC['YearsCode'] = dfC['YearsCode'].str.extract(r'(\d+)') #Extracts the number of years coded from the text.
```

```
['3' '15' '7' '32' '38' '21' '10' '6' '40' '20' '9' '25' '12' '14' '11'
 '16' '18' '28' '37' '30' '47' '17' '36' '19' '24' '2' '5' '34' '23' '26'
 '13' '33' '31' '45' '22' '4' '35' '27' '48' '8' '42' '29'
 'More than 50 years' '39' '43' '44' '50' '1' 'Less than 1 year' '46' '41'
 '49']
```

1.16.1 Normalizing Data Using Min-Max Scaling

- Initialize a `MinMaxScaler` to normalize data features to a range between 0 and 1.

- Apply the scaler to the dataframe containing years of coding experience, salary, and job satisfaction.
- Create a new dataframe with the scaled values, preserving original column names.

Normalization facilitates comparison and modeling by bringing all variables to the same scale.

[30]:
```
scaler = MinMaxScaler() #Initializes a MinMaxScaler for normalization.
dfC_scaled = pd.DataFrame(scaler.fit_transform(dfC), columns=dfC.columns)␣
 ↪#Scales all dataframe columns to a 0-1 range.
```

1.16.2 Scatter Plot: Years of Coding vs. Salary Colored by Job Satisfaction (Normalized)

- Group the normalized data by years of coding experience and calculate mean values.
- Create a scatter plot comparing normalized yearly compensation against normalized years of coding.
- Color each point based on normalized job satisfaction levels, using the 'viridis' colormap.
- Axis labels and title describe the variables and purpose.
- A colorbar shows the gradient for job satisfaction values.

This visualization reveals potential relationships between coding experience, salary, and job satisfaction on a normalized scale.

[31]:
```
dfC_grouped = dfC_scaled.groupby('YearsCode').mean().reset_index() #Calculates␣
 ↪average scaled values grouped by years coding.

plt.figure(figsize=(10, 6))  # Initialize new matplotlib figure
plt.scatter(dfC_grouped['YearsCode'], dfC_grouped['ConvertedCompYearly'],␣
 ↪c=dfC_grouped['JobSat'], cmap='viridis', s=100) #Creates a scatter plot of␣
 ↪salary vs. years coded, colored by job satisfaction.

plt.xlabel('Years of Coding (Normalized)') # Set x-axis label
plt.ylabel('Yearly Compensation (Normalized)') # Set y-axis label
plt.title('Comparison of Years of Coding, Compensation, and Job Satisfaction␣
 ↪(Normalized)')  # Set plot title

plt.colorbar(label='Job Satisfaction (Normalized)') #Adds a colorbar.
plt.figtext(0.5, -0.05,'Figure 11. Comparison of Years of Coding, Compensation,␣
 ↪and Job Satisfaction', # Adds centered caption below the plot
            wrap=True, horizontalalignment='center', fontsize=10)
plt.show()  # Display the plot
```

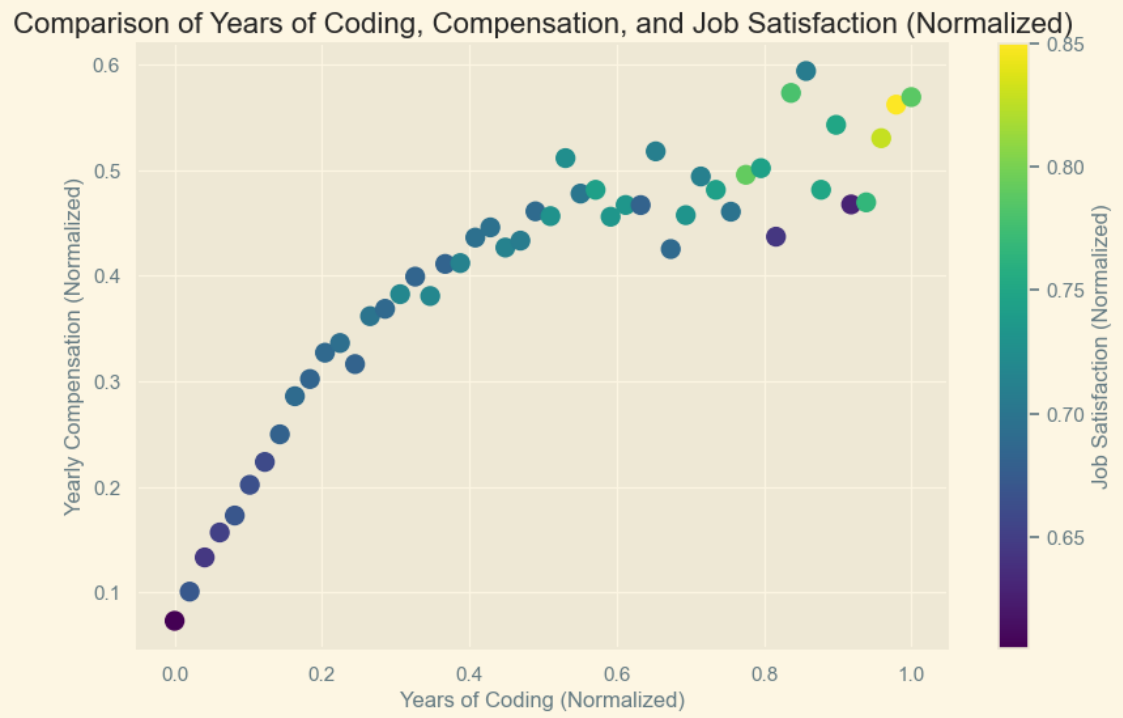

Figure 11. Comparison of Years of Coding, Compensation, and Job Satisfaction

1.16.3 Conclusions: Coding Experience vs Compensation & Job Satisfaction (Normalized)

- A clear **positive correlation** exists between **years of coding** and **annual compensation**: more experienced coders tend to earn higher salaries.
- **Job satisfaction** also increases moderately with coding experience and compensation, though with some variation.
- The highest job satisfaction (in yellow-green tones) appears toward the upper right, among those with **high compensation and significant experience**.
- Despite normalization, a few outliers show **lower job satisfaction** even at high compensation levels, hinting at non-monetary factors influencing satisfaction.
- The curve flattens slightly after the midpoint, suggesting **diminishing returns** in compensation with increased experience beyond a certain point.
- Normalization allowed for better comparative analysis, though specific numeric values are abstracted.

1.17 FINAL CONCLUSIONS

- **Work experience** has a strong positive correlation with salary, especially in the first **5–10 years**, where annual compensation increases sharply. Salary growth slows after 15+ years.

- **Age** shows a similar pattern: compensation rises steadily with age until around 60, then begins to decline slightly, indicating that career growth tends to level off in the later stages."
- **Education level** impacts earnings significantly, those with **professional, bachelor's, or master's degrees** earn the highest salaries, while primary/secondary education yields the lowest.
- **Remote workers** earn the most on average (**$78,700 USD**), followed by hybrid workers, while **in-person workers earn 33% less**, highlighting the market value of work flexibility.
- The relationship between **years of coding, compensation, and job satisfaction (normalized)** confirms a trend: more experience leads to higher pay and generally higher satisfaction.
- **React** is the most preferred web framework among developers, followed by **Node.js**, **Next.js**, and **Vue.js**, with JavaScript frameworks dominating modern web development.
- **JavaScript**, **HTML/CSS**, and **Python** are the most used and most wanted programming languages globally, particularly in countries like the **USA**, **India**, and the **UK**.
- **USA** has the highest number of developers, followed by **India**, **Germany**, and the **UK**. Representation drops off in regions like Africa, South America, and parts of Asia.
- Median **compensation by country** reflects economic differences, **USA** leads, followed by **Canada** and the **UK**, while **India** and **Brazil** have lower medians but many outliers.
- **Database usage** by age group shows that **PostgreSQL** and **MySQL** are dominant across all groups, especially ages **18–34**. Usage declines with age but remains consistent in younger respondents.
- In **top programming languages wanted by country**, trends largely mirror actual usage: developers tend to want to work with the technologies they already use, with **Python**, **TypeScript**, and **JavaScript** leading interest.

These findings provide a holistic view of how experience, age, education, work modality, and regional context influence compensation, preferences, and satisfaction in the software development industry.