In [ ]:
```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.
mount("/content/drive", force_remount=True).
```

In [ ]:
```
!ls /content/drive/MyDrive
```

```
'Colab Notebooks'   data_Set   files
```

In [ ]:
```
import pandas as pd

# Replace with the correct path based on where you uploaded it
file_path = '/content/drive/MyDrive/data_Set/qcew-2020-2022.csv'
df = pd.read_csv(file_path)

df.head()
```

Out[ ]:

	Area Type	Area Name	Year	Time Period	Ownership	NAICS Level	NAICS Code	Industry Name	Establishments
0	County	Alameda County	2020	1st Qtr	Federal Government	2	1024	Professional and Business Services	2
1	County	Alameda County	2020	1st Qtr	Federal Government	3	491	Postal Service	28
2	County	Alameda County	2020	1st Qtr	Federal Government	3	541	Professional and Technical Services	2
3	County	Alameda County	2020	1st Qtr	Federal Government	5	54133	Engineering Services	1
4	County	Alameda County	2020	1st Qtr	Federal Government	2	62	Health Care and Social Assistance	1

In [ ]:
```
# Initial shape and duplicate removal
initial_shape = df.shape
duplicates = df.duplicated().sum()
df = df.drop_duplicates()
final_shape = df.shape

# Missing value summary
missing = df.isnull().sum()
missing = missing[missing > 0]
```

```
# Print cleaning summary
print("Initial shape:", initial_shape)
print("Duplicates removed:", duplicates)
print("Final shape after cleaning:", final_shape)
print("\nColumns with missing values:\n", missing)
```

```
Initial shape: (798464, 15)
Duplicates removed: 0
Final shape after cleaning: (798464, 15)

Columns with missing values:
 Series([], dtype: int64)
```

In [ ]:
```
print("To know the basic statistics of dataset")
df.describe()
```

```
To know the basic statistics of dataset
```

Out[ ]:

	Year	NAICS Level	Establishments	Average Monthly Employment	1st Month Emp	2nd M
count	798464.000000	798464.000000	7.984640e+05	7.984640e+05	7.984640e+05	7.98464(
mean	2021.043745	4.738440	2.594658e+03	3.304981e+04	2.624615e+04	2.647355
std	0.823588	1.273008	9.615226e+04	1.176228e+06	1.045006e+06	1.053440
min	2020.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000
25%	2020.000000	4.000000	6.000000e+00	6.000000e+01	1.000000e+01	1.000000
50%	2021.000000	5.000000	1.900000e+01	2.640000e+02	1.280000e+02	1.290000
75%	2022.000000	6.000000	8.700000e+01	1.411000e+03	8.450000e+02	8.520000
max	2022.000000	6.000000	1.175491e+07	1.525102e+08	1.522428e+08	1.527627

In [ ]:
```
print("To know the information of columns")
df.info()
```

```
To know the information of columns
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 798464 entries, 0 to 798463
Data columns (total 15 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   Area Type                    798464 non-null  object
 1   Area Name                    798464 non-null  object
 2   Year                         798464 non-null  int64
 3   Time Period                  798464 non-null  object
 4   Ownership                    798464 non-null  object
 5   NAICS Level                  798464 non-null  int64
 6   NAICS Code                   798464 non-null  object
 7   Industry Name                798464 non-null  object
 8   Establishments               798464 non-null  int64
 9   Average Monthly Employment   798464 non-null  int64
 10  1st Month Emp                798464 non-null  int64
 11  2nd Month Emp                798464 non-null  int64
 12  3rd Month Emp                798464 non-null  int64
 13  Total Wages (All Workers)    798464 non-null  float64
 14  Average Weekly Wages         798464 non-null  int64
dtypes: float64(1), int64(8), object(6)
memory usage: 91.4+ MB
```

In [ ]:
```
print("The list of all columns present in the dataset are: ")
print("-----------------------------------------------------------")
df.columns
```

```
The list of all columns present in the dataset are:
-----------------------------------------------------------
```

Out[ ]:
```
Index(['Area Type', 'Area Name', 'Year', 'Time Period', 'Ownership',
       'NAICS Level', 'NAICS Code', 'Industry Name', 'Establishments',
       'Average Monthly Employment', '1st Month Emp', '2nd Month Emp',
       '3rd Month Emp', 'Total Wages (All Workers)', 'Average Weekly Wages'],
      dtype='object')
```

In [ ]:
```
print("The unique datatypes and their count in the dataset are: ")
df.dtypes.value_counts()
```

```
The unique datatypes and their count in the dataset are:
```

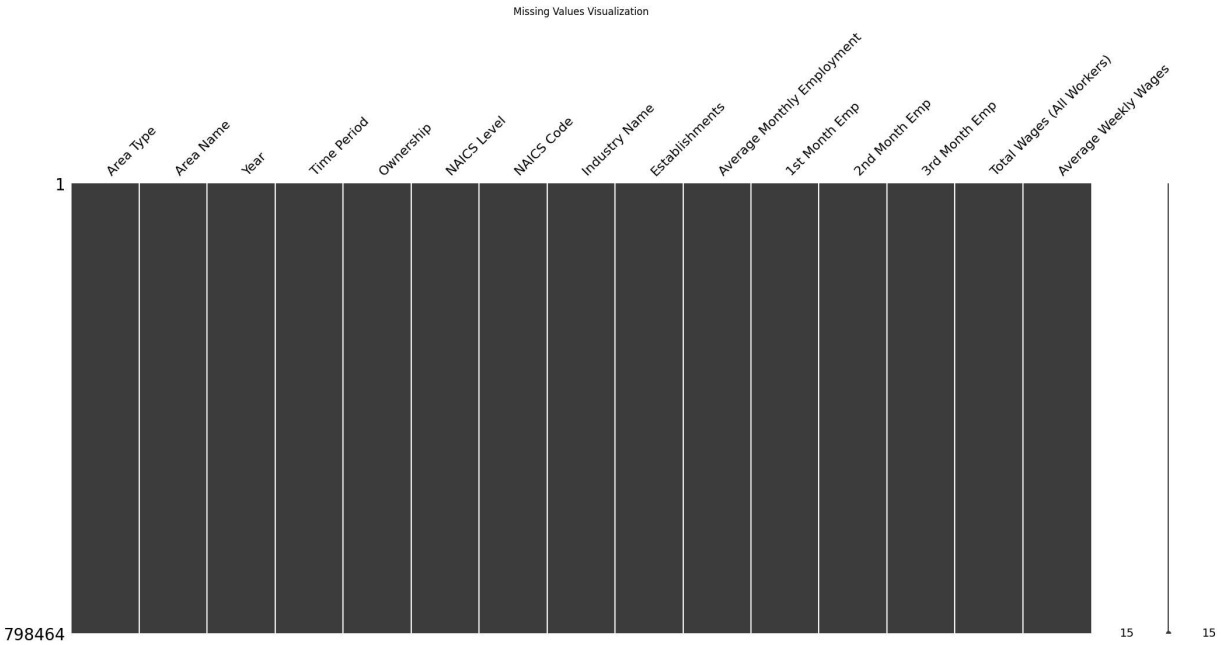
Out[ ]:

	count
int64	8
object	6
float64	1

dtype: int64

In [ ]:
```
import missingno as msno
import matplotlib.pyplot as plt

# Check missing values
msno.matrix(df)
```

```
plt.title('Missing Values Visualization', pad=20)
plt.show()
```



In [ ]:
```
print("To know the last 5 tupples from the dataset ")
df.tail()
```

```
To know the last 5 tupples from the dataset
```

Out[ ]:

	Area Type	Area Name	Year	Time Period	Ownership	NAICS Level	NAICS Code	Industry Name	Establishme
798459	United States	United States	2022	Annual	Private	5	61171	Educational Support Services	27(
798460	United States	United States	2022	Annual	Private	5	62441	Child Day Care Services	76(
798461	United States	United States	2022	Annual	Private	5	71113	Musical Groups and Artists	5(
798462	United States	United States	2022	Annual	Private	5	71132	Promoters without Facilities	5
798463	United States	United States	2022	Annual	Private	6	712190	Nature Parks & Other Similar Institution	

In [ ]:
```
print("The top 20 rows in the dataset are ")
df.head(20)
```

```
The top 20 rows in the dataset are
```

Out[ ]:

	Area Type	Area Name	Year	Time Period	Ownership	NAICS Level	NAICS Code	Industry Name	Establishm
0	County	Alameda County	2020	1st Qtr	Federal Government	2	1024	Professional and Business Services	
1	County	Alameda County	2020	1st Qtr	Federal Government	3	491	Postal Service	
2	County	Alameda County	2020	1st Qtr	Federal Government	3	541	Professional and Technical Services	
3	County	Alameda County	2020	1st Qtr	Federal Government	5	54133	Engineering Services	
4	County	Alameda County	2020	1st Qtr	Federal Government	2	62	Health Care and Social Assistance	
5	County	Alameda County	2020	1st Qtr	Federal Government	4	6211	Offices of Physicians	
6	County	Alameda County	2020	1st Qtr	Federal Government	1	102	Service-Providing	
7	County	Alameda County	2020	1st Qtr	Federal Government	2	1025	Education and Health Services	
8	County	Alameda County	2020	1st Qtr	Federal Government	2	1023	Financial Activities	
9	County	Alameda County	2020	1st Qtr	Federal Government	2	1028	Public Administration	
10	County	Alameda County	2020	1st Qtr	Federal Government	6	522298	All Other Nondeposit Credit Intermediatn	
11	County	Alameda County	2020	1st Qtr	Federal Government	2	54	Professional and Technical Services	
12	County	Alameda County	2020	1st Qtr	Federal Government	6	541330	Engineering Services	
13	County	Alameda County	2020	1st Qtr	Federal Government	4	5416	Management & Technical Consulting Svc	
14	County	Alameda County	2020	1st Qtr	Federal Government	6	541620	Environmental Consulting Services	

	Area Type	Area Name	Year	Time Period	Ownership	NAICS Level	NAICS Code	Industry Name	Establishm
15	County	Alameda County	2020	1st Qtr	Federal Government	3	621	Ambulatory Health Care Services	
16	County	Alameda County	2020	1st Qtr	Federal Government	6	621111	Offices of Physicians, ex. Mental Health	
17	County	Alameda County	2020	1st Qtr	Federal Government	4	7121	Museums, Parks and Historical Sites	
18	County	Alameda County	2020	1st Qtr	Federal Government	5	71219	Nature Parks & Other Similar Institution	
19	County	Alameda County	2020	1st Qtr	Federal Government	4	7139	Other Amusement & Recreation Industries	

In [ ]:
```
print("Missing Values:\n", df.isnull().sum())
```

```
Missing Values:
 Area Type                    0
Area Name                     0
Year                          0
Time Period                   0
Ownership                     0
NAICS Level                   0
NAICS Code                    0
Industry Name                 0
Establishments                0
Average Monthly Employment    0
1st Month Emp                 0
2nd Month Emp                 0
3rd Month Emp                 0
Total Wages (All Workers)     0
Average Weekly Wages          0
dtype: int64
```

cleaning steps

In [ ]:
```
# Filter Ownership to Primary Types ——— #
valid_ownerships = ['Private', 'Federal Government', 'State Government', 'Local Gov
df = df[df['Ownership'].isin(valid_ownerships)]
```

In [ ]:
```
#Remove or Handle 'Annual' Time Period Separately ——— #
df = df[df['Time Period'].isin(['1st Qtr', '2nd Qtr', '3rd Qtr', '4th Qtr'])]
```

In [ ]:
```
#Standardize and Filter Area Name and Area Type ——— #
# Filter only California counties
df = df[(df['Area Type'] == 'County') & (df['Area Name'] != 'California') & (df['Ar
```

In [ ]:
```
#Create Avg Employment per Establishment ——— #
df['Avg_Emp_Per_Est'] = df['Average Monthly Employment'] / df['Establishments']
df['Avg_Emp_Per_Est'] = df['Avg_Emp_Per_Est'].replace([float('inf'), -float('inf')]
```

In [ ]:
```
#Optional - Remove numeric outliers using IQR on Employment ——— #
def remove_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    return data[(data[column] >= Q1 - 1.5 * IQR) & (data[column] <= Q3 + 1.5 * IQR)
```

In [ ]:
```
df = remove_outliers_iqr(df, 'Avg_Emp_Per_Est')
df = remove_outliers_iqr(df, 'Average Weekly Wages')
```

In [ ]:
```
# Unique values in important categorical columns
print("Unique Area Types:", df['Area Type'].unique())
print("Unique Ownerships:", df['Ownership'].unique())
print("Unique Years:", df['Year'].unique())
print("Unique Time Periods:", df['Time Period'].unique())
```

```
Unique Area Types: ['County']
Unique Ownerships: ['Federal Government' 'State Government' 'Local Government' 'Priv
ate']
Unique Years: [2020 2021 2022]
Unique Time Periods: ['1st Qtr' '2nd Qtr' '3rd Qtr' '4th Qtr']
```

In [ ]:
```
# Wage Growth Rate: percentage change in wages per group
df['Wage Growth Rate'] = df.groupby(['Industry Name'])['Average Weekly Wages'].pct_

# Year-Quarter combined (for time series plots)
df['Year_Quarter'] = df['Year'].astype(str) + ' Q' + df['Time Period'].astype(str)
```

In [ ]:
```
# Average wages per industry
industry_avg_wages = df.groupby('Industry Name')['Average Weekly Wages'].mean().sor
print(industry_avg_wages.head())
```

```
Industry Name
Guided Missiles and Space Vehicles       2514.333333
Space Research and Technology            2483.000000
Automobile Manufacturing                 2372.166667
Fossil Fuel Electric Power Generation    2309.444444
Oil and Gas Extraction                   2227.500000
Name: Average Weekly Wages, dtype: float64
```

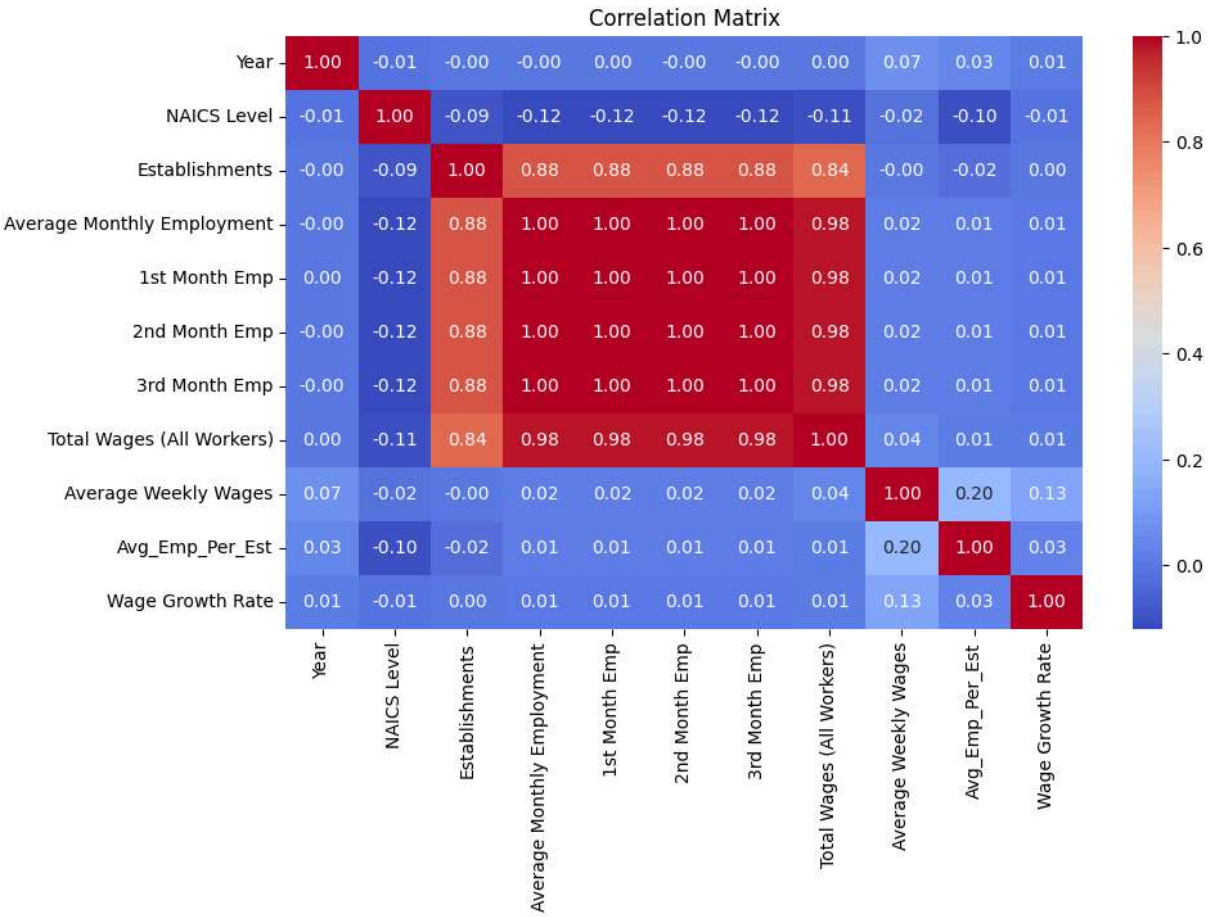
In [ ]:
```
# Total wages per year
yearly_wages = df.groupby('Year')['Total Wages (All Workers)'].sum()
print(yearly_wages)
```

```
Year
2020    5.275076e+12
2021    5.630752e+12
2022    6.114267e+12
Name: Total Wages (All Workers), dtype: float64
```

In [ ]:
```
# Employment by Ownership Type
ownership_employment = df.groupby('Ownership')['Average Monthly Employment'].sum()
print(ownership_employment)
```

```
Ownership
Federal Government       2444860
Local Government         8562952
Private               1048384928
State Government        21991981
Name: Average Monthly Employment, dtype: int64
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Correlation matrix
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



In [ ]:
```
# Encode relevant categorical columns
le = LabelEncoder()
```

```
df['Ownership_Code'] = le.fit_transform(df['Ownership'])
df['Industry_Code'] = le.fit_transform(df['Industry Name'])
df['Area_Code'] = le.fit_transform(df['Area Name'])
```

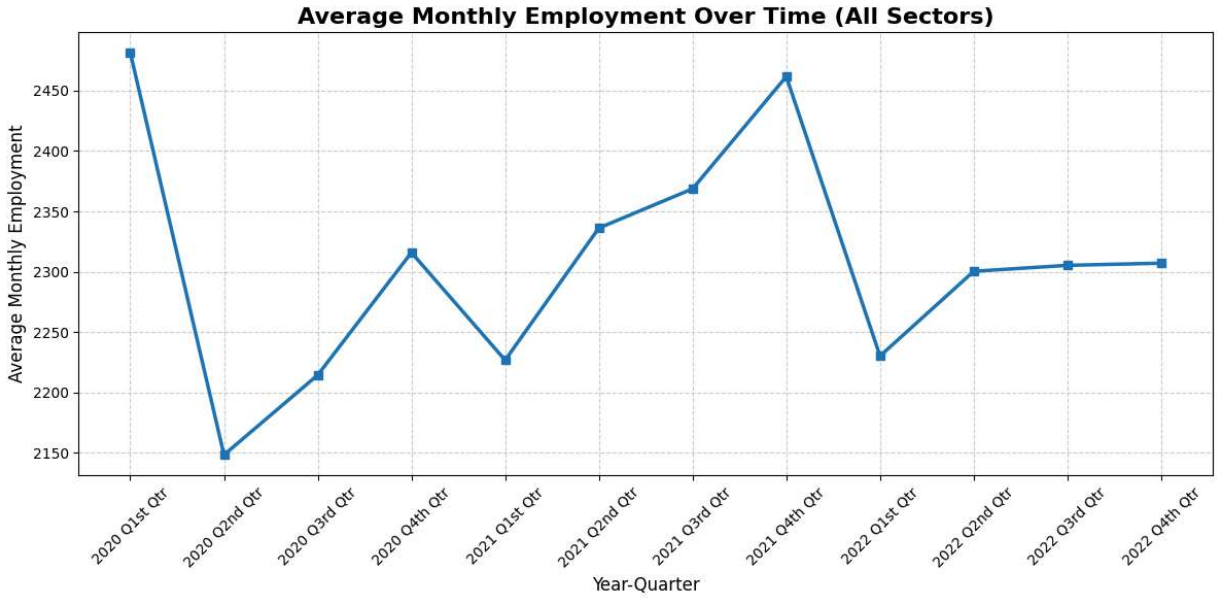
In [ ]:
```
# 1. Pearson Correlation Matrix
pearson_corr = df[['Average Weekly Wages', 'Average Monthly Employment', 'Ownership
print("Pearson Correlation Matrix:\n", pearson_corr)
```

```
Pearson Correlation Matrix:
                          Average Weekly Wages  Average Monthly Employment  \
Average Weekly Wages                  1.000000                    0.015397
Average Monthly Employment            0.015397                    1.000000
Ownership_Code                       -0.049444                    0.019623
Industry_Code                         0.009836                    0.022981

                          Ownership_Code  Industry_Code
Average Weekly Wages           -0.049444       0.009836
Average Monthly Employment      0.019623       0.022981
Ownership_Code                  1.000000       0.037781
Industry_Code                   0.037781       1.000000
```

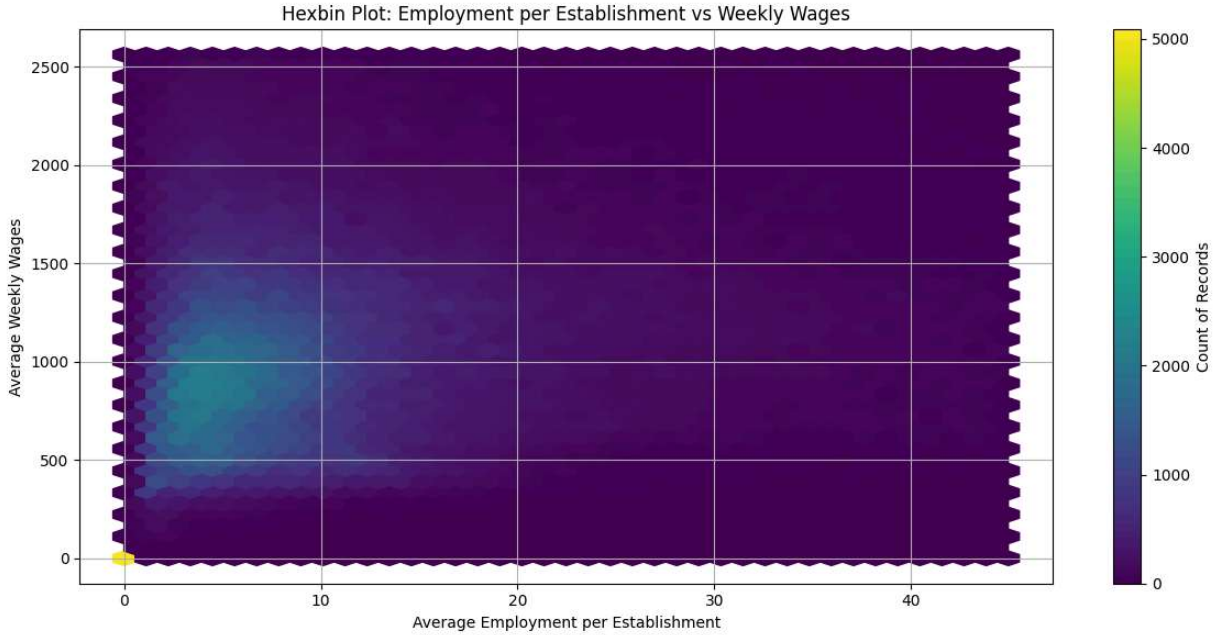
In [ ]:
```
# 2. Chi-Square Test between Ownership and Industry
from scipy import stats
from sklearn.preprocessing import LabelEncoder, StandardScaler
contingency = pd.crosstab(df['Ownership'], df['Industry Name'])
chi2, p, dof, expected = stats.chi2_contingency(contingency)
print(f"Chi-square test result: Chi2 = {chi2:.2f}, p-value = {p:.4f}")
```

```
Chi-square test result: Chi2 = 500265.71, p-value = 0.0000
```

In [ ]:
```
# Group and compute average employment over time
employment_trend = df.groupby('Year_Quarter')['Average Monthly Employment'].mean()

# Plot with improved styling
plt.figure(figsize=(12, 6))
plt.plot(employment_trend.index, employment_trend.values, marker='s', linestyle='-'

# Enhancements
plt.title('Average Monthly Employment Over Time (All Sectors)', fontsize=16, fontwe
plt.xlabel('Year-Quarter', fontsize=12)
plt.ylabel('Average Monthly Employment', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=10)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()

# Optional: Add data labels for key points
for x, y in employment_trend.items():
    if x.endswith('Q1') or x.endswith('Q4'):  # example condition for key quarters
        plt.text(x, y, f'{y:,.0f}', ha='center', va='bottom', fontsize=9, rotation=

plt.show()
```

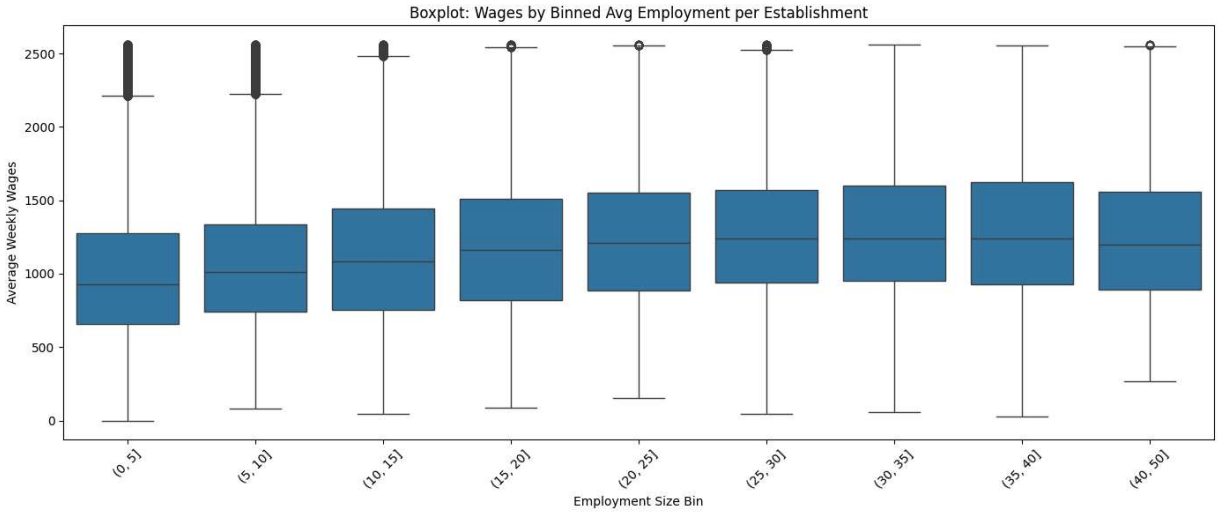


In [ ]:
```
plt.figure(figsize=(12, 6))
plt.hexbin(df['Avg_Emp_Per_Est'], df['Average Weekly Wages'], gridsize=40, cmap='vi
plt.colorbar(label='Count of Records')
plt.title('Hexbin Plot: Employment per Establishment vs Weekly Wages')
plt.xlabel('Average Employment per Establishment')
plt.ylabel('Average Weekly Wages')
plt.grid(True)
plt.tight_layout()
plt.show()
```

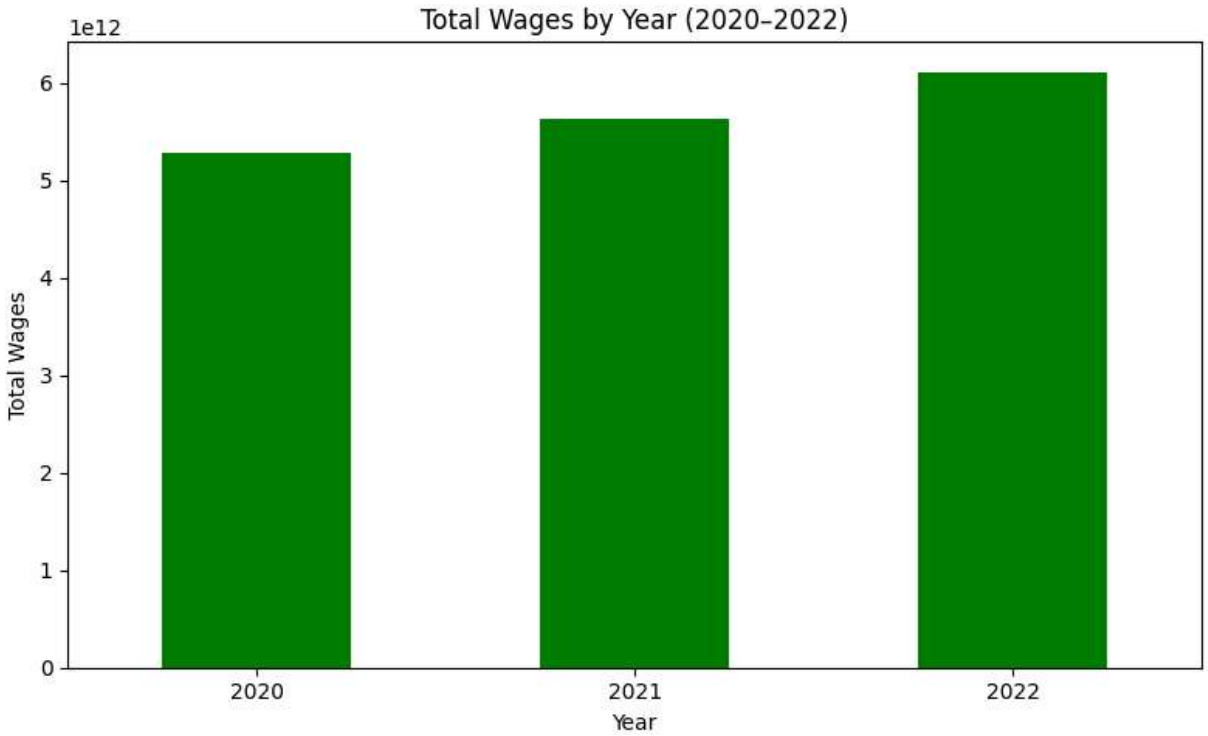


In [ ]:
```
# Create bins of Avg_Emp_Per_Est
df['Emp_Bin'] = pd.cut(df['Avg_Emp_Per_Est'], bins=[0, 5, 10, 15, 20, 25, 30, 35, 4

plt.figure(figsize=(14, 6))
sns.boxplot(x='Emp_Bin', y='Average Weekly Wages', data=df)
plt.title('Boxplot: Wages by Binned Avg Employment per Establishment')
plt.xlabel('Employment Size Bin')
```

```
plt.ylabel('Average Weekly Wages')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

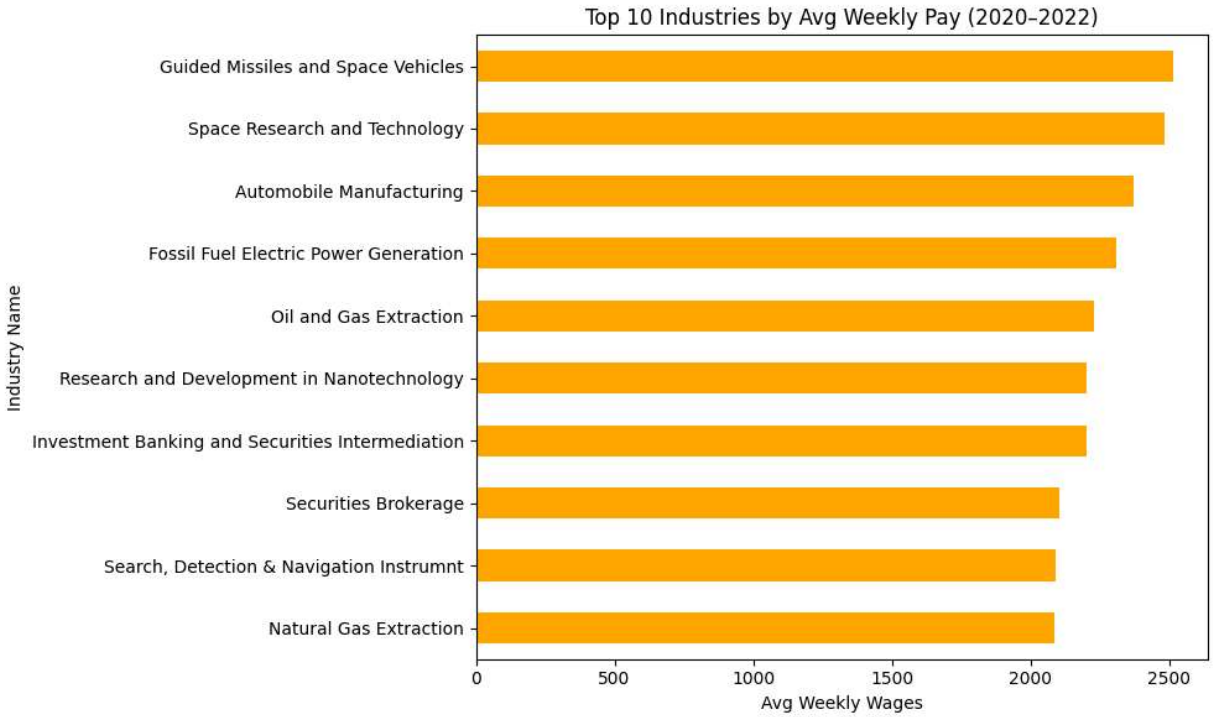


In [ ]:
```
df.groupby('Year')['Total Wages (All Workers)'].sum().plot(kind='bar', figsize=(8,
plt.title('Total Wages by Year (2020-2022)')
plt.ylabel('Total Wages')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

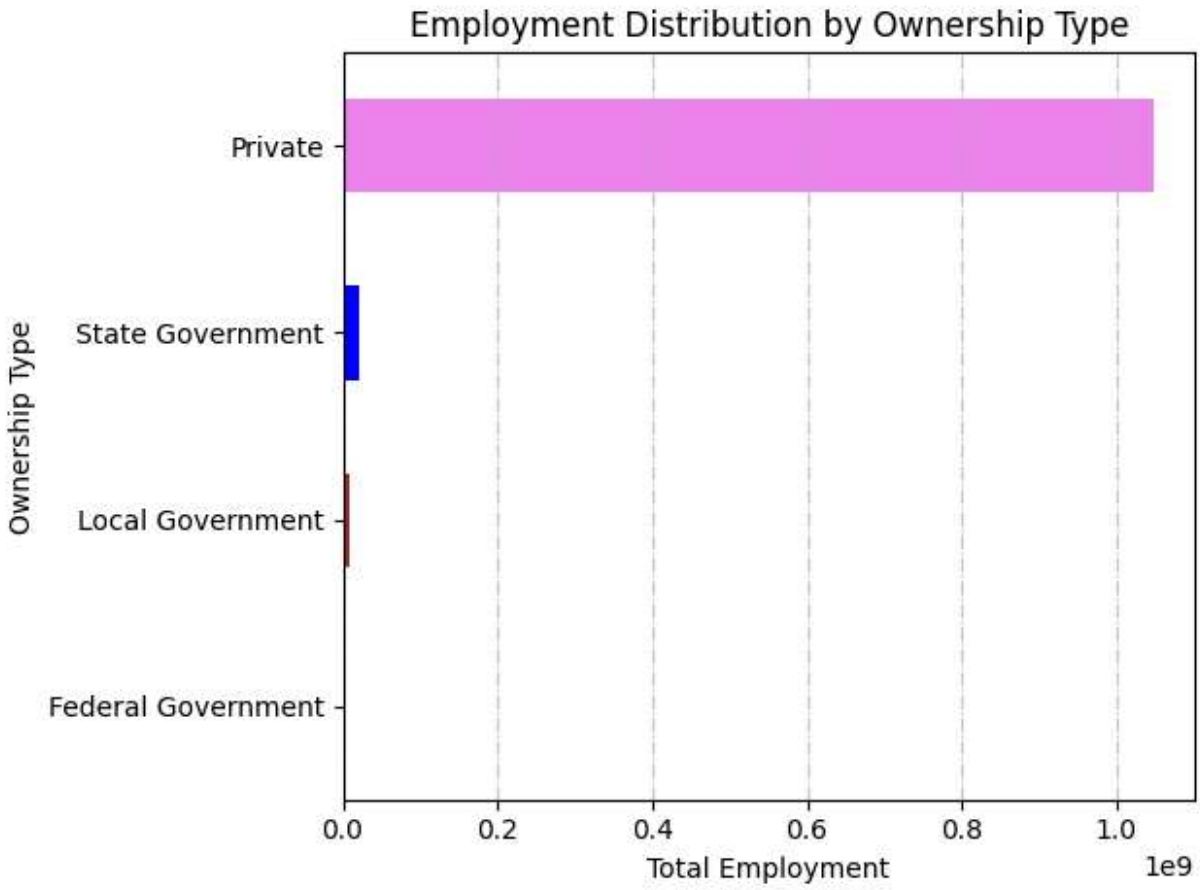


In [ ]:
```
top10_wages = industry_avg_wages.head(10)

top10_wages.plot(kind='barh', figsize=(10, 6), color='orange')
plt.xlabel('Avg Weekly Wages')
plt.title('Top 10 Industries by Avg Weekly Pay (2020-2022)')
```

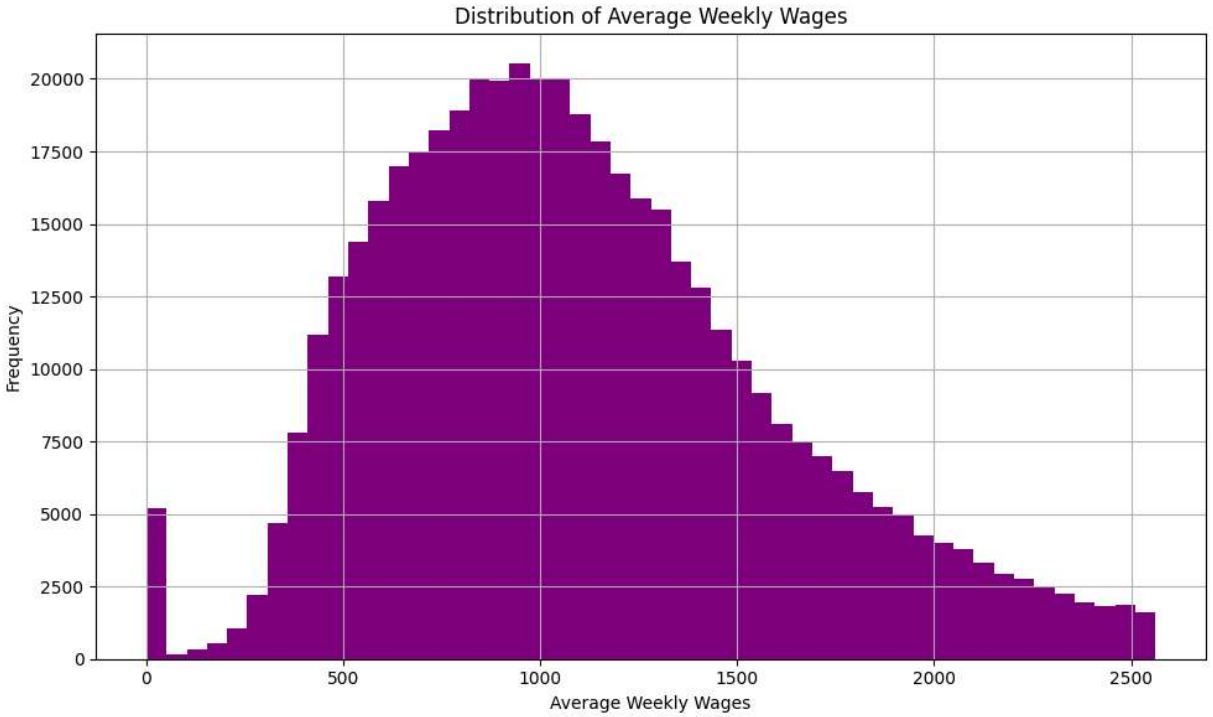
```
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



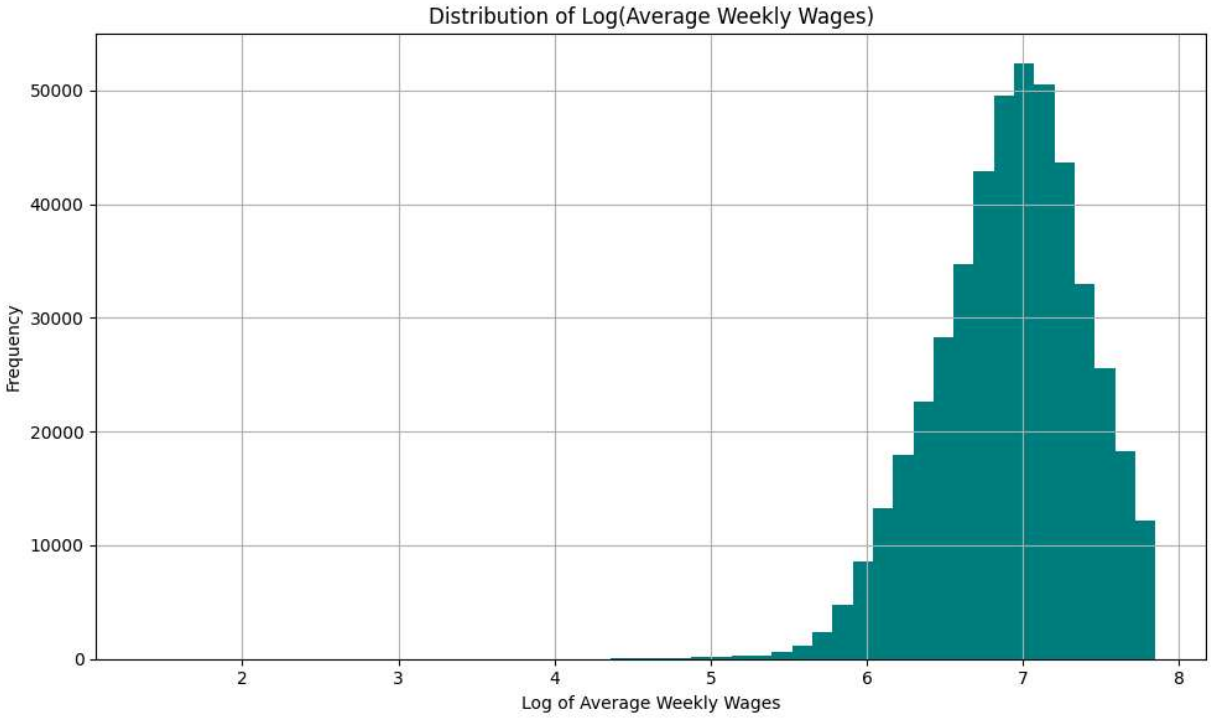
In [ ]:
```
ownership_employment = df.groupby('Ownership')['Average Monthly Employment'].sum()
ownership_employment = ownership_employment.sort_values(ascending=True)

ownership_employment.plot(kind='barh', color=['#228B22', 'brown', 'blue', 'violet']
plt.title('Employment Distribution by Ownership Type')
plt.xlabel('Total Employment')
plt.ylabel('Ownership Type')
plt.grid(axis='x', linestyle='-.', alpha=0.7)
plt.tight_layout()
plt.show()
```



In [ ]:
```
df['Average Weekly Wages'].plot(kind='hist', bins=50, figsize=(10, 6), color='purpl
plt.title('Distribution of Average Weekly Wages')
plt.xlabel('Average Weekly Wages')
plt.grid(True)
plt.tight_layout()
plt.show()
```

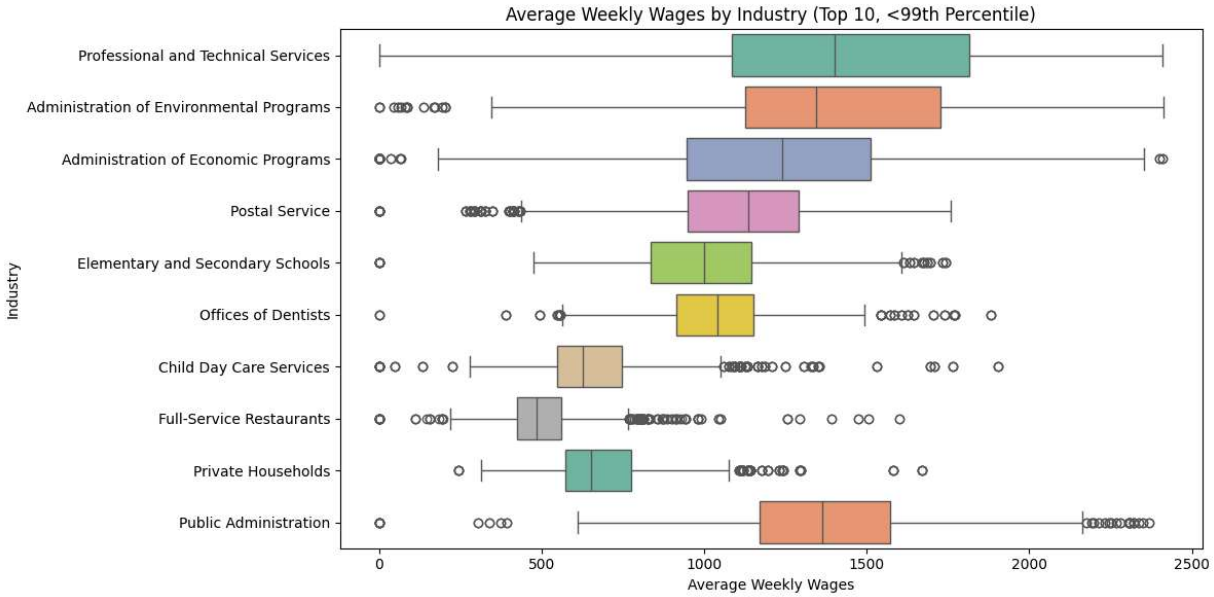


In [ ]:
```python
import numpy as np

# Use log scale on wages (after removing 0s or negatives)
df_log = df[df['Average Weekly Wages'] > 0].copy()
df_log['log_wages'] = np.log(df_log['Average Weekly Wages'])

df_log['log_wages'].plot(kind='hist', bins=50, figsize=(10, 6), color='teal')
plt.title('Distribution of Log(Average Weekly Wages)')
plt.xlabel('Log of Average Weekly Wages')
plt.grid(True)
plt.tight_layout()
plt.show()
```

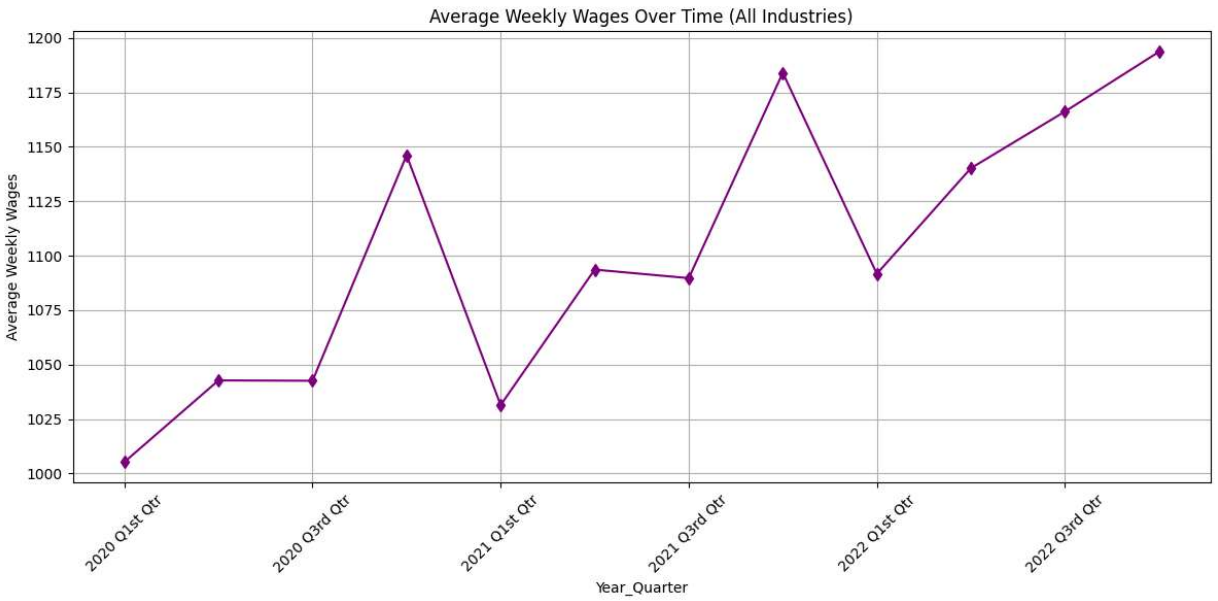


In [ ]:
```
# Filter to keep only data below the 99th percentile
wage_99 = df['Average Weekly Wages'].quantile(0.99)
filtered_df = df[df['Average Weekly Wages'] <= wage_99]

# Optional: Filter to top 10 industries by number of records
top_industries = df['Industry Name'].value_counts().nlargest(10).index
filtered_top = df[df['Industry Name'].isin(top_industries) & (df['Average Weekly Wa

# Box plot
plt.figure(figsize=(12, 6))
sns.boxplot(data=filtered_top, x='Average Weekly Wages', y='Industry Name', palette
plt.title('Average Weekly Wages by Industry (Top 10, <99th Percentile)')
plt.xlabel('Average Weekly Wages')
plt.ylabel('Industry')
plt.tight_layout()
plt.show()
```

```
<ipython-input-34-f6b7e9fc5755>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data=filtered_top, x='Average Weekly Wages', y='Industry Name', palett
e='Set2')
```

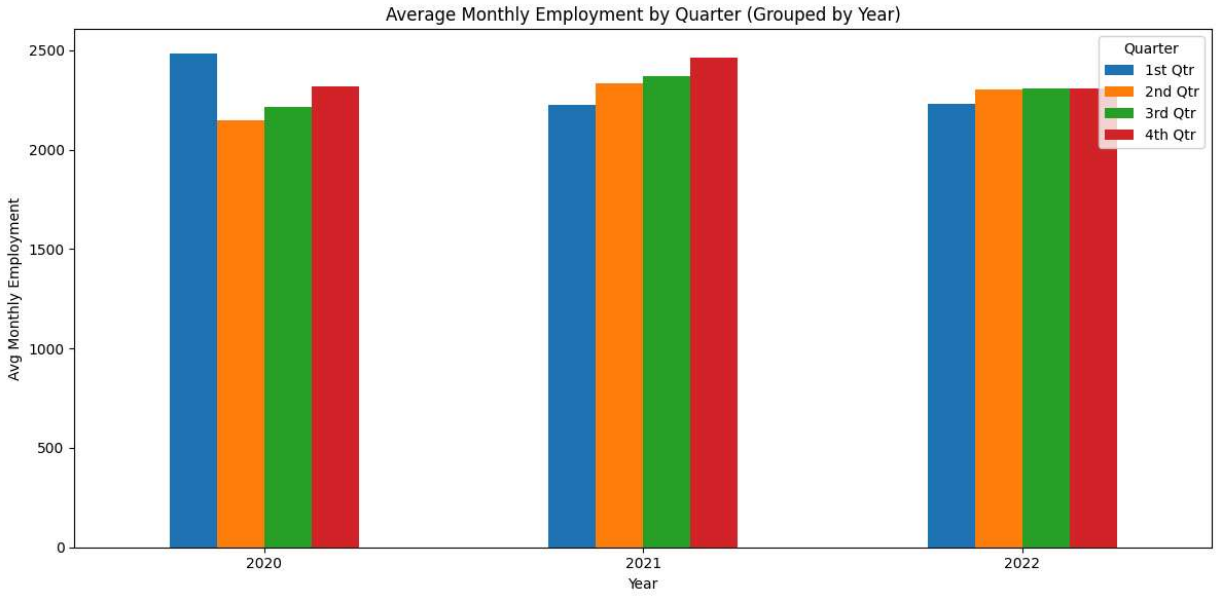


In [ ]:
```
# First, ensure the data is sorted
df.sort_values(by=['Year', 'Time Period'], inplace=True)

# Group: Mean wages over time
wage_trend = df.groupby('Year_Quarter')['Average Weekly Wages'].mean()

# Plot
wage_trend.plot(kind='line', marker='d', figsize=(12, 6), color='purple')
plt.title('Average Weekly Wages Over Time (All Industries)')
plt.ylabel('Average Weekly Wages')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

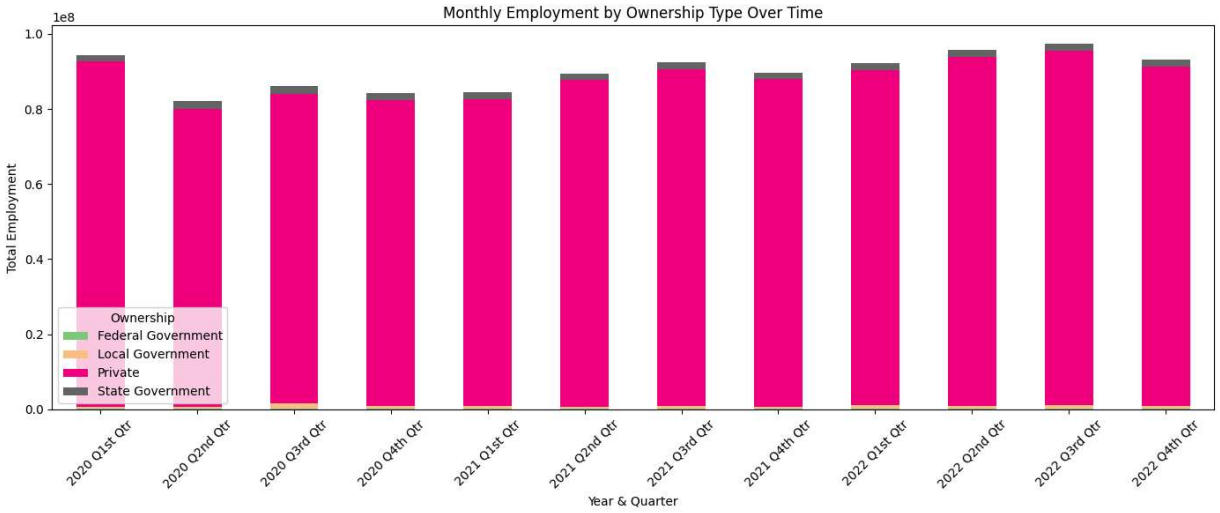


In [ ]:
```
#Grouped Bar Chart – Average Monthly Employment by Quarter per Year

# Group by Year and Quarter to get average employment
```

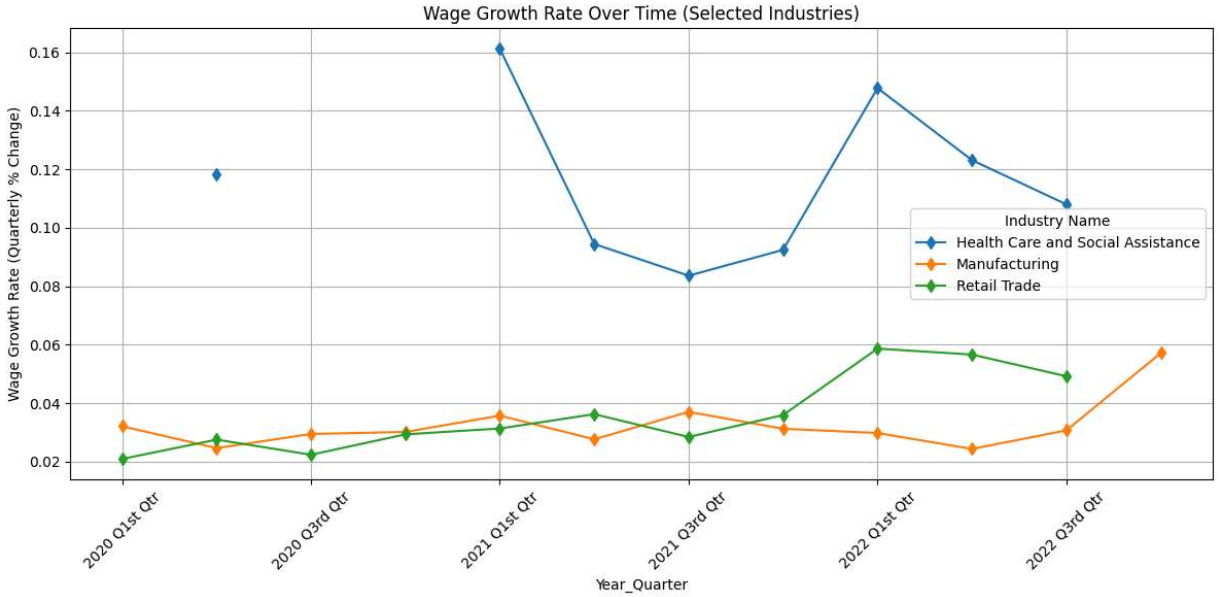
```
employment_by_quarter = df.groupby(['Year', 'Time Period'])['Average Monthly Employ

# Plot grouped bars
employment_by_quarter.plot(kind='bar', figsize=(12, 6))
plt.title('Average Monthly Employment by Quarter (Grouped by Year)')
plt.xlabel('Year')
plt.ylabel('Avg Monthly Employment')
plt.legend(title='Quarter')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```



In [ ]:
```
#Stacked Bar Chart – Employment by Ownership Over Time

# Group by time and ownership
ownership_time = df.groupby(['Year_Quarter', 'Ownership'])['Average Monthly Employm

# Plot stacked bars
ownership_time.plot(kind='bar', stacked=True, figsize=(14, 6), colormap='Accent')
plt.title('Monthly Employment by Ownership Type Over Time')
plt.ylabel('Total Employment')
plt.xlabel('Year & Quarter')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



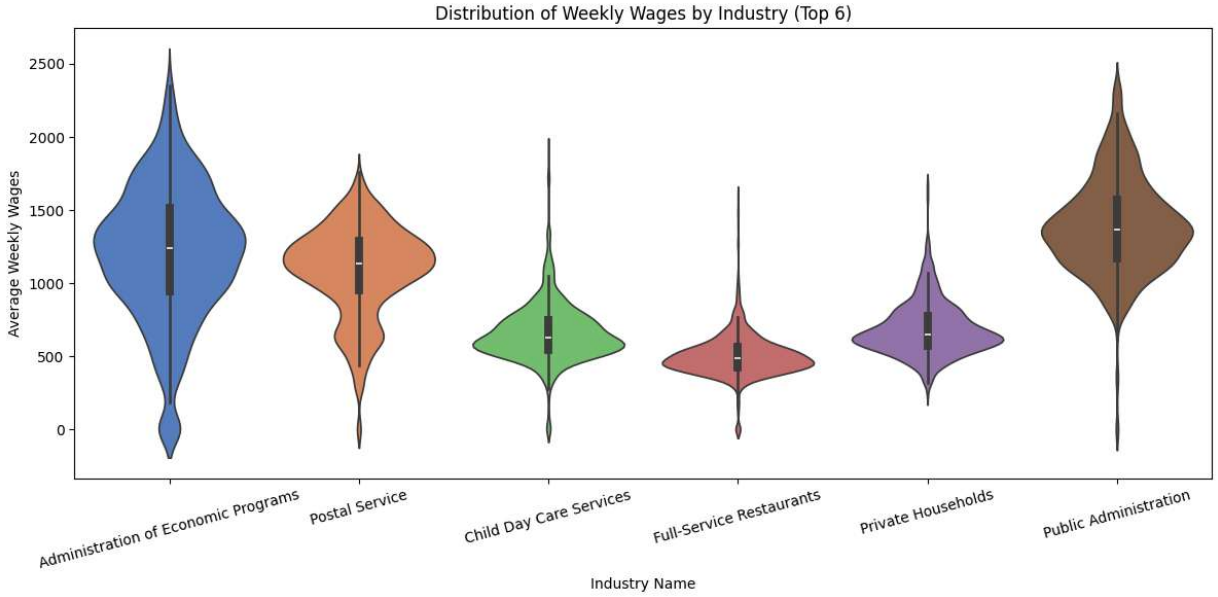
In [ ]:
```
#Line Plot – Wage Growth Rate Over Time (Selected Industries)

# Select a few major industries
selected_industries = ['Health Care and Social Assistance', 'Retail Trade', 'Manufa

# Filter and pivot for plotting
subset = df[df['Industry Name'].isin(selected_industries)]
pivot_growth = subset.pivot_table(index='Year_Quarter', columns='Industry Name', va

# Plot
pivot_growth.plot(figsize=(12, 6), marker='d')
plt.title('Wage Growth Rate Over Time (Selected Industries)')
plt.ylabel('Wage Growth Rate (Quarterly % Change)')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



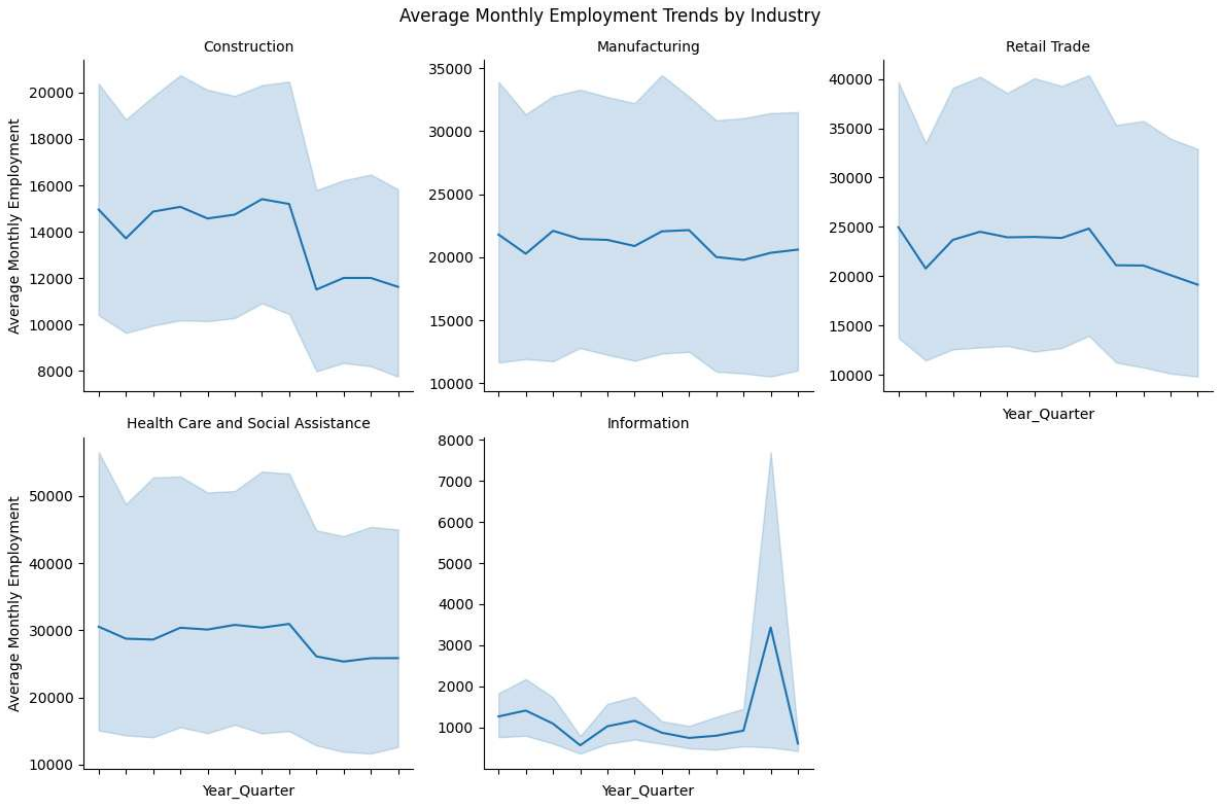
In [ ]:
```
#Violin Plot – Wages by Industry (Top 6)

# Top 6 industries by frequency
```

```
top6 = df['Industry Name'].value_counts().nlargest(6).index
df_top6 = df[df['Industry Name'].isin(top6) & (df['Average Weekly Wages'] <= wage_9

plt.figure(figsize=(12, 6))
sns.violinplot(data=df_top6, x='Industry Name', y='Average Weekly Wages', palette='
plt.title('Distribution of Weekly Wages by Industry (Top 6)')
plt.xticks(rotation=15)
plt.tight_layout()
plt.show()
```

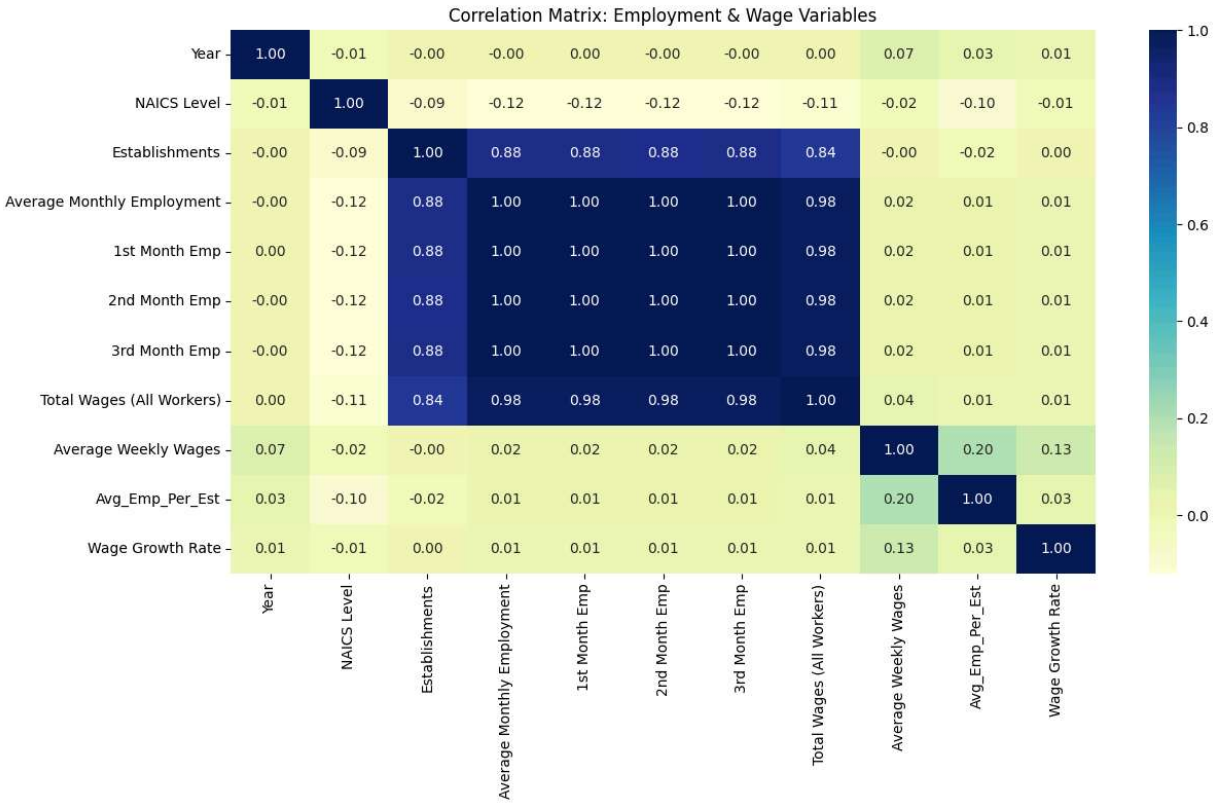
```
<ipython-input-39-f2ba94246785>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.violinplot(data=df_top6, x='Industry Name', y='Average Weekly Wages', palette
='muted')
```



In [ ]:
```
#Faceted Line Plot – Employment Trends by Industry

import seaborn as sns

# Pick a few industries
selected = df[df['Industry Name'].isin(['Retail Trade', 'Manufacturing', 'Construct
                                        'Health Care and Social Assistance', 'Infor

# Create a faceted line plot
g = sns.FacetGrid(selected, col="Industry Name", col_wrap=3, height=4, sharey=False
g.map_dataframe(sns.lineplot, x='Year_Quarter', y='Average Monthly Employment')
g.set_xticklabels(rotation=45)
g.set_titles("{col_name}")
g.fig.subplots_adjust(top=0.9)
g.fig.suptitle('Average Monthly Employment Trends by Industry')
plt.tight_layout()
plt.show()
```



In [ ]:
```
# Heatmap – Correlation Between All Numerical Variables

plt.figure(figsize=(13, 8))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='YlGnBu', fmt=".2f")
plt.title("Correlation Matrix: Employment & Wage Variables")
plt.tight_layout()
plt.show()
```



In [ ]:
```
# 3. T-test
private_wages = df[df['Ownership'] == 'Private']['Average Weekly Wages']
localgov_wages = df[df['Ownership'] == 'Local Government']['Average Weekly Wages']
t_stat, p_val = stats.ttest_ind(private_wages, localgov_wages, equal_var=False)
print(f"T-test: T-statistic = {t_stat:.2f}, p-value = {p_val:.4f}")
```

```
T-test: T-statistic = -9.08, p-value = 0.0000
```

In [ ]:
```
# 4. ANOVA (compare wages across industries)
industry_groups = [group['Average Weekly Wages'].dropna() for _, group in df.groupb
anova_result = stats.f_oneway(*industry_groups)
print(f"ANOVA: F-statistic = {anova_result.statistic:.2f}, p-value = {anova_result.
```

```
ANOVA: F-statistic = 285.22, p-value = 0.0000
```

In [ ]:
```
# Check unique values to pick a valid California county
print("Sample area names:", df['Area Name'].unique()[:5])
print("Sample time periods:", df['Time Period'].unique())

# Convert Year + Time Period to datetime
def convert_quarter_to_date(row):
    if pd.isna(row['Year']) or pd.isna(row['Time Period']):
        return np.nan
    quarter_str = str(row['Time Period'])[0]  # Extract '1' from '1st Qtr'
    if not quarter_str.isdigit():
        return np.nan
    quarter = int(quarter_str)
    month = (quarter - 1) * 3 + 1
    return pd.Timestamp(year=int(row['Year']), month=month, day=1)

df['Date'] = df.apply(convert_quarter_to_date, axis=1)
```

```
# Check if Date column was created correctly
print("Date column sample:", df['Date'].dropna().unique()[:5])

# Select a county with enough data
county_df = df[df['Area Name'] == 'Los Angeles County, CA']
ts = county_df.groupby('Date')['Average Weekly Wages'].mean().dropna()

print("Number of data points in ts:", len(ts))  # Ensure ≥ 8 points

if len(ts) >= 8:
    ts = ts.asfreq('QS')
    from statsmodels.tsa.seasonal import seasonal_decompose
    decomp = seasonal_decompose(ts, model='additive', period=4)
    decomp.plot()
    plt.tight_layout()
    plt.show()
else:
    print("Not enough data points for seasonal decomposition. Try a different count
```

```
Sample area names: ['Alameda County' 'Alpine County' 'Amador County' 'Butte County'
 'Calaveras County']
Sample time periods: ['1st Qtr' '2nd Qtr' '3rd Qtr' '4th Qtr']
Date column sample: <DatetimeArray>
['2020-01-01 00:00:00', '2020-04-01 00:00:00', '2020-07-01 00:00:00',
 '2020-10-01 00:00:00', '2021-01-01 00:00:00']
Length: 5, dtype: datetime64[ns]
Number of data points in ts: 0
Not enough data points for seasonal decomposition. Try a different county.
```

Modeling

Step 1: Encode categorical variables

In [ ]:
```
df_model = df.copy()

# Drop nulls in target
df_model = df_model.dropna(subset=['Average Weekly Wages'])

# One-hot encode categorical variables
df_encoded = pd.get_dummies(df_model[['Year', 'Time Period', 'Industry Name', 'Owne

# Concatenate numerical columns
X = pd.concat([df_encoded, df_model[['Average Monthly Employment']]], axis=1)
y = df_model['Average Weekly Wages']
```

In [ ]:
```
# Keep only top 20 most frequent industries to limit dummy variables
top_industries = df['Industry Name'].value_counts().nlargest(20).index
df_small = df[df['Industry Name'].isin(top_industries)]
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

df_model = df_small.copy()

# Label encode high-cardinality variables
```

```
le_industry = LabelEncoder()
le_area = LabelEncoder()

df_model['Industry Encoded'] = le_industry.fit_transform(df_model['Industry Name'])
df_model['Area Encoded'] = le_area.fit_transform(df_model['Area Type'])

# Select features
features = ['Year', 'Time Period', 'Ownership', 'Industry Encoded', 'Area Encoded',
df_model = df_model.dropna(subset=['Average Weekly Wages'])

# One-hot encode remaining low-cardinality categorical vars
X_encoded = pd.get_dummies(df_model[['Year', 'Time Period', 'Ownership']], drop_fir

# Final input matrix
X = pd.concat([X_encoded, df_model[['Industry Encoded', 'Area Encoded', 'Average Mo
y = df_model['Average Weekly Wages']
```

In [ ]:

Step 2: Fit a simple regression model

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import numpy as np

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Fit
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predict
y_pred = lr.predict(X_test)

# Evaluate
print("R² Score:", r2_score(y_test, y_pred))
print("MAE     :", mean_absolute_error(y_test, y_pred))
# MSE and RMSE manually
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print("MSE     :", mse)
print("RMSE    :", rmse)
```

```
R² Score: 0.2024917557387803
MAE     : 300.2261356743714
MSE     : 157145.33709832875
RMSE    : 396.41561157241114
```

Random Forest Regression for Wage Prediction

Step 1: Setup & Import

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import numpy as np
```

Step 2: Prepare Data (Reusing your cleaned + encoded dataframe)

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

df_model = df.copy()

# Limit to top industries for memory efficiency
top_industries = df_model['Industry Name'].value_counts().nlargest(20).index
df_model = df_model[df_model['Industry Name'].isin(top_industries)]

# Encode string fields
le_industry = LabelEncoder()
le_area = LabelEncoder()
df_model['Industry Encoded'] = le_industry.fit_transform(df_model['Industry Name'])
df_model['Area Encoded'] = le_area.fit_transform(df_model['Area Type'])

# Drop rows with missing target
df_model = df_model.dropna(subset=['Average Weekly Wages'])

# Encode low-cardinality categorical features
X_encoded = pd.get_dummies(df_model[['Year', 'Time Period', 'Ownership']], drop_fir

# Final feature matrix
X = pd.concat([X_encoded, df_model[['Industry Encoded', 'Area Encoded', 'Average Mo
y = df_model['Average Weekly Wages']
```

```
<ipython-input-47-32cc0c6d3794>:12: 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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_model['Industry Encoded'] = le_industry.fit_transform(df_model['Industry Nam
e'])
<ipython-input-47-32cc0c6d3794>:13: 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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_model['Area Encoded'] = le_area.fit_transform(df_model['Area Type'])
```

Step 3: Train the Random Forest Model

In [ ]:
```
# Split into train/test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize and train model
```

```
rf = RandomForestRegressor(n_estimators=100, max_depth=15, random_state=42, n_jobs=
rf.fit(X_train, y_train)

# Predict
y_pred = rf.predict(X_test)
```

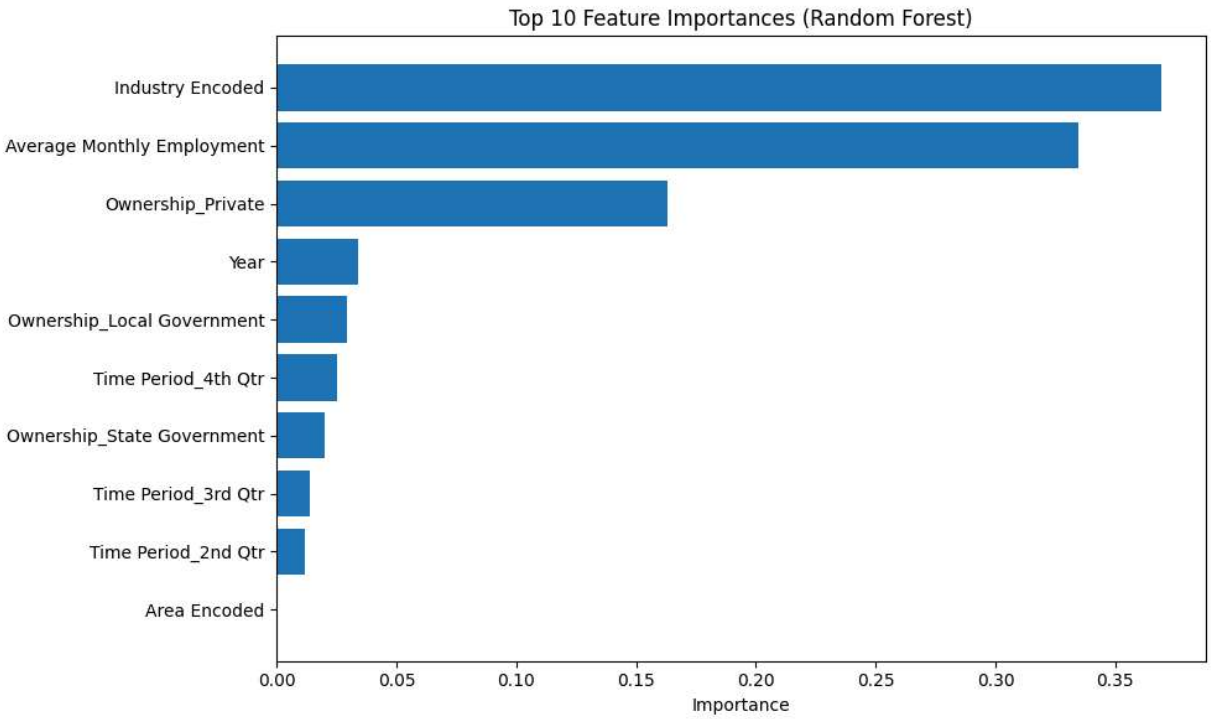
Step 4: Evaluate the Model

In [ ]:
```
# Evaluation
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print("📊 Random Forest Results:")
print(f"R² Score : {r2:.4f}")
print(f"MAE      : ${mae:.2f}")
print(f"RMSE     : ${rmse:.2f}")
```

```
📊 Random Forest Results:
R² Score : 0.8575
MAE      : $104.35
RMSE     : $167.58
```

Step 5: Feature Importance

In [ ]:
```
import matplotlib.pyplot as plt

# Get feature importances
importances = rf.feature_importances_
feature_names = X.columns

# Plot
indices = np.argsort(importances)[-10:]  # Top 10
plt.figure(figsize=(10, 6))
plt.barh(range(len(indices)), importances[indices], align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.title("Top 10 Feature Importances (Random Forest)")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```



XGBoost for Predicting Wages

In [ ]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages
(2.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (fro
m xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-pa
ckages (from xgboost) (2.21.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (fro
m xgboost) (1.14.1)
```

Step 1: Import Libraries

In [ ]:
```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
#train xgbooster

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize the XGBoost regressor
xgb_model = xgb.XGBRegressor(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=6,
    subsample=0.8,
```

```
    colsample_bytree=0.8,
    random_state=42,
    n_jobs=-1
)

# Train the model
xgb_model.fit(X_train, y_train)

# Predict
y_pred = xgb_model.predict(X_test)
```

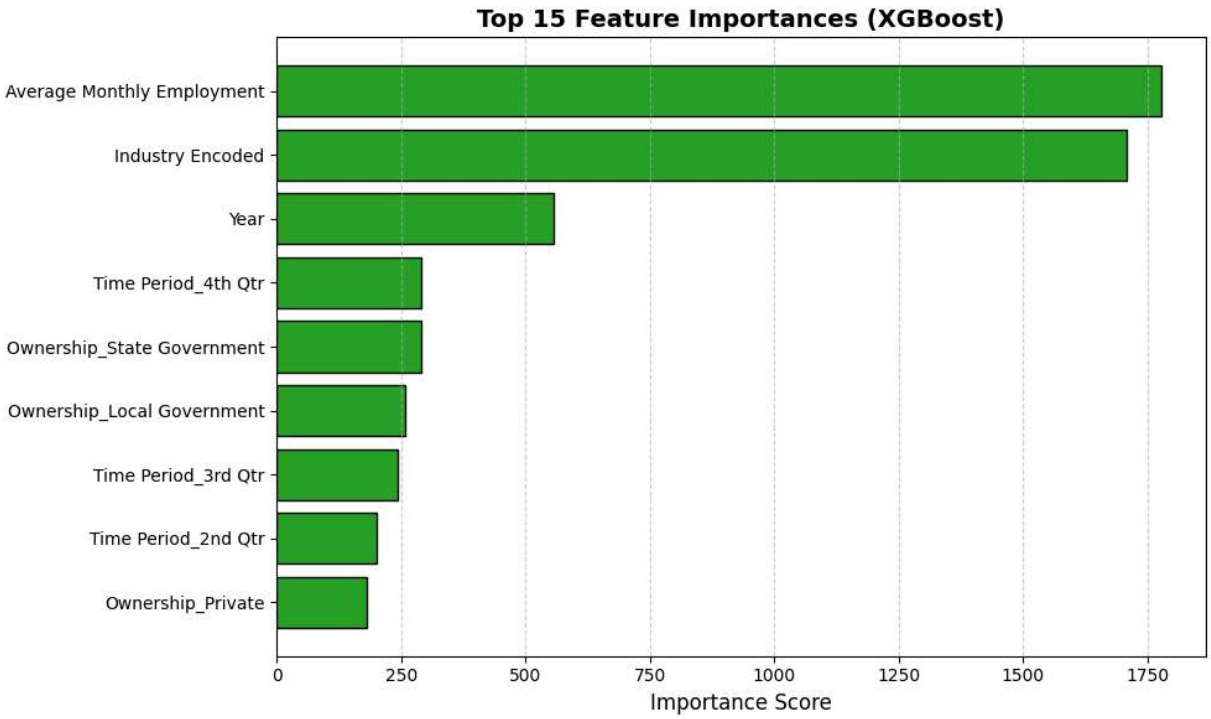
Evaluate the Model

In [ ]:
```
# Evaluate
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(" XGBoost Results:")
print(f"R² Score : {r2:.4f}")
print(f"MAE      : ${mae:.2f}")
print(f"RMSE     : ${rmse:.2f}")
```

```
 XGBoost Results:
R² Score : 0.7407
MAE      : $159.83
RMSE     : $226.05
```

Plot Feature Importances

In [ ]:
```
# Extract feature importances and sort them
importance_dict = xgb_model.get_booster().get_score(importance_type='weight')
importance_df = pd.DataFrame(list(importance_dict.items()), columns=['Feature', 'Im
importance_df = importance_df.sort_values(by='Importance', ascending=False).head(15

# Plotting as a horizontal bar chart
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='#2ca02c', ed
plt.xlabel('Importance Score', fontsize=12)
plt.title('Top 15 Feature Importances (XGBoost)', fontsize=14, fontweight='bold')
plt.gca().invert_yaxis()  # Highest importance on top
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



Hyperparameter Tuning with RandomizedSearchCV

In [ ]:
```
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
```

Define Parameter Grid

In [ ]:
```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 6, 8],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0]
}
```

Run RandomizedSearchCV

In [ ]:
```
xgb_reg = xgb.XGBRegressor(objective='reg:squarederror', random_state=42, n_jobs=-1

# Initialize random search
random_search = RandomizedSearchCV(
    estimator=xgb_reg,
    param_distributions=param_grid,
    n_iter=25,  # Try 25 random combinations
    cv=3,
    verbose=1,
    scoring='r2',
    random_state=42,
    n_jobs=-1
)
```

```
# Fit on training data only
random_search.fit(X_train, y_train)

# Best parameters
print(" Best Parameters:")
print(random_search.best_params_)
```

```
Fitting 3 folds for each of 25 candidates, totalling 75 fits
 Best Parameters:
{'subsample': 0.8, 'n_estimators': 300, 'max_depth': 8, 'learning_rate': 0.2, 'colsa
mple_bytree': 0.8}
```

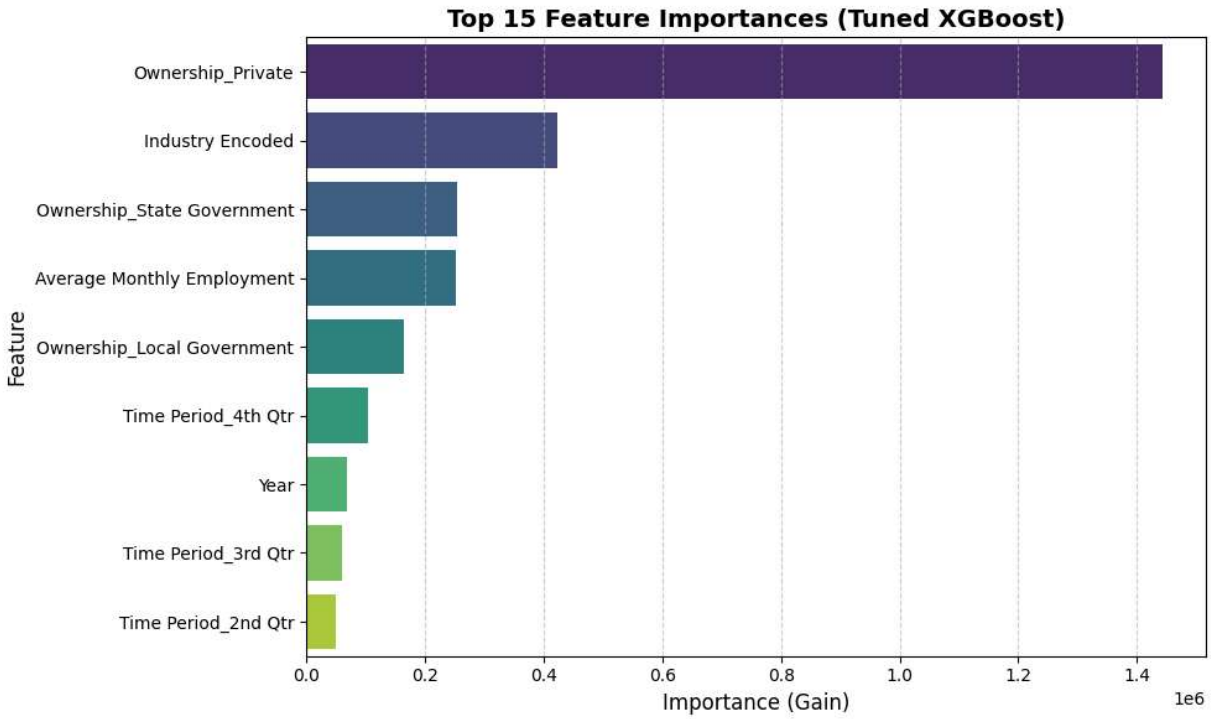
Evaluate Tuned Model on Test Set

In [ ]:
```
# Use the best model to predict
best_model = random_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(" Tuned XGBoost Results:")
print(f"R² Score : {r2:.4f}")
print(f"MAE      : ${mae:.2f}")
print(f"RMSE     : ${rmse:.2f}")
```

```
 Tuned XGBoost Results:
R² Score : 0.8355
MAE      : $118.21
RMSE     : $180.05
```

In [ ]:
```
# Extract and process feature importances
importance_dict = best_model.get_booster().get_score(importance_type='gain')  # you
importance_df = pd.DataFrame(list(importance_dict.items()), columns=['Feature', 'Im
importance_df = importance_df.sort_values(by='Importance', ascending=False).head(15

# Set plot style
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df, palette='viridis')

# Styling
plt.title('Top 15 Feature Importances (Tuned XGBoost)', fontsize=14, fontweight='bo
plt.xlabel('Importance (Gain)', fontsize=12)
plt.ylabel('Feature', fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
```

```
<ipython-input-61-17b4d8cd5380>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Importance', y='Feature', data=importance_df, palette='viridis')
```



Model Performance Comparison

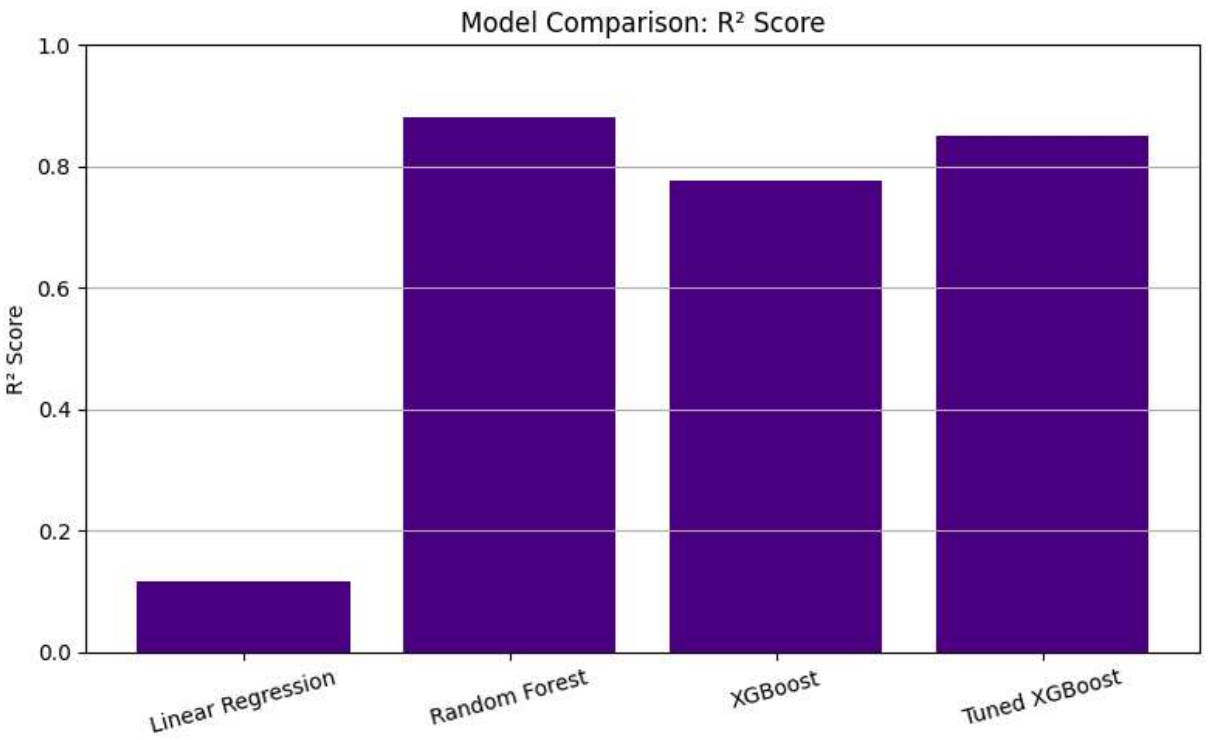
Enter Model Metrics

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

# Define model performance manually (from your outputs)
data = {
    'Model': [
        'Linear Regression',
        'Random Forest',
        'XGBoost',
        'Tuned XGBoost'
    ],
    'R2 Score': [0.1167, 0.8814, 0.7762, 0.8511],
    'MAE': [402.36, 133.60, 197.81, 154.20],
    'RMSE': [590.91, 216.53, 297.41, 242.62]
}

# Create DataFrame
df_compare = pd.DataFrame(data)
```

Plot R^2 Score Comparison

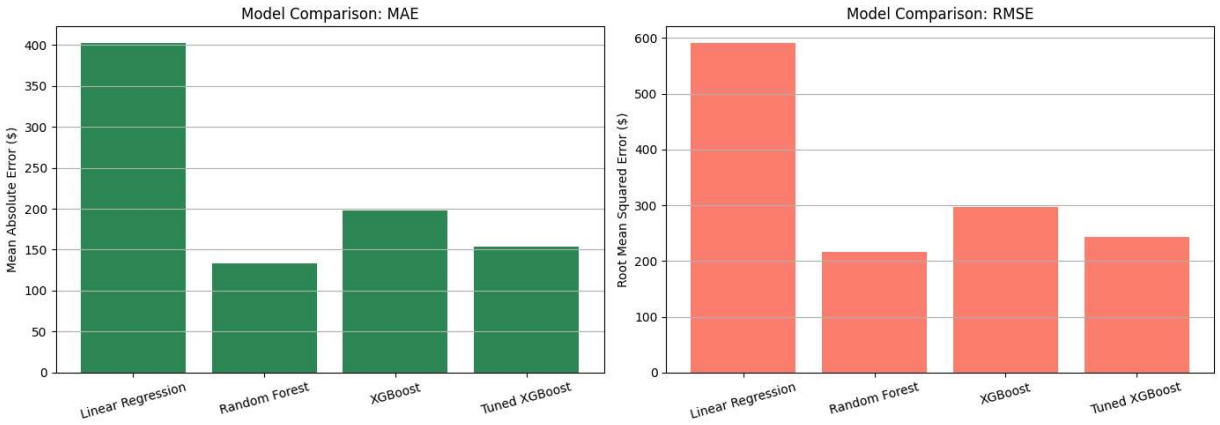
In [ ]:
```
plt.figure(figsize=(8, 5))
plt.bar(df_compare['Model'], df_compare['R2 Score'], color='indigo')
plt.title('Model Comparison: R² Score')
plt.ylabel('R² Score')
plt.ylim(0, 1)
```

```
plt.xticks(rotation=15)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



In [ ]:
```
# MAE and RMSE bar plots
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# MAE
axes[0].bar(df_compare['Model'], df_compare['MAE'], color='#2E8B57')
axes[0].set_title('Model Comparison: MAE')
axes[0].set_ylabel('Mean Absolute Error ($)')
axes[0].grid(axis='y')

# RMSE
axes[1].bar(df_compare['Model'], df_compare['RMSE'], color='salmon')
axes[1].set_title('Model Comparison: RMSE')
axes[1].set_ylabel('Root Mean Squared Error ($)')
axes[1].grid(axis='y')

for ax in axes:
    ax.set_xticklabels(df_compare['Model'], rotation=15)

plt.tight_layout()
plt.show()
```

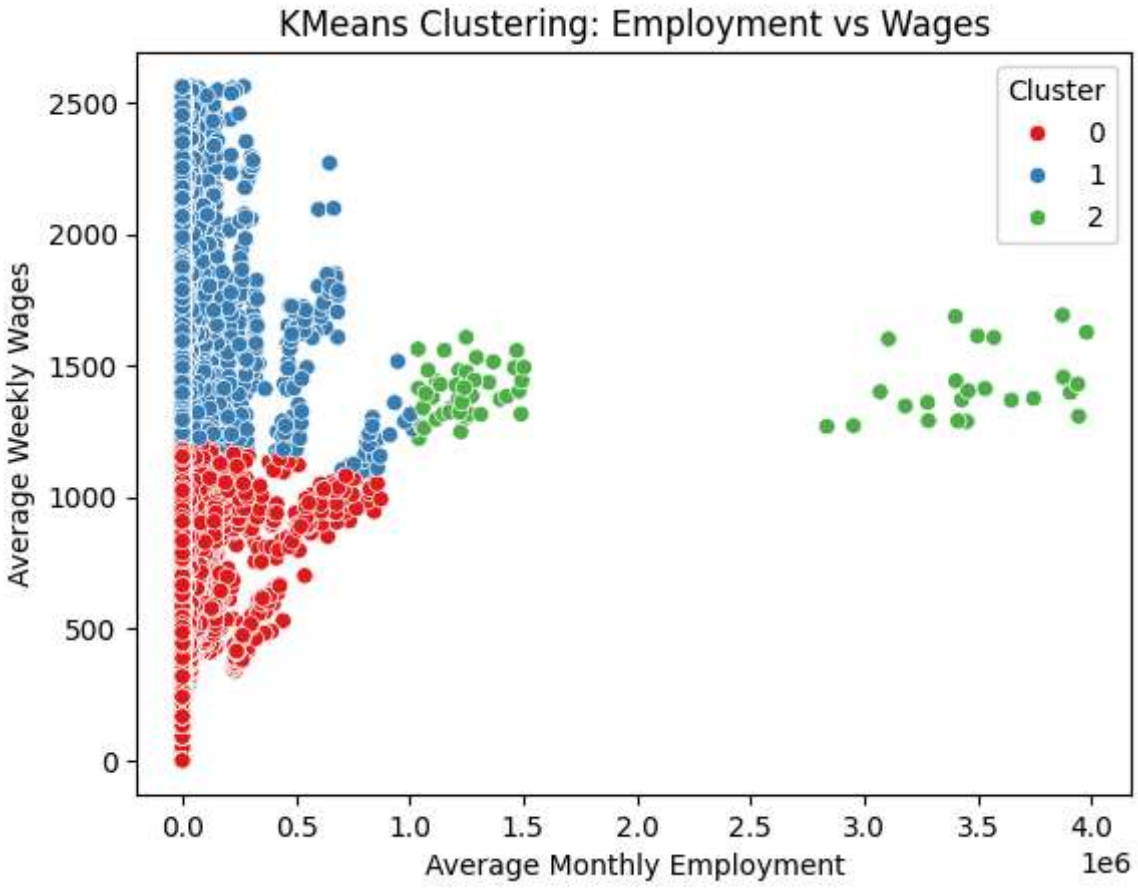
```
<ipython-input-64-c6fa431806d8>:17: UserWarning: set_ticklabels() should only be use
d with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_xticklabels(df_compare['Model'], rotation=15)
<ipython-input-64-c6fa431806d8>:17: UserWarning: set_ticklabels() should only be use
d with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_xticklabels(df_compare['Model'], rotation=15)
```



In [ ]:
```
# 6. Logistic Regression (insert after feature creation for classification)

from sklearn.linear_model import LogisticRegression

df['High_Wage'] = (df['Average Weekly Wages'] > df['Average Weekly Wages'].median()
X = df[['Average Monthly Employment', 'Ownership_Code', 'Industry_Code']]
y = df['High_Wage']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
print("Logistic Regression Accuracy:", logreg.score(X_test, y_test))
```

```
Logistic Regression Accuracy: 0.5053489339462653
```

In [ ]:
```
 #7. KMeans Clustering (insert after scaling wage/employment for clustering)

from sklearn.cluster import KMeans

df_cluster = df[['Average Monthly Employment', 'Average Weekly Wages']].dropna()
scaler = StandardScaler()
scaled = scaler.fit_transform(df_cluster)
kmeans = KMeans(n_clusters=3, random_state=42)
df_cluster['Cluster'] = kmeans.fit_predict(scaled)
sns.scatterplot(data=df_cluster, x='Average Monthly Employment', y='Average Weekly
plt.title('KMeans Clustering: Employment vs Wages')
plt.show()
```

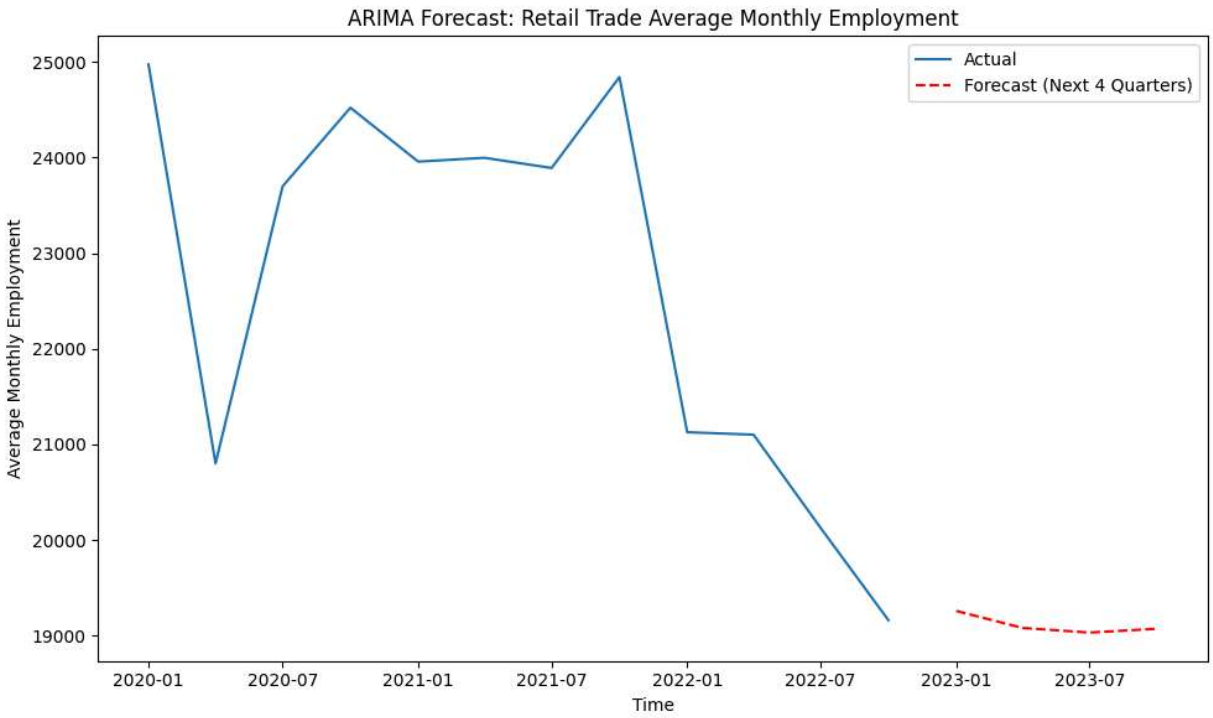


In [ ]:
```
# Extract numeric quarter from 'Time Period' (e.g., "1st Qtr" -> 1)
df['Quarter'] = df['Time Period'].str.extract(r'(\d)').astype(int)
```

In [ ]:
```
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt

# Group Retail Trade employment by year & quarter
retail_df = df[df['Industry Name'] == 'Retail Trade'].copy()

# Extract numeric Quarter if not already done
retail_df['Quarter'] = retail_df['Time Period'].str.extract(r'(\d)').astype(int)

# Create datetime index
retail_df['Year_Quarter'] = retail_df['Year'].astype(str) + ' Q' + retail_df['Quart
quarter_mapping = {'Q1': '01', 'Q2': '04', 'Q3': '07', 'Q4': '10'}
retail_df['date'] = retail_df['Year_Quarter'].apply(
    lambda x: pd.to_datetime(x.split()[0] + '-' + quarter_mapping[x.split()[1]] + '

# Group by quarter and get mean employment
retail_ts = retail_df.groupby('date')['Average Monthly Employment'].mean().sort_ind

# Fit ARIMA model (order can be tuned using auto_arima)
model_arima = ARIMA(retail_ts, order=(5, 1, 0))
model_fit = model_arima.fit()

# Forecast next 4 quarters
forecast = model_fit.predict(start=len(retail_ts), end=len(retail_ts) + 3, typ='lev
forecast_index = pd.date_range(start=retail_ts.index[-1] + pd.offsets.QuarterBegin(
```

```
forecast.index = forecast_index

# Plot
plt.figure(figsize=(10, 6))
plt.plot(retail_ts, label='Actual')
plt.plot(forecast, label='Forecast (Next 4 Quarters)', color='red', linestyle='--')
plt.title('ARIMA Forecast: Retail Trade Average Monthly Employment')
plt.xlabel('Time')
plt.ylabel('Average Monthly Employment')
plt.legend()
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning:

No frequency information was provided, so inferred frequency QS-OCT will be used.

/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning:

No frequency information was provided, so inferred frequency QS-OCT will be used.

/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning:

No frequency information was provided, so inferred frequency QS-OCT will be used.

/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: U
serWarning:

Non-stationary starting autoregressive parameters found. Using zeros as starting par
ameters.

/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/statespace/representation.p
y:374: FutureWarning:

Unknown keyword arguments: dict_keys(['typ']).Passing unknown keyword arguments will
raise a TypeError beginning in version 0.15.
```



Explicit Hypothesis Testing – Public vs. Private Wages (2022)

In [ ]:
```
from scipy import stats

# Ensure column is correctly named (adjust if it's 'Average Weekly Wages')
# Hypothesis: Public vs. Private sector wage difference in 2022
private_wages_2022 = df[(df['Year'] == 2022) & (df['Ownership'] == 'Private Industr
public_wages_2022 = df[(df['Year'] == 2022) & (df['Ownership'].isin(['Federal Gover

# Remove missing values
private_wages_2022 = private_wages_2022.dropna()
public_wages_2022 = public_wages_2022.dropna()

# Perform independent t-test
t_stat, p_val = stats.ttest_ind(private_wages_2022, public_wages_2022)

print(f"T-statistic: {t_stat:.2f}")
print(f"P-value: {p_val:.3f}")

alpha = 0.05
if p_val < alpha:
    print("✅ Reject the null hypothesis: Public and Private sector wages differ si
else:
    print("❌ Fail to reject the null hypothesis: No significant wage difference be
```

```
T-statistic: nan
P-value: nan
❌ Fail to reject the null hypothesis: No significant wage difference between secto
rs.
```

```
<ipython-input-103-793368fa0e31>:13: SmallSampleWarning:

One or more sample arguments is too small; all returned values will be NaN. See docu
mentation for sample size requirements.
```

In [ ]: