```
[notice] A new release of pip is available: 23.3.1 -> 24.0
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

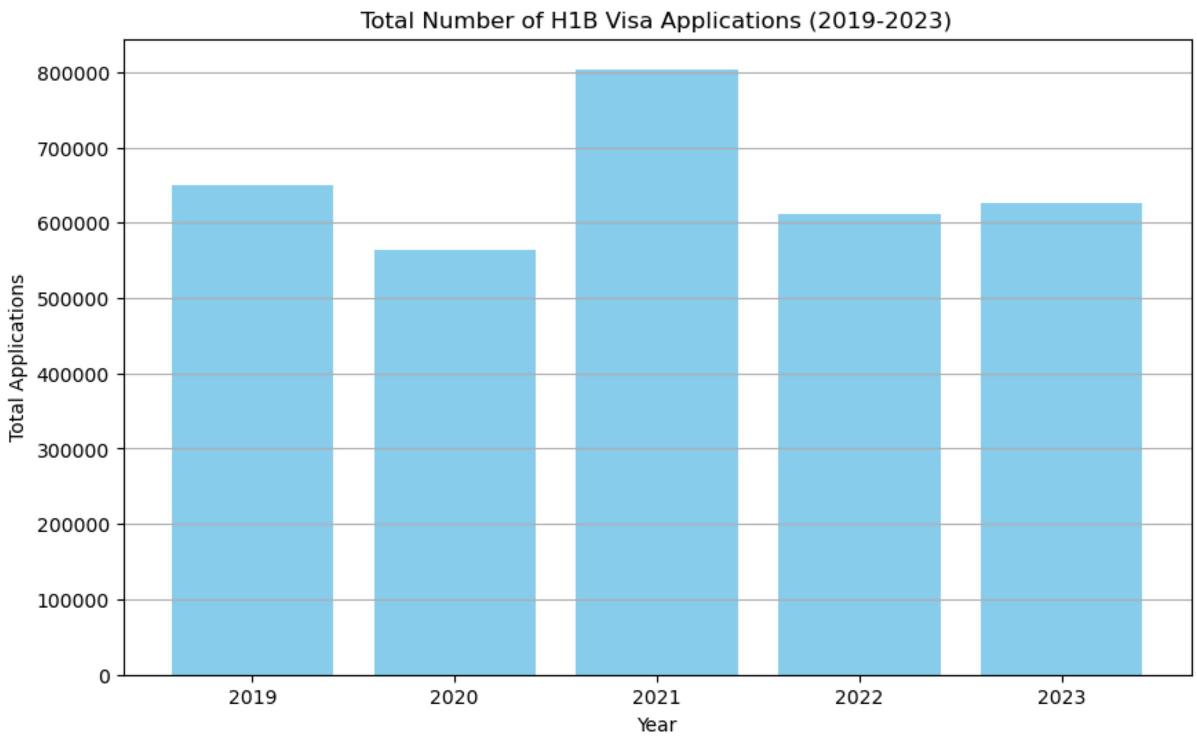
Discover The Pattern of H1-B Visa From 2019-2023

In [27]:
```
# import

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
import geopandas as gpd
import requests
import folium

import pingouin as pg
from urllib.request import urlopen
from io import BytesIO
import matplotlib.pyplot as plt
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd
```

In [47]:
```

# Load datasets
y2019 = pd.read_csv('./LCA_FY_2019.csv', low_memory=False)
y2020 = pd.read_csv('./LCA_FY_2020.csv', low_memory=False)
y2021 = pd.read_csv('./LCA_FY_2021.csv', low_memory=False)
y2022 = pd.read_csv('./LCA_FY_2022.csv', low_memory=False)
y2023 = pd.read_csv('./LCA_FY_2023.csv', low_memory=False)

#y2019.shape
#y2020.shape
#y2021.shape
y2023.columns.tolist()
```

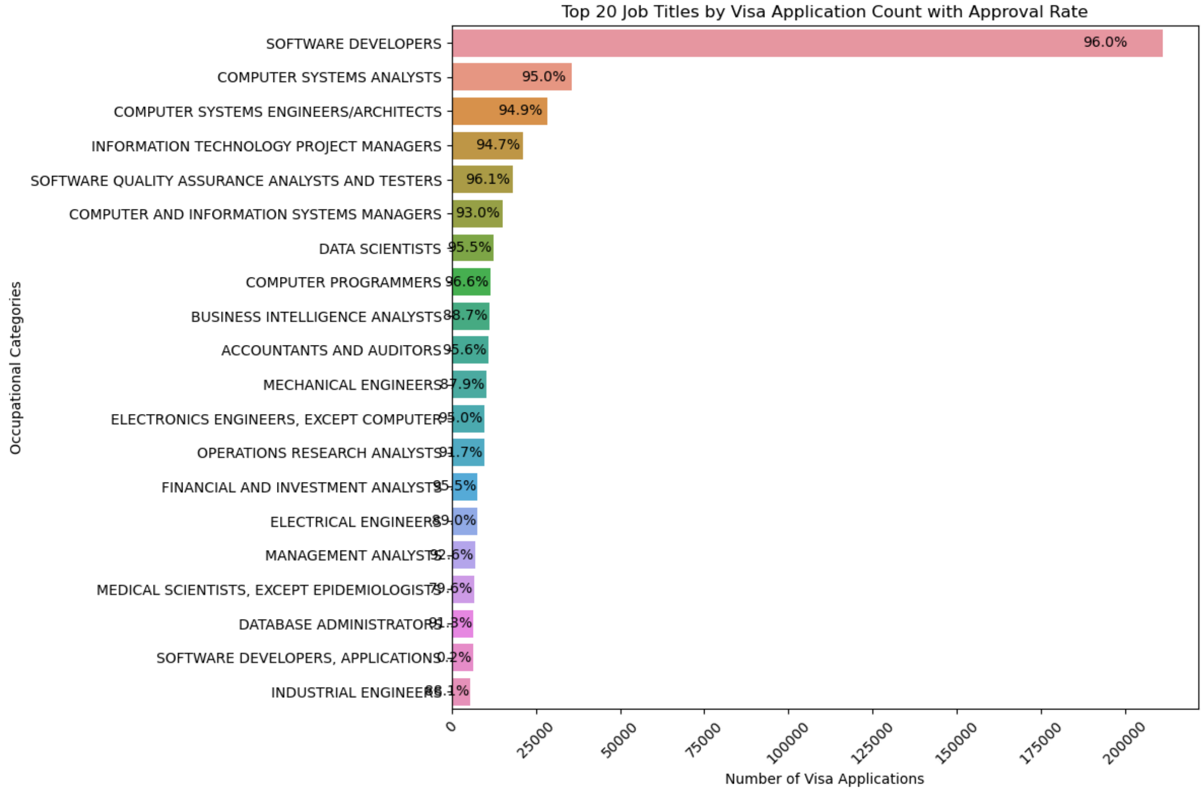
Out[47]:
```
['Visa_Class',
 'Job_Title',
 'Employer_Name',
 'SOC_Title',
 'Full_Time_Position',
 'Prevailing_Wage',
 'Unit_Of_Pay',
 'Employer_Country',
 'Case_Status',
 'Worksite',
 'Employer_Location']
```

In [6]:
```
# Convert 'Case_Status' values to uppercase
for df in [y2019, y2020, y2021, y2022, y2023]:
    df['Case_Status'] = df['Case_Status'].str.upper()


# Calculate total number of applications for each year
total_applications_per_year = [len(df) for df in [y2019, y2020, y2021, y:

# Plot total number of applications across the five years
years = ['2019', '2020', '2021', '2022', '2023']
plt.figure(figsize=(10, 6))
plt.bar(years, total_applications_per_year, color='skyblue')
plt.title('Total Number of H1B Visa Applications (2019-2023)')
plt.xlabel('Year')
plt.ylabel('Total Applications')
plt.grid(axis='y')
plt.show()
```

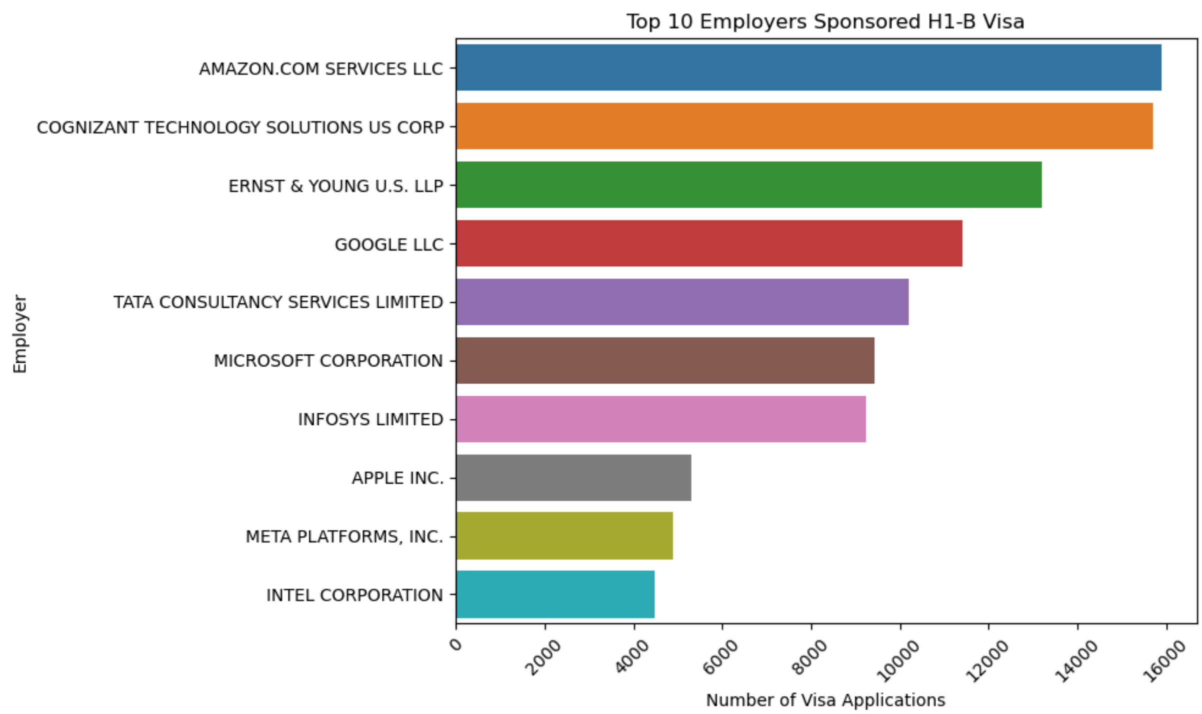


Top 20 Job Titles That Was Sponsored And Their Approval Rate

In [8]:
```
# Group the data by job title and count the number of visa applications
job_title_counts = y2023['SOC_Title'].str.upper().str.strip().value_coun
# Count the number of Certified application by Job Titles
certified_counts = y2023[y2023['Case_Status'] == 'Certified']['SOC_Title


# Make the DataFrame
job_title_df = job_title_counts.reset_index()
job_title_df.columns = ['Job Title', 'Total Applications']

certified_df = certified_counts.reset_index()
certified_df.columns = ['Job Title', 'Certified Applications']

# Merge dataframes on Job Title
final_df = pd.merge(job_title_df, certified_df, on='Job Title', how='lef

# Calculate the percentage of certified applications
final_df['Certified %'] = (final_df['Certified Applications'] / final_df

# Display the DataFrame for verification
#print(final_df)

# Create a horizontal bar plot with certified percentages
plt.figure(figsize=(12, 8))
bar = sns.barplot(x='Total Applications', y='Job Title', data=final_df)

# Add text annotations for the percentage of certified applications
for index, row in final_df.iterrows():
    # Formatting the label to show percentage with one decimal
    label = f"{row['Certified %']:.1f}%"
    # Adding the text inside the bar
    plt.text(row['Total Applications'] - (row['Total Applications'] * 0.(

plt.title('Top 20 Job Titles by Visa Application Count with Approval Rat(
plt.xlabel('Number of Visa Applications')
plt.ylabel('Occupational Categories')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```


Top 20 Job Titles by Visa Application Count with Approval Rate
SOFTWARE DEVELOPERS 96.0%
COMPUTER SYSTEMS ANALYSTS 95.0%
COMPUTER SYSTEMS ENGINEERS/ARCHITECTS 94.9%
INFORMATION TECHNOLOGY PROJECT MANAGERS 94.7%
SOFTWARE QUALITY ASSURANCE ANALYSTS AND TESTERS 96.1%
COMPUTER AND INFORMATION SYSTEMS MANAGERS 93.0%
DATA SCIENTISTS 95.5%
COMPUTER PROGRAMMERS 96.6%
BUSINESS INTELLIGENCE ANALYSTS 88.7%
ACCOUNTANTS AND AUDITORS 95.6%
MECHANICAL ENGINEERS 87.9%
ELECTRONICS ENGINEERS, EXCEPT COMPUTER 95.0%
OPERATIONS RESEARCH ANALYSTS 91.7%
FINANCIAL AND INVESTMENT ANALYSTS 95.5%
ELECTRICAL ENGINEERS 89.0%
MANAGEMENT ANALYSTS 92.6%
MEDICAL SCIENTISTS, EXCEPT EPIDEMIOLOGISTS 79.6%
DATABASE ADMINISTRATORS 91.3%
SOFTWARE DEVELOPERS, APPLICATIONS 0.2%
INDUSTRIAL ENGINEERS 86.1%
Occupational Categories
0
25000
50000
75000
100000
125000
150000
175000
200000
Number of Visa Applications

In [38]:
```
employer_count = y2023['Employer_Name'].str.upper().str.strip().value_co

# Create DataFrame
employer_df = employer_count.reset_index()
employer_df.columns = ['Employer Name', 'Number of Visa Applications']

# Create a horizontal bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Number of Visa Applications', y='Employer Name', data=emp
plt.title('Top 10 Employers Sponsored H1-B Visa')
plt.xlabel('Number of Visa Applications')
plt.ylabel('Employer')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



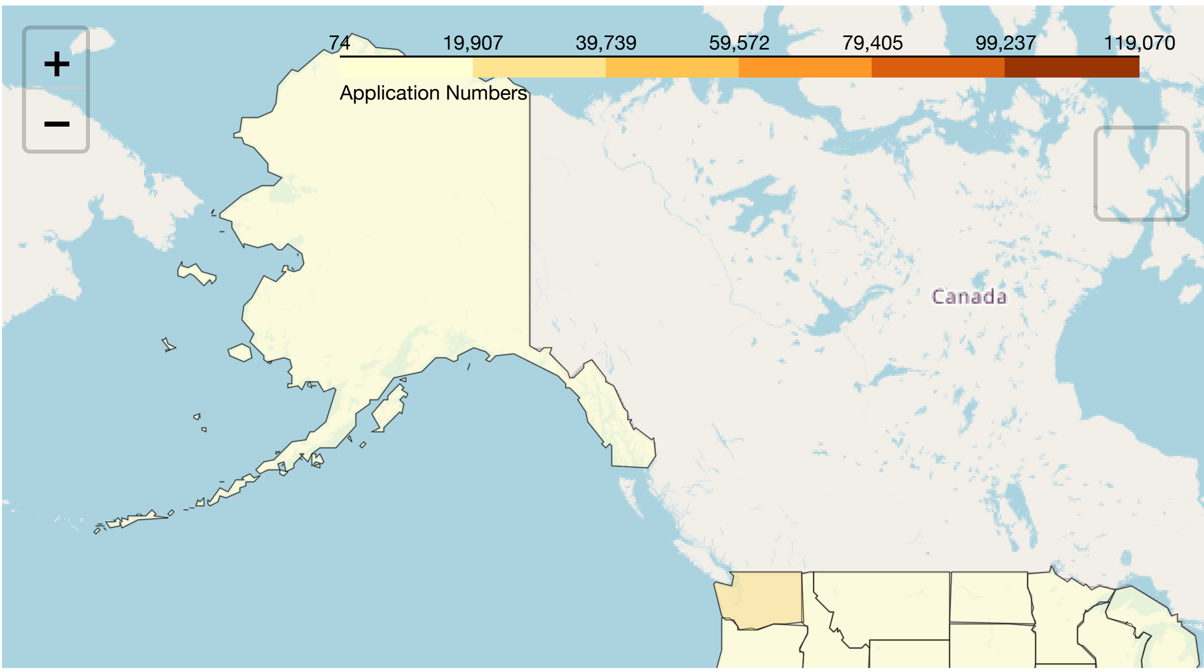
Spatial Visualization of Application Across States

Hover across states to explore the top H1-B job sectors that represent that state.

In [12]:
```
# Read the
gdf = gpd.read_file("./USShapefiles/tl_2023_us_state.shp")
# Display gdf
#print(gdf)

#gdf.explore()

# Make a copy of y2023 datasets for further modification
data = y2023
data['STUSPS'] = data['Worksite'].apply(lambda x: x.split(',')[-1].strip

# Get the top SOC_Title for each state
top_soc = data.groupby('STUSPS')['SOC_Title'].agg(lambda x: x.value_coun
top_soc.columns = ['STUSPS', 'Top_SOC_Title']

#Count the number of application for each state
state_counts = data['STUSPS'].value_counts().reset_index()
state_counts.columns = ['STUSPS', 'Number of Applications']
#state_counts

# Perform the join
formapping = pd.merge(gdf, state_counts, on='STUSPS', how='left')
formapping = pd.merge(formapping, top_soc, on='STUSPS', how='left')

#print(formapping)
gdata = formapping.to_json()
#print(gdata)

m = folium.Map(location=[48, -102], zoom_start=3)

folium.Choropleth(
    geo_data=gdata,
    name='choropleth',
    data=formapping,
    columns=["STUSPS", "Number of Applications"],
    key_on="feature.properties.STUSPS",
    fill_color="YlOrBr",
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name="Application Numbers",
).add_to(m)

# Function to display additional data on hover
style_function = lambda x: {
    'fillColor': '#ffffff',
    'color':'#000000',
    'fillOpacity': 0.1,
    'weight': 0.1
}

highlight_function = lambda x: {
    'fillColor': '#000000',
    'color':'#000000',
    'fillOpacity': 0.50,
    'weight': 0.1
}
```

```
# Add Hover functionality
info = folium.features.GeoJson(
    data = gdata,
    control=False,
    style_function=style_function,
    highlight_function=highlight_function,
    tooltip=folium.features.GeoJsonTooltip(
        fields=['STUSPS', 'Number of Applications', 'Top_SOC_Title'],
        aliases=['State: ', 'Number of Applications: ', 'Top Job Title:
        style=("background-color: white; color: #333333; font-family: Ar
    )
)
m.add_child(info)
m.keep_in_front(info)

folium.LayerControl().add_to(m)

m.save("my_folium_map.html")

m
```

Out[12]:

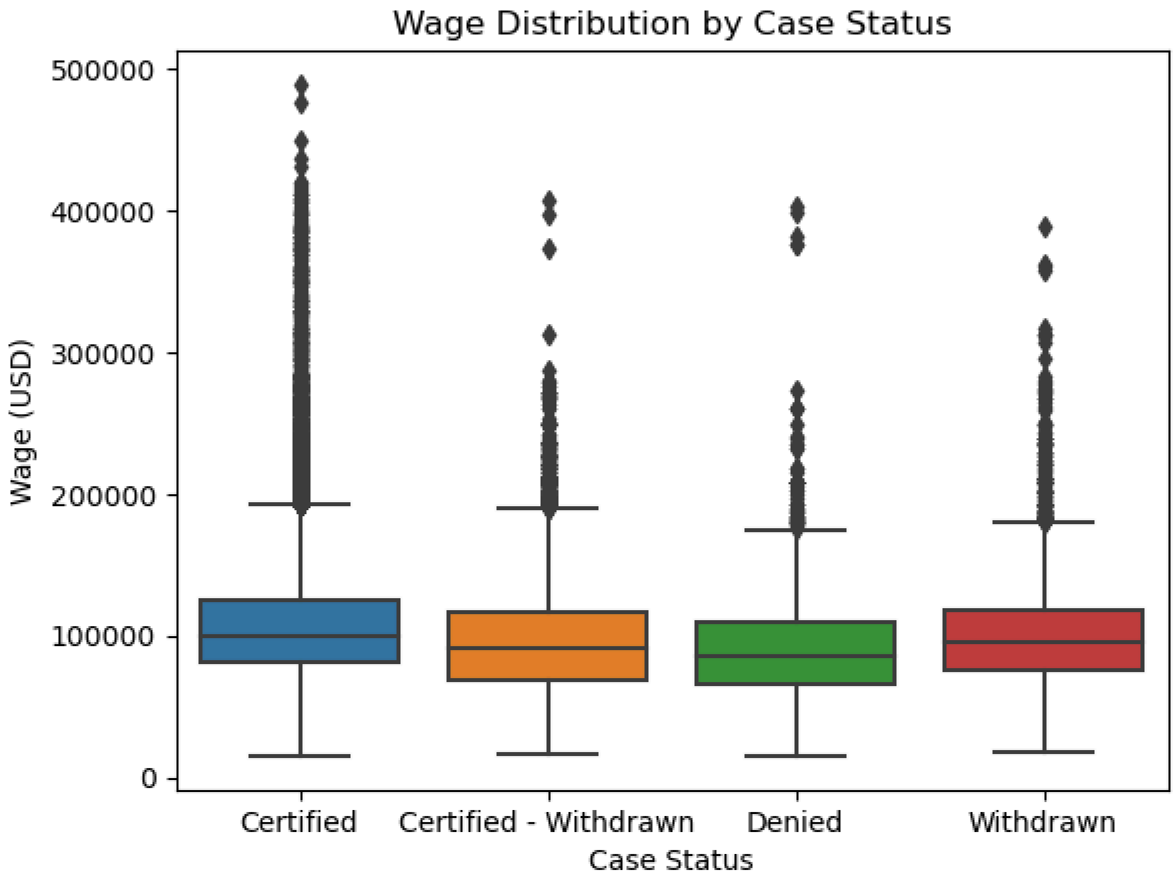


Software Developer is dominant among other occupations. As in Alaska, Elementary School Teacher is the top job title for H1-B visa. In Montana and North Dakota, Medical and Laboratory Technologist is in most demand for H1-B visa. California is in 1st place that sponsors for H1-B visa with the top occupation of Software Developer. As expected, since California is the global center for high technology and innovation where Sillicon Valley located. It is already accounted for 1/6 of the total H1-B applications across the US. Georgia is ranked in 6th place with the Software Developer occupation as well.

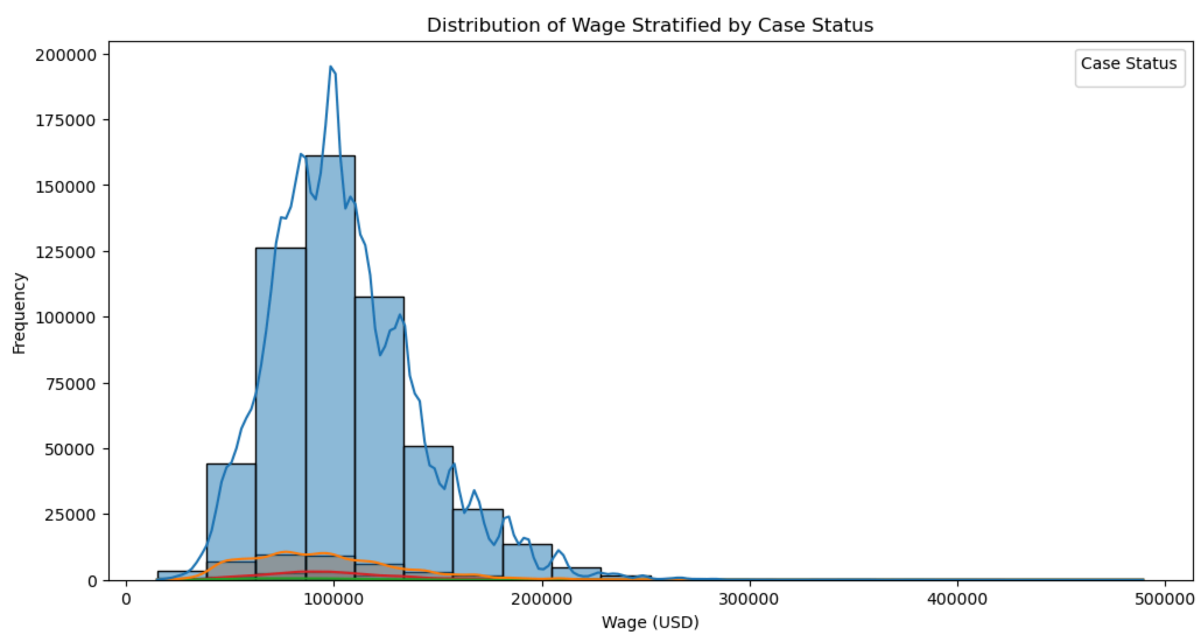
Descriptive Analysis Of Prevailing Wage Among Applicants

In [13]:
```
#print(y2023['Prevailing_Wage'].dtype)
# Filter the dataset where the 'Unit_Of_Pay' is 'Year'
yearly_wages = y2023[y2023['Unit_Of_Pay'] == 'Year']

# Convert 'Prevailing_Wage' column to numeric
yearly_wages['Prevailing_Wage'] = pd.to_numeric(yearly_wages['Prevailing_

# Descriptive Statistics
desc_stats = yearly_wages.groupby('Case_Status')['Prevailing_Wage'].desc
print(desc_stats)

# Visualization
sns.boxplot(x='Case_Status', y='Prevailing_Wage', data=yearly_wages)
plt.title('Wage Distribution by Case Status')
plt.xlabel('Case Status')
plt.ylabel('Wage (USD)')
plt.show()


# Create a histogram for each origin category to visualize the distribut
plt.figure(figsize=(12, 6))
sns.histplot(data=yearly_wages, x='Prevailing_Wage', hue='Case_Status',
plt.title('Distribution of Wage Stratified by Case Status')
plt.xlabel('Wage (USD)')
plt.ylabel('Frequency')
plt.legend(title='Case Status')
plt.show()
```

```
/var/folders/5g/k3w_fwxs7y9ghhj9bf_7bp2c0000gn/T/ipykernel_10603/337594
0575.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (ht
tps://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy)
  yearly_wages['Prevailing_Wage'] = pd.to_numeric(yearly_wages['Prevail
ing_Wage'], errors='coerce')

                                mean       50%            std
Case_Status
Certified               105185.439545  100069.0  35738.801541
Certified - Withdrawn    95098.152835   90750.0  36915.303262
Denied                   92076.440431   85842.0  39448.788410
Withdrawn               100581.078035   95701.0  36469.657967
```



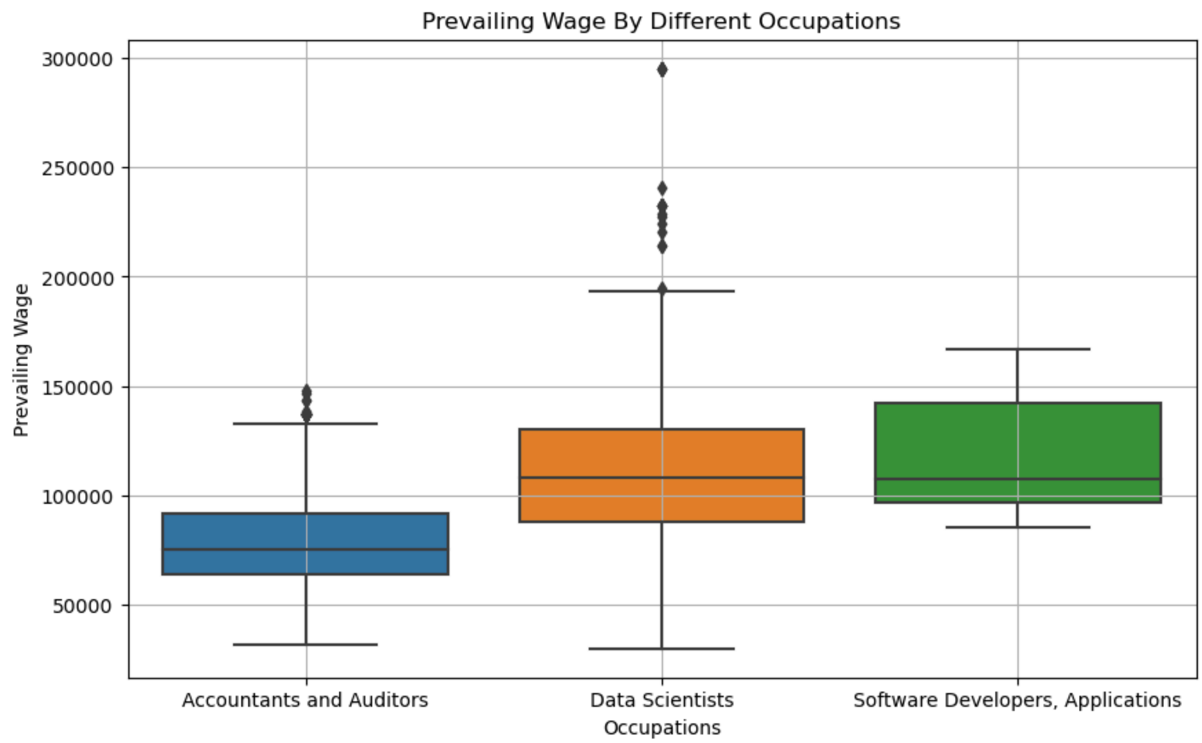

```
No artists with labels found to put in legend.  Note that artists whose
label start with an underscore are ignored when legend() is called with
no argument.
```



Test if there is a significant difference in prevailing wage between "Software Developers, Accountants and Auditors, and Data Scientists

In [50]:
```
# Filter data for certified applications and specified job titles
certified_data = y2023[y2023['Case_Status'] == 'Certified']
job_titles = ['Software Developers, Applications', 'Accountants and Audi
filtered_data = certified_data[certified_data['SOC_Title'].isin(job_titl

# Filter the dataset where the 'Unit_Of_Pay' is 'Year'
filtered_data = filtered_data[filtered_data['Unit_Of_Pay'] == 'Year']
filtered_data['Prevailing_Wage'] = pd.to_numeric(filtered_data['Prevaili

# Visualizing the differences in wage
plt.figure(figsize=(10, 6))
sns.boxplot(x='SOC_Title', y='Prevailing_Wage', data=filtered_data)
plt.title('Prevailing Wage By Different Occupations')
plt.ylabel('Prevailing Wage')
plt.xlabel('Occupations')
plt.grid(True)
plt.show()

# Conducting Levene's test for equality of variances
levene_test = stats.levene(*[group['Prevailing_Wage'].values for name, g
print(f"Levene's test: Statistic={levene_test.statistic}, p-value={leven

if levene_test.pvalue < 0.05:
    print("Variances are unequal, proceeding with Welch's ANOVA.")
    # Conducting Welch's ANOVA
    anova_results = pg.welch_anova(dv='Prevailing_Wage', between='SOC_Ti
    print(anova_results)
else:
    # Conducting conventional ANOVA
    model = ols('Prevailing_Wage ~ C(SOC_Title)', data=filtered_data).fi
    anova_results = sm.stats.anova_lm(model, typ=2)
    print(anova_results)

    # Visualizing F-distribution with critical values
    fig, ax = plt.subplots(1, 1, figsize=(10, 6))

    # Plotting F-distribution
    dfn, dfd = model.df_model, model.df_resid  # degrees of freedom
    f_val = anova_results['F'][0]  # F-value from our ANOVA test
    x = np.linspace(stats.f.ppf(0.01, dfn, dfd), stats.f.ppf(0.99, dfn, 
    ax.plot(x, stats.f.pdf(x, dfn, dfd), 'r-', lw=2, label='F-distributi

    # Highlighting critical region
    critical_value = stats.f.ppf(0.95, dfn, dfd)  # 95% confidence
    ax.fill_between(x, 0, stats.f.pdf(x, dfn, dfd), where=(x > critical_

    # Marking the F-value from our test
    ax.axvline(f_val, color='black', lw=2, linestyle='--', label=f'F-val

    ax.legend(loc='best')
    ax.set_title('F-distribution with F-value and Critical Region')
    ax.set_xlabel('F-value')
    ax.set_ylabel('Probability Density')
    plt.show()

# Conduct Tukey's HSD test for wage across different occupations
tukey_results = pairwise_tukeyhsd(endog=filtered_data['Prevailing_Wage']
```

```
# Print the results
print(tukey_results)
```

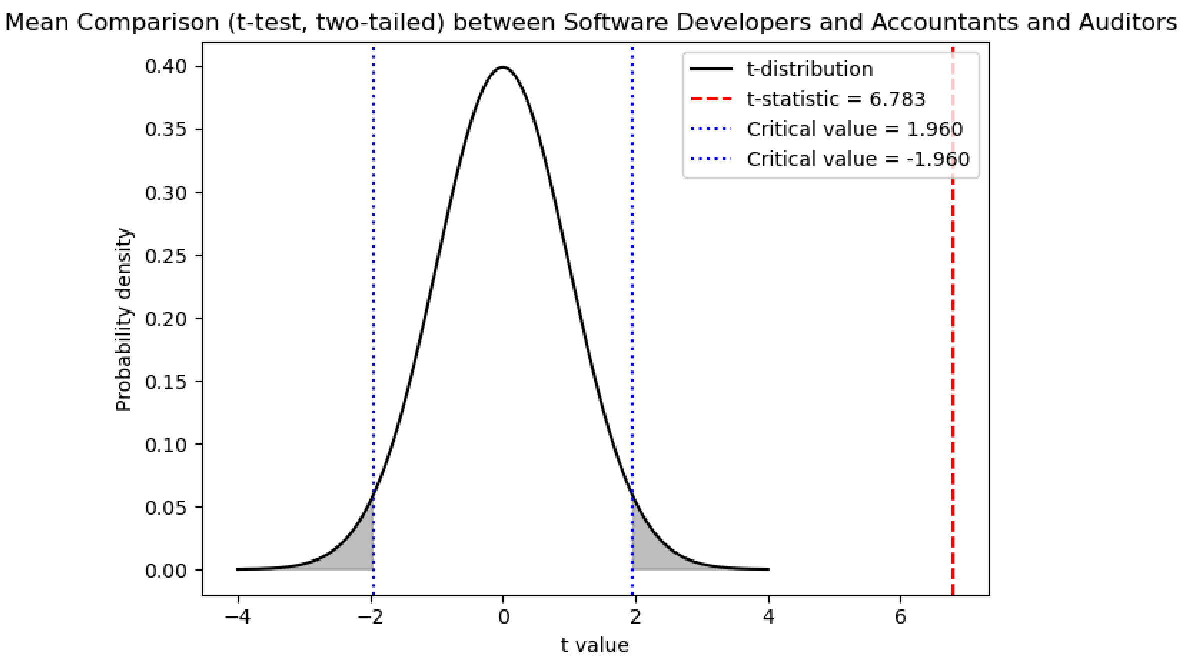



```
Levene's test: Statistic=402.6334631111706, p-value=2.1616634533657138e
-172
Variances are unequal, proceeding with Welch's ANOVA.
      Source  ddof1      ddof2          F         p-unc       np2
0  SOC_Title      2  37.344115  3948.83407  3.499821e-44  0.262109
                           Multiple Comparison of Means - Tukey HSD, FW
ER=0.05
=======================================================================
==================================
         group1                            group2               meandiff  p
-adj    lower      upper    reject
-----------------------------------------------------------------------
----------------------------------
Accountants and Auditors                  Data Scientists 31575.0892
0.0 30727.5485 32422.6298   True
Accountants and Auditors Software Developers, Applications 37962.9769
0.0 21969.2572 53956.6966   True
         Data Scientists Software Developers, Applications  6387.8878
0.6173 -9603.9793 22379.7548  False
-----------------------------------------------------------------------
----------------------------------
```

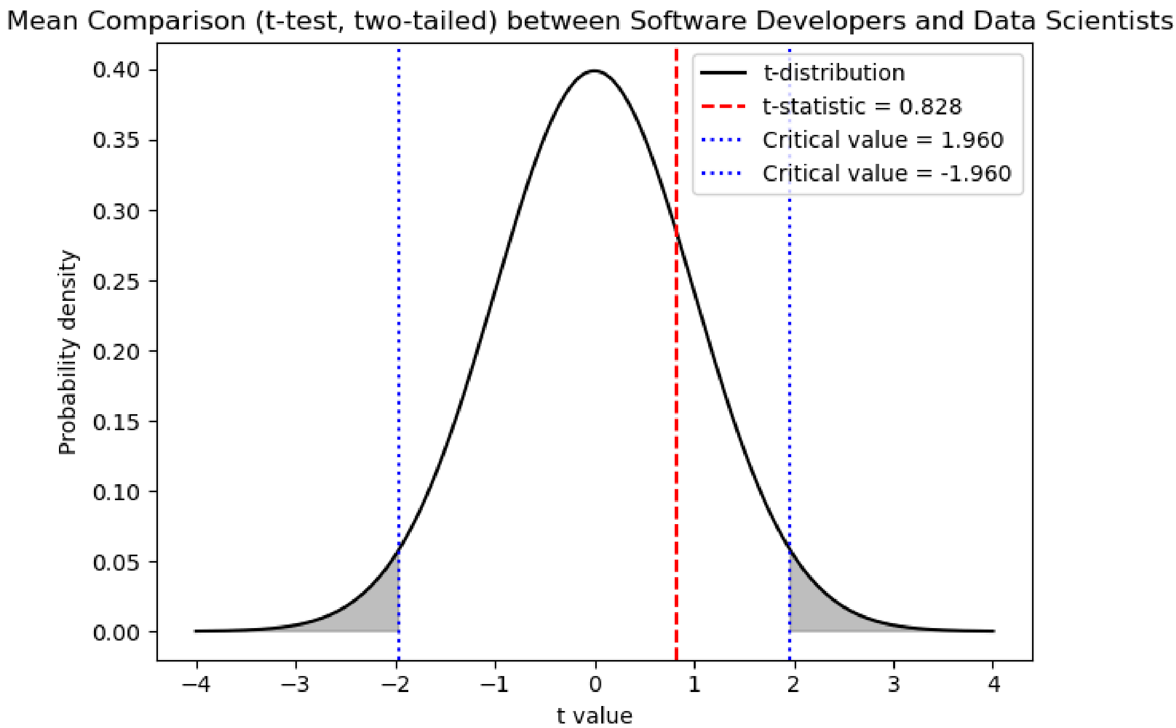
In [54]:
```
from scipy.stats import ttest_ind, levene, t, f
# Filter the DataFrame for developers' wage and accountants' wage
dev_wages = filtered_data[filtered_data['SOC_Title'] == 'Software Develop
acct_wages = filtered_data[filtered_data['SOC_Title'] == 'Accountants and
data_sci_wages = filtered_data[filtered_data['SOC_Title'] == 'Data Scient

# Define pairs for comparison
pairs = [
    (dev_wages, acct_wages, 'Software Developers', 'Accountants and Audit
    (dev_wages, data_sci_wages, 'Software Developers', 'Data Scientists'
    (acct_wages, data_sci_wages, 'Accountants and Auditors', 'Data Scient
]

# Function to perform t-tests and visualize results
def perform_ttest_and_visualize(wages1, wages2, label1, label2):
    # Set the confidence level
    alpha = 0.05  # Significance level
    test_type = "two-tailed"  # Can be "one-tailed" or "two-tailed"

    if len(wages1) < 2 or len(wages2) < 2:
        print(f"Insufficient data to conduct the test between {label1} an
        return

    else:
        # Conduct Levene's test for equality of variances
        statistic, p_value_levene = levene(wages1, wages2)
        print("Levene's Test - Statistic:", statistic)
        print("Levene's Test - P-Value:", p_value_levene)

        # Determine if variances are equal or not
        equal_var = p_value_levene > alpha
        variances_text = "equal" if equal_var else "unequal"
        print(f"Since p-value {'>' if equal_var else '<='} {alpha}, varia

        # Conduct the appropriate type of t-test
        t_statistic, p_value_ttest = ttest_ind(wages1, wages2, equal_var=

        # Adjust p-value and critical value for one-tailed test, if speci
        if test_type == "one-tailed":
            p_value_ttest /= 2  # Halve the p-value for one-tailed test

        print(f"T-Test - Statistic: {t_statistic}")
        print(f"T-Test - P-Value: {p_value_ttest} (adjusted for {test_typ

        # Degrees of freedom for the t-test
        df_t = len(wages1) + len(wages2) - 2

        # Adjust alpha for one-tailed test
        critical_alpha = alpha / 2 if test_type == "two-tailed" else alph
        t_critical = t.ppf(1 - critical_alpha, df_t)  # Critical value fo

    # Make the plot
    t_values = np.linspace(-4, 4, 1000)
    plt.plot(t_values, t.pdf(t_values, df_t), 'k-', label='t-distributio
    plt.axvline(x=t_statistic, color='red', linestyle='--', label=f't-sta
    plt.axvline(x=t_critical, color='blue', linestyle=':', label=f'Criti
    if test_type == "two-tailed":
```

```
        plt.axvline(x=-t_critical, color='blue', linestyle=':', label=f'(
        plt.fill_between(t_values, 0, t.pdf(t_values, df_t), where=(t_va
    else:
        plt.fill_between(t_values, 0, t.pdf(t_values, df_t), where=t_val
    plt.title(f'Mean Comparison (t-test, {test_type}) between {label1} an
    plt.xlabel('t value')
    plt.ylabel('Probability density')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Perform tests for all pairs
for wages1, wages2, label1, label2 in pairs:
    perform_ttest_and_visualize(wages1, wages2, label1, label2)
```

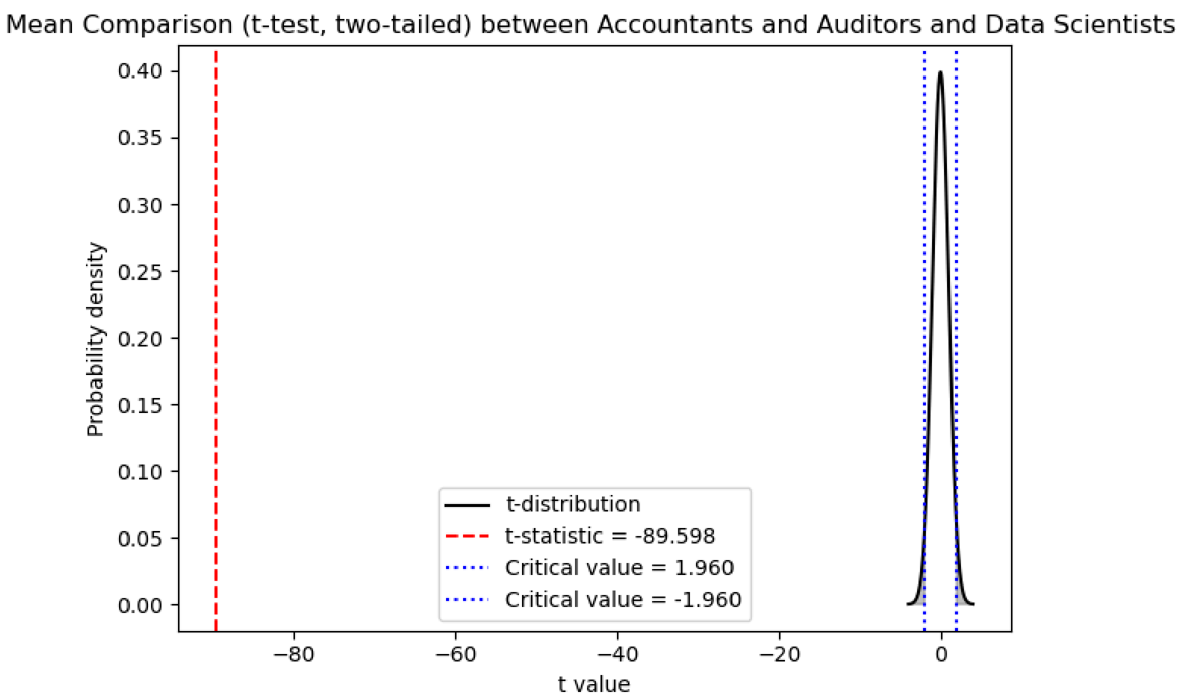
```
Levene's Test - Statistic: 1.9746275010306096
Levene's Test - P-Value: 0.15998901722608638
Since p-value > 0.05, variances are assumed to be equal.
T-Test - Statistic: 6.782524890809181
T-Test - P-Value: 1.2486511053383116e-11 (adjusted for two-tailed test)
```




```
Levene's Test - Statistic: 0.08773541640593217
Levene's Test - P-Value: 0.767081374874681
Since p-value > 0.05, variances are assumed to be equal.
T-Test - Statistic: 0.8282627278649924
T-Test - P-Value: 0.4075387093645133 (adjusted for two-tailed test)
```




```
Levene's Test - Statistic: 805.595628994482
Levene's Test - P-Value: 5.301012185768495e-174
Since p-value <= 0.05, variances are assumed to be unequal.
T-Test - Statistic: -89.5976974180104
T-Test - P-Value: 0.0 (adjusted for two-tailed test)
```



In [ ]: