In [1]:
```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns  # type: ignore[reportMissingTypeStubs]
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#Problem 1: Weights.tsv
print("\n")
print("====================================")
print("Problem 1: Weights.tsv")
print("====================================")
```

```
====================================
Problem 1: Weights.tsv
====================================
```

In [3]:
```
#1.1 Import Data from weights.tsv to a Pandas Series
print("----------------------------------------------------")
print("1.1: Import Data from weights.tsv to a Pandas Series")
print("----------------------------------------------------")
weights_df = pd.read_csv('weights.tsv', header=None)
weights_lbs = pd.Series(weights_df[0].values)
print(f"Weight (lbs):\n{weights_lbs}")
```

```
----------------------------------------------------
1.1: Import Data from weights.tsv to a Pandas Series
----------------------------------------------------
Weight (lbs):
0     164
1     158
2     172
3     153
4     144
5     156
6     189
7     163
8     134
9     159
10    143
11    176
12    177
13    162
14    141
15    151
16    182
17    185
18    171
19    152
dtype: int64
```

In [4]:
```
#1.2 Create new series with weights in kilograms
print("\n")
print("----------------------------------------------------")
```

```
print("1.2: Create new series with weights in kilograms")
print("-----------------------------------------------------------")
weights_kgs = (weights_lbs * 0.453592).round(2)
print(f"Weight (kgs):\n{weights_kgs}")
```

```
-----------------------------------------------------
1.2: Create new series with weights in kilograms
-----------------------------------------------------
Weight (kgs):
0     74.39
1     71.67
2     78.02
3     69.40
4     65.32
5     70.76
6     85.73
7     73.94
8     60.78
9     72.12
10    64.86
11    79.83
12    80.29
13    73.48
14    63.96
15    68.49
16    82.55
17    83.91
18    77.56
19    68.95
dtype: float64
```

In [5]:
```
#1.3 Create new series with weights in kilograms
print("\n")
print("-----------------------------------------------------------")
print("1.3: Find the mean, median, and standard deviation")
print("-----------------------------------------------------------")
print("Stuff for lbs:")
print(f"Mean (lbs): {weights_lbs.mean():.2f}lbs")
print(f"Median (lbs): {weights_lbs.median():.2f}lbs")
print(f"Standard Deviation (lbs): {weights_lbs.std():.2f}lbs")
print("\n")
print("Stuff for kgs:")
print(f"Mean (kgs): {weights_kgs.mean():.2f}kgs")
print(f"Median (kgs): {weights_kgs.median():.2f}kgs")
print(f"Standard Deviation (kgs): {weights_kgs.std():.2f}kgs")
```

```
--------------------------------------------------
1.3: Find the mean, median, and standard deviation
--------------------------------------------------
Stuff for lbs:
Mean (lbs): 161.60lbs
Median (lbs): 160.50lbs
Standard Deviation (lbs): 15.45lbs


Stuff for kgs:
Mean (kgs): 73.30kgs
Median (kgs): 72.80kgs
Standard Deviation (kgs): 7.01kgs
```

In [6]:
```python
#1.4 Plot a histogram of the weights in kilograms
print("\n")
print("--------------------------------------------------")
print("1.4: Plot a histogram of the weights in kilograms")
print("--------------------------------------------------")
plt.figure(figsize=(10, 6))
plt.hist(weights_kgs, bins=10, color='skyblue', edgecolor='black')
plt.title('Histogram of Weights in Kilograms', fontsize=16)
plt.xlabel('Weight (kgs)', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(axis='y', alpha=0.75)
plt.savefig('weights_histogram.png')
print("Histogram saved as 'weights_histogram.png'")
#plt.show()
plt.close()
```

```
--------------------------------------------------
1.4: Plot a histogram of the weights in kilograms
--------------------------------------------------
Histogram saved as 'weights_histogram.png'
```

In [ ]:
```python

```

In [7]:
```python
#Problem 2: Boston.csv
print("\n")
print("====================================")
print("Problem 2: Boston.csv")
print("====================================")
```

```
====================================
Problem 2: Boston.csv
====================================
```

In [8]:
```python
#2.1 Import Data from Boston.csv
print("\n")
print("--------------------------------------------------")
print("2.1: Import Data from Boston.csv")
print("--------------------------------------------------")
boston_df = pd.read_csv('boston.csv')
print(f"Data Dimentions: {boston_df.shape[0]} rows and {boston_df.shape[1]}
```

```
print("First 5 rows of the dataset:")
print(boston_df.head())
```

```
------------------------------------------------------
2.1: Import Data from Boston.csv
------------------------------------------------------
Data Dimentions: 506 rows and 13 columns
First 5 rows of the dataset:
      CRIM    ZN  NDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO
\
0  0.00632  18.0  2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0  7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0  7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0  2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0  2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

   LSTAT  MEDV
0   4.98  24.0
1   9.14  21.6
2   4.03  34.7
3   2.94  33.4
4   5.33  36.2
```

In [9]:
```
#2.2 What is the MEDV (Owner-occupied home value) for the lowest NOX (Nitric
print("\n")
print("------------------------------------------------------")
print("2.2: What is the MEDV for the lowest NOX value?")
print("------------------------------------------------------")
min_nox_index = boston_df['NOX'].idxmin()
medv_at_min_nox = boston_df.loc[min_nox_index, 'MEDV']
min_nox = boston_df.loc[min_nox_index, 'NOX']
print(f"Lowest NOX concentration: {min_nox}")
print(f"MEDV at lowest NOX concentration: ${medv_at_min_nox}k")
```

```
------------------------------------------------------
2.2: What is the MEDV for the lowest NOX value?
------------------------------------------------------
Lowest NOX concentration: 0.385
MEDV at lowest NOX concentration: $20.1k
```

In [10]:
```
#2.3 Boxplot of CRIM and calculate IQR
print("\n")
print("------------------------------------------------------")
print("2.3: Boxplot of per capita crime rate (CRIM)")
print("------------------------------------------------------")
plt.figure(figsize=(10, 6))
plt.boxplot(boston_df['CRIM'], vert=False)
plt.title('Boxplot of Per Capita Crime Rate (CRIM)', fontsize=16)
plt.xlabel('CRIM', fontsize=14)
plt.grid(axis='x', alpha=0.75)
plt.savefig('crim_boxplot.png')
print("Boxplot saved as 'crim_boxplot.png'")
#plt.show()
plt.close()
# Calculate IQR
```

```
Q1 = boston_df['CRIM'].quantile(0.25)
Q3 = boston_df['CRIM'].quantile(0.75)
IQR = Q3 - Q1
print(f"Q1 of CRIM: {Q1:.4f}")
print(f"Q3 of CRIM: {Q3:.4f}")
print(f"IQR of CRIM: {IQR:.4f}")
```

```
--------------------------------------------------
2.3: Boxplot of per capita crime rate (CRIM)
--------------------------------------------------
Boxplot saved as 'crim_boxplot.png'
Q1 of CRIM: 0.0820
Q3 of CRIM: 3.6771
IQR of CRIM: 3.5950
```

In [11]:
```
#2.4 Subset of Outliers and Compare AGE mean
print("\n")
print("--------------------------------------------------------")
print("2.4: Subset outlier crime rates and compare AGE means")
print("--------------------------------------------------------")
l_bound = Q1 - 1.5 * IQR
u_bound = Q3 + 1.5 * IQR
print(f"Outlier Bounds: Lower = {l_bound:.4f}, Upper = {u_bound:.4f}")
outliers_df = boston_df[(boston_df['CRIM'] < l_bound) | (boston_df['CRIM'] >
non_outliers_df = boston_df[(boston_df['CRIM'] >= l_bound) & (boston_df['CRI
```

```
--------------------------------------------------
2.4: Subset outlier crime rates and compare AGE means
--------------------------------------------------
Outlier Bounds: Lower = -5.3105, Upper = 9.0696
```

In [12]:
```
print(f"Number of Outliers: {len(outliers_df)}")
print(f"Number of Non-Outliers: {len(non_outliers_df)}")
```

```
Number of Outliers: 66
Number of Non-Outliers: 440
```

In [13]:
```
#Compare Age means
mean_age_outliers = outliers_df['AGE'].mean()
mean_age_non_outliers = non_outliers_df['AGE'].mean()
total_mean_age = boston_df['AGE'].mean()
```

In [14]:
```
print("\nMean AGE Comparison:")
print(f"Mean AGE of Outliers: {mean_age_outliers:.2f}")
print(f"Mean AGE of Non-Outliers: {mean_age_non_outliers:.2f}")
print(f"Overall Mean AGE: {total_mean_age:.2f}")
```

```
Mean AGE Comparison:
Mean AGE of Outliers: 94.23
Mean AGE of Non-Outliers: 64.73
Overall Mean AGE: 68.57
```

In [15]:
```
print(f"\nObservation: The mean AGE of outlier neighborhoods is {'higher' if
print("than that of non-outlier neighborhoods. This suggests that neighborhc
print("stock compared to those with typical crime rates.")
```

```
Observation: The mean AGE of outlier neighborhoods is higher
than that of non-outlier neighborhoods. This suggests that neighborhoods wit
h extreme crime rates tend to have older housing
stock compared to those with typical crime rates.
```

In [16]:
```
#2.5 Scatter plot of DIS vs NOX with Correlation
print("\n")
print("-----------------------------------------------------")
print("2.5: Scatter plot of DIS vs NOX with Correlation")
print("-----------------------------------------------------")
plt.figure(figsize=(10, 6))
plt.scatter(boston_df['DIS'], boston_df['NOX'], color='purple', alpha=0.6)
plt.title('Scatter Plot of DIS vs NOX', fontsize=16)
plt.xlabel('DIS (Weighted distances to five Boston employment centres)', for
plt.ylabel('NOX (Nitric oxides concentration)', fontsize=14)
plt.grid(alpha=0.75)
plt.savefig('dis_vs_nox_scatter.png')
print("Scatter plot saved as 'dis_vs_nox_scatter.png'")
#plt.show()
plt.close()
```

```
-----------------------------------------------------
2.5: Scatter plot of DIS vs NOX with Correlation
-----------------------------------------------------
Scatter plot saved as 'dis_vs_nox_scatter.png'
```

In [17]:
```
correlation_DISvsNOX = boston_df['DIS'].corr(boston_df['NOX'])
print(f"\nCorrelation coefficient between DIS and NOX: {correlation_DISvsNOX
print(f"Observation: There is a strong negative correlation between DIS and
print("that as the distance to employment centers increases, the nitric oxic
print("This suggests that areas further from employment centers may have low
```

```
Correlation coefficient between DIS and NOX: -0.7692
Observation: There is a strong negative correlation between DIS and NOX, ind
icating
that as the distance to employment centers increases, the nitric oxides conc
entration tends to decrease.
This suggests that areas further from employment centers may have lower poll
ution levels.
```

In [18]:
```
#2.6 Scatter plot of RAD vs TAX with Correlation
print("\n")
print("-----------------------------------------------------")
print("2.6: Scatter plot of RAD vs TAX with Correlation")
print("-----------------------------------------------------")
plt.figure(figsize=(10, 6))
plt.scatter(boston_df['RAD'], boston_df['TAX'], color='green', alpha=0.6)
plt.title('Scatter Plot of RAD vs TAX', fontsize=16)
plt.xlabel('RAD (Index of accessibility to radial highways)', fontsize=14)
plt.ylabel('TAX (Full-value property-tax rate per $10,000)', fontsize=14)
plt.grid(alpha=0.75)
plt.savefig('rad_vs_tax_scatter.png')
print("Scatter plot saved as 'rad_vs_tax_scatter.png'")
#plt.show()
plt.close()
```

```
--------------------------------------------------
2.6: Scatter plot of RAD vs TAX with Correlation
--------------------------------------------------
Scatter plot saved as 'rad_vs_tax_scatter.png'
```

In [19]:
```
correlation_RADvsTAX = boston_df['RAD'].corr(boston_df['TAX'])
print(f"\nCorrelation coefficient between RAD and TAX: {correlation_RADvsTAX
print(f"Observation: There is a strong correlation between RAD and TAX, indi
print("the property-tax rate also tends to increase. This suggests that prop
print("leading to higher taxes.")
```

```
Correlation coefficient between RAD and TAX: 0.9102
Observation: There is a strong correlation between RAD and TAX, indicating t
hat as accessibility to radial highways increases,
the property-tax rate also tends to increase. This suggests that properties
with better highway access may be valued higher,
leading to higher taxes.
```

In [20]:
```
#Check discrete RAD
print("\n")
print(f"Unique values in RAD: {sorted(boston_df['RAD'].unique())}")
print("Observation: The RAD variable is discrete, representing specific indi
```

```
Unique values in RAD: [1, 2, 3, 4, 5, 6, 7, 8, 24]
Observation: The RAD variable is discrete, representing specific indices of
accessibility to radial highways.
```

In [ ]:
```
```

In [21]:
```
#Problem 3: Tips from Seaborn Library
print("\n")
print("====================================")
print("Problem 3: Tips from Seaborn Library")
print("====================================")
```

```
====================================
Problem 3: Tips from Seaborn Library
====================================
```

In [22]:
```
#3.1 Import Tips dataset
print("\n")
print("--------------------------------------------------")
print("3.1: Import Tips dataset")
print("--------------------------------------------------")
```

```
--------------------------------------------------
3.1: Import Tips dataset
--------------------------------------------------
```

In [23]:
```
#get the tips dataset from seaborn
tips_df = sns.load_dataset('tips')
tips_df['tip_percent'] = ((tips_df['tip'] / tips_df['total_bill']) * 100).ro
```

In [24]:
```
print(f"Tips Dataset Dimensions: {tips_df.shape[0]} rows and {tips_df.shape[
print("First 5 rows of the dataset with tip_percent column:")
print(tips_df.head())
```

```
Tips Dataset Dimensions: 244 rows and 8 columns
First 5 rows of the dataset with tip_percent column:
   total_bill   tip     sex smoker  day    time  size  tip_percent
0       16.99  1.01  Female     No  Sun  Dinner     2         5.94
1       10.34  1.66    Male     No  Sun  Dinner     3        16.05
2       21.01  3.50    Male     No  Sun  Dinner     3        16.66
3       23.68  3.31    Male     No  Sun  Dinner     2        13.98
4       24.59  3.61  Female     No  Sun  Dinner     4        14.68
```

In [25]:
```
#3.2 Days of the Week and Highest Bill
print("\n")
print("--------------------------------------------------")
print("3.2: Days of the Week and Highest Bill")
print("--------------------------------------------------")
available_days = tips_df['day'].dropna().unique()
print(f"Days represented in the dataset: {sorted(available_days)}")
```

```
--------------------------------------------------
3.2: Days of the Week and Highest Bill
--------------------------------------------------
Days represented in the dataset: ['Fri', 'Sat', 'Sun', 'Thur']
```

In [26]:
```
avg_bill_by_day = tips_df.groupby('day')['total_bill'].mean().round(2)
day_highest_avg_bill = avg_bill_by_day.idxmax()
print("\nAverage total bill amount by day:")
print(avg_bill_by_day)
print(f"\nDay with highest average total bill: {day_highest_avg_bill} (${avg
```

```
Average total bill amount by day:
day
Thur    17.68
Fri     17.15
Sat     20.44
Sun     21.41
Name: total_bill, dtype: float64

Day with highest average total bill: Sun ($21.41)
```

In [27]:
```
#3.3 Lunch vs Dinner Smoker Analysis
print("\n")
print("--------------------------------------------------")
print("3.3: Lunch vs Dinner Smoker Analysis")
print("--------------------------------------------------")
```

```
--------------------------------------------------
3.3: Lunch vs Dinner Smoker Analysis
--------------------------------------------------
```

In [28]:
```
#Lunch vs Dinner total
time_count = tips_df.groupby('time').size().to_frame('Total Count')
```

```
print("Total Customers by Time:")
print(time_count)
```

```
Total Customers by Time:
        Total Count
time
Lunch            68
Dinner          176
```

In [29]:
```
#Lunch vs Dinner smokers
smoker_time_count = tips_df.groupby(['time', 'smoker']).size().unstack(fill_
print("\nSmokers by Time:")
print(smoker_time_count)
```

```
Smokers by Time:
smoker  Yes   No
time
Lunch    23   45
Dinner   70  106
```

In [30]:
```
#Calculate Total Smokers vs Non-Smokers
smoker_total = tips_df.groupby('time')['smoker'].value_counts().unstack(fill
smokers_yesNo = smoker_total.get('Yes', pd.Series(0, index=smoker_total.inde
```

In [31]:
```
#Join Dataframe
combined_diners = pd.DataFrame({
    'total_count': time_count['Total Count'],
    'smokers': smokers_yesNo
})
```

In [32]:
```
combined_diners['smoker_percentage'] = (combined_diners['smokers'] / combine
print("\nSmoker Analysis by Time:")
print(combined_diners)
```

```
Smoker Analysis by Time:
        total_count  smokers  smoker_percentage
time
Lunch            68       23              33.82
Dinner          176       70              39.77
```

In [33]:
```
print("\nObservation: Dinner has a higher total number of customers as well
```

```
Observation: Dinner has a higher total number of customers as well as a high
er number of smokers compared to lunch.
```

In [34]:
```
#3.4 Boxplot of Total Bill by Sex
print("\n")
print("-----------------------------------------------------")
print("3.4: Boxplot of Tip by Sex")
print("-----------------------------------------------------")
plt.figure(figsize=(10, 6))
sns.boxplot(x='sex', y='tip', data=tips_df, palette='Set2')
plt.title('Boxplot of Tip by Sex', fontsize=16)
plt.xlabel('Sex', fontsize=14)
plt.ylabel('Tip Amount ($)', fontsize=14)
plt.grid(axis='y', alpha=0.75)
plt.savefig('tip_by_sex_boxplot.png')
```

```
print("Boxplot saved as 'tip_by_sex_boxplot.png'")
#plt.show()
plt.close()
```

```
--------------------------------------------------
3.4: Boxplot of Tip by Sex
--------------------------------------------------
Boxplot saved as 'tip_by_sex_boxplot.png'
```

In [35]:
```
#Outliers Count
male_tips = tips_df[tips_df['sex'] == 'Male']['tip']
female_tips = tips_df[tips_df['sex'] == 'Female']['tip']
```

In [36]:
```
def count_outliers(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    l_bound = Q1 - 1.5 * IQR
    u_bound = Q3 + 1.5 * IQR
    outliers = data[(data < l_bound) | (data > u_bound)]
    return len(outliers)
```

In [37]:
```
male_outliers_count = count_outliers(male_tips)
female_outliers_count = count_outliers(female_tips)
```

In [38]:
```
#Analysis by Kawalski
print(f"\nNumber of Outliers in Tips")
print(f"Males: {male_outliers_count}")
print(f"Females: {female_outliers_count}")
```

```
Number of Outliers in Tips
Males: 6
Females: 1
```

In [39]:
```
outlier_group = 'males' if male_outliers_count > female_outliers_count else
print(f"\nObservation: The boxplot alongside the outlier counts shows that {
```

```
Observation: The boxplot alongside the outlier counts shows that males produ
ce more extreme tip values.
```

In [40]:
```
#3.5 Boxblot of Tip Percentage by Sex that are below 70%
print("\n")
print("------------------------------------------------------------")
print("3.5: Boxplot of Tip Percentage by Sex that are below 70%")
print("------------------------------------------------------------")
tips_df_filtered = tips_df[tips_df['tip_percent'] < 70].copy()
plt.figure(figsize=(10, 6))
sns.boxplot(x='sex', y='tip_percent', data=tips_df_filtered, palette='Set3')
plt.title('Boxplot of Tip Percentage by Sex (Below 70%)', fontsize=16)
plt.xlabel('Sex', fontsize=14)
plt.ylabel('Tip Percentage (%)', fontsize=14)
plt.grid(axis='y', alpha=0.75)
plt.savefig('tip_percentage_by_sex_boxplot.png')
print("Boxplot saved as 'tip_percentage_by_sex_boxplot.png'")
```

```
#plt.show()
plt.close()
```

```
------------------------------------------------------------
3.5: Boxplot of Tip Percentage by Sex that are below 70%
------------------------------------------------------------
Boxplot saved as 'tip_percentage_by_sex_boxplot.png'
```

In [41]:
```
male_percentages = tips_df_filtered[tips_df_filtered['sex'] == 'Male']['tip_
female_percentages = tips_df_filtered[tips_df_filtered['sex'] == 'Female']['
```

In [42]:
```
#outliers count
male_percentage_outliers = count_outliers(male_percentages)
female_percentage_outliers = count_outliers(female_percentages)
```

In [43]:
```
#Skewness test
male_is_skewed = male_percentages.skew()
female_is_skewed = female_percentages.skew()
```

In [44]:
```
#Analysis by Analysis person
print(f"\nOutlier/Skewness Analysis of Tip Percentages")
print(f"Male Outliers: {male_percentage_outliers}, Skewness: {male_is_skewed
print(f"Female Outliers: {female_percentage_outliers}, Skewness: {female_is_
print(f"\n{'Males' if male_percentage_outliers > female_percentage_outliers
```

```
Outlier/Skewness Analysis of Tip Percentages
Male Outliers: 1, Skewness: 0.2108
Female Outliers: 6, Skewness: 1.3332

Females have more outliers in tip percentage.
```

In [45]:
```
def _skew_magnitude(value):
    if pd.isna(value):
        return None
    numeric_value = float(value)
    return numeric_value * numeric_value
```

In [46]:
```
male_skew_magnitude = _skew_magnitude(male_is_skewed)
female_skew_magnitude = _skew_magnitude(female_is_skewed)
if male_skew_magnitude is None and female_skew_magnitude is None:
    skew_message = "Skewness comparison unavailable due to insufficient data
elif male_skew_magnitude is None:
    skew_message = "Females have less skewed distribution of data."
elif female_skew_magnitude is None:
    skew_message = "Males have less skewed distribution of data."
elif male_skew_magnitude < female_skew_magnitude:
    skew_message = "Males have less skewed distribution of data."
elif male_skew_magnitude > female_skew_magnitude:
    skew_message = "Females have less skewed distribution of data."
else:
    skew_message = "Males and Females have similarly skewed distributions."
print(skew_message)
```

```
Males have less skewed distribution of data.
```

In [ ]:

In [47]:
```
#Problem 4: Avocado.csv
print("\n")
print("====================================")
print("Problem 4: Avocado.csv")
print("====================================")
```

```
====================================
Problem 4: Avocado.csv
====================================
```

In [48]:
```
#4.1 Import Avocado dataset
print("\n")
print("----------------------------------------------------")
print("4.1: Import Avocado dataset")
print("----------------------------------------------------")
print("\n")
avocado_df = pd.read_csv('avocado.csv')
print(f"Dataset Dimensions: {avocado_df.shape[0]} rows and {avocado_df.shape
print(f"\nMissing Values in Each Column:\n{avocado_df.isnull().sum()}")
```

```
----------------------------------------------------
4.1: Import Avocado dataset
----------------------------------------------------


Dataset Dimensions: 18249 rows and 10 columns

Missing Values in Each Column:
Date            176
AveragePrice    184
TotalVolume     192
Small           194
Large           178
AllSizes        184
TotalBags       184
Type            204
Year            196
Region          169
dtype: int64
```

In [49]:
```
#Missing Values Analysis
missing_values = pd.DataFrame({
    'Missing_Count': avocado_df.isnull().sum(),
    'Percentage (%)': (avocado_df.isnull().sum() / len(avocado_df) * 100).ro
})
```

In [50]:
```
print(f"\nMissing values summary:\n{missing_values[missing_values['Missing_C
```

```
Missing values summary:
              Missing_Count  Percentage (%)
Date                    176            0.96
AveragePrice            184            1.01
TotalVolume             192            1.05
Small                   194            1.06
Large                   178            0.98
AllSizes                184            1.01
TotalBags               184            1.01
Type                    204            1.12
Year                    196            1.07
Region                  169            0.93
```

In [51]:
```
#Missing Values Handling
print("\nHandling missing values...")
```

```
Handling missing values...
```

In [52]:
```
numeric_columns = avocado_df.select_dtypes(include='number').columns
categorical_columns = avocado_df.select_dtypes(include='object').columns
```

In [53]:
```
for column in categorical_columns:
    missing_count = avocado_df[column].isna().sum()
    if missing_count > 0:
        replacement = avocado_df[column].mode().iat[0]
        avocado_df[column] = avocado_df[column].fillna(replacement)
        print(f"Filled missing values in '{column}' with mode '{replacement}
```

```
Filled missing values in 'Date' with mode '2015-01-18' because it preserves
the most common category.
Filled missing values in 'Type' with mode 'organic' because it preserves the
most common category.
Filled missing values in 'Region' with mode 'LosAngeles' because it preserve
s the most common category.
```

In [54]:
```
for column in numeric_columns:
    missing_count = avocado_df[column].isna().sum()
    if missing_count > 0:
        replacement = avocado_df[column].median()
        avocado_df[column] = avocado_df[column].fillna(replacement)
        print(f"Filled missing values in '{column}' with median {replacement
```

```
Filled missing values in 'AveragePrice' with median 1.37 to limit the influe
nce of outliers.
Filled missing values in 'TotalVolume' with median 107376.76 to limit the in
fluence of outliers.
Filled missing values in 'Small' with median 8641.10 to limit the influence
of outliers.
Filled missing values in 'Large' with median 29178.56 to limit the influence
of outliers.
Filled missing values in 'AllSizes' with median 185.11 to limit the influenc
e of outliers.
Filled missing values in 'TotalBags' with median 39729.54 to limit the influ
ence of outliers.
Filled missing values in 'Year' with median 2016.00 to limit the influence o
f outliers.
```

In [55]:
```
print(f"\nRemaining missing values after imputation:\n{avocado_df.isnull().s
```

```
Remaining missing values after imputation:
Date            0
AveragePrice    0
TotalVolume     0
Small           0
Large           0
AllSizes        0
TotalBags       0
Type            0
Year            0
Region          0
dtype: int64
```

In [56]:
```
#4.2 Convert to Categorical Data Type
print("\n")
print("-----------------------------------------------------")
print("4.2: Convert to Categorical Data Type")
print("-----------------------------------------------------")
avocado_df['Type'] = avocado_df['Type'].astype('category')
avocado_df['Year'] = avocado_df['Year'].round().astype(int).astype('category
avocado_df['Region'] = avocado_df['Region'].astype('category')
avocado_df['Date'] = pd.to_datetime(avocado_df['Date'], format='%Y-%m-%d')
```

```
-----------------------------------------------------
4.2: Convert to Categorical Data Type
-----------------------------------------------------
```

In [57]:
```
#Exclude TotalUS and West regions from Region
avocado_df_no_west = avocado_df[~avocado_df['Region'].isin(['TotalUS', 'West
avocado_df_no_west = avocado_df_no_west.sort_values(by='Date')
```

In [58]:
```
print(f"Filtered Dataset Dimensions (excluding TotalUS and West): {avocado_d
```

```
Filtered Dataset Dimensions (excluding TotalUS and West): 17587 rows and 10
columns
```

In [59]:
```
#2016 vs 2017
print("\n")
mean_2016 = avocado_df_no_west[avocado_df_no_west['Year'] == 2016]['AverageP
mean_2017 = avocado_df_no_west[avocado_df_no_west['Year'] == 2017]['AverageP
print(f"Mean Average Price in 2016: ${mean_2016:.2f}")
print(f"Mean Average Price in 2017: ${mean_2017:.2f}")
if mean_2017 > mean_2016:
    comparison_note = "increased"
elif mean_2017 < mean_2016:
    comparison_note = "decreased"
else:
    comparison_note = "remained the same"
print(f"Observation: The mean average price of avocados {comparison_note} fr
```

```
Mean Average Price in 2016: $1.35
Mean Average Price in 2017: $1.52
Observation: The mean average price of avocados increased from 2016 to 2017.
```

In [60]:
```python
#4.3 Total Volume by Region Bar Chart
print("\n")
print("-----------------------------------------------------")
print("4.3: Total Volume by Region Bar Chart")
print("-----------------------------------------------------")
```

```
-----------------------------------------------------
4.3: Total Volume by Region Bar Chart
-----------------------------------------------------
```

In [61]:
```python
vol_by_region = avocado_df_no_west.groupby('Region')['TotalVolume'].sum().sc
print(f"\nTotal Volume by Region:\n{vol_by_region}")
```

Total Volume by Region:

Region	
TotalUS	0.000000e+00
West	0.000000e+00
Syracuse	1.098534e+07
Boise	1.465659e+07
Spokane	1.549849e+07
Louisville	1.613807e+07
Albany	1.647273e+07
Pittsburgh	1.904010e+07
BuffaloRochester	2.315238e+07
Roanoke	2.486842e+07
Jacksonville	2.872243e+07
Columbus	2.969744e+07
GrandRapids	2.991259e+07
Indianapolis	3.017407e+07
StLouis	3.236534e+07
Charlotte	3.551101e+07
Nashville	3.551418e+07
HarrisburgScranton	4.126832e+07
RichmondNorfolk	4.159564e+07
CincinnatiDayton	4.364913e+07
NewOrleansMobile	4.522012e+07
RaleighGreensboro	4.766922e+07
HartfordSpringfield	5.069890e+07
LasVegas	5.413663e+07
Orlando	5.814280e+07
SouthCarolina	6.029813e+07
Detroit	6.335935e+07
Tampa	6.594103e+07
Philadelphia	6.918221e+07
NorthernNewEngland	6.987287e+07
Sacramento	7.439150e+07
Atlanta	8.749169e+07
SanDiego	8.763083e+07
Boston	9.546010e+07
MiamiFtLauderdale	9.644258e+07
Seattle	1.072417e+08
Portland	1.079773e+08
SanFrancisco	1.302724e+08
Chicago	1.311253e+08
Denver	1.319239e+08
BaltimoreWashington	1.335774e+08
WestTexNewMexico	1.441955e+08
PhoenixTucson	1.925421e+08
Houston	1.979232e+08
DallasFtWorth	2.038000e+08
NewYork	2.361829e+08
Plains	3.009895e+08
Midsouth	5.056297e+08
GreatLakes	5.785591e+08
Southeast	6.067094e+08
Northeast	6.833292e+08
LosAngeles	7.206075e+08
SouthCentral	9.902826e+08

```
California             1.010946e+09
Name: TotalVolume, dtype: float64
```

In [62]:
```
#Create Bar Chart
plt.figure(figsize=(12, 8))
vol_by_region.plot(kind='barh', color='teal')
plt.title('Total Volume of Avocados Sold by Region', fontsize=16)
plt.xlabel('Total Volume', fontsize=14)
plt.ylabel('Region', fontsize=14)
plt.grid(axis='x', alpha=0.75)
plt.savefig('total_volume_by_region_bar_chart.png')
print("Bar chart saved as 'total_volume_by_region_bar_chart.png'")
#plt.show()
plt.close()
```

```
Bar chart saved as 'total_volume_by_region_bar_chart.png'
```

In [63]:
```
highest_volume_region = vol_by_region.idxmax()
highest_volume_amount = vol_by_region.max()
print(f"\nRegion with highest total volume: {highest_volume_region} with {hi
```

```
Region with highest total volume: California with 1,010,946,430 units sold.
```

In [64]:
```
region_focus_df = avocado_df_no_west[avocado_df_no_west['Region'] == highest
```

In [65]:
```
plt.figure(figsize=(10, 6))
sns.histplot(data=region_focus_df, x='AveragePrice', bins=25, color='steelbl
plt.title(f'Histogram of Average Price in {highest_volume_region}', fontsize
plt.xlabel('Average Price ($)', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(axis='y', alpha=0.75)
plt.savefig('state_total_volume_histogram.png')
print("Histogram saved as 'state_total_volume_histogram.png'")
#plt.show()
plt.close()
```

```
Histogram saved as 'state_total_volume_histogram.png'
```

In [66]:
```
mean_price_region = region_focus_df['AveragePrice'].mean()
median_price_region = region_focus_df['AveragePrice'].median()
price_range_region = region_focus_df['AveragePrice'].max() - region_focus_df
print(f"\nObservation: In {highest_volume_region}, the mean average price is
    f"the median is ${median_price_region:.2f}, and prices vary across a ran
    "The histogram highlights how prices cluster for the busiest market.")
```

```
Observation: In California, the mean average price is $1.40, the median is
$1.37, and prices vary across a range of $1.91.
The histogram highlights how prices cluster for the busiest market.
```

In [67]:
```
correlation_price_volume = region_focus_df['AveragePrice'].corr(region_focus
print(f"Correlation coefficient between Average Price and Total Volume in {h
if pd.notna(correlation_price_volume):
    if correlation_price_volume > 0:
        corr_note = "a slight positive relationship, meaning higher prices c
    elif correlation_price_volume < 0:
        corr_note = "a negative relationship, where higher prices tend to al
    else:
```

```
        corr_note = "no linear relationship between price and volume."
    print(f"Interpretation: The correlation indicates {corr_note}")
```

```
Correlation coefficient between Average Price and Total Volume in Californi
a: -0.7864
Interpretation: The correlation indicates a negative relationship, where hig
her prices tend to align with lower sales volume.
```

In [68]:
```
#4.4 Timeline Plot
print("\n")
print("-----------------------------------------------------")
print("4.4: Timeline Plot of Average Price Over Time")
print("-----------------------------------------------------")
```

```
-----------------------------------------------------
4.4: Timeline Plot of Average Price Over Time
-----------------------------------------------------
```

In [69]:
```
monthly_vol = avocado_df_no_west.copy()
monthly_vol['Year'] = monthly_vol['Date'].dt.year
monthly_vol['Month'] = monthly_vol['Date'].dt.month
```

In [70]:
```
#Group them thangs
timeline = (
    monthly_vol.groupby(["Year", "Month"])["TotalVolume"].sum().reset_index(
)
timeline['PeriodStart'] = timeline.apply(lambda x: pd.Timestamp(year=int(x['
```

In [71]:
```
#Plot that Timeline
plt.figure(figsize=(12, 6))
sns.lineplot(
    data=timeline,
    x='PeriodStart',
    y='TotalVolume',
    marker='o',
    color='orange'
)
plt.title('Timeline of Total Volume of Avocados Sold Over Time', fontsize=16
plt.xlabel('Date', fontsize=14)
plt.ylabel('Total Volume', fontsize=14)
plt.grid(alpha=0.75)
plt.savefig('timeline_total_volume_over_time.png')
print("Timeline plot saved as 'timeline_total_volume_over_time.png'")
#plt.show()
plt.close()
```

```
Timeline plot saved as 'timeline_total_volume_over_time.png'
```

In [72]:
```
#Highest average
month_avg_volume = monthly_vol.groupby('Month')['TotalVolume'].mean()
highest_volume_month = int(month_avg_volume.idxmax())
month_names = ['', 'January', 'February', 'March', 'April', 'May', 'June', '
top_month_each_year = timeline.loc[timeline.groupby('Year')['TotalVolume'].i
unique_top_months = top_month_each_year['Month'].unique()
if len(unique_top_months) == 1:
    consistency_note = f"{month_names[unique_top_months[0]]} dominates every
```

```
else:
    consistency_note = "Peak sales months vary year by year, although the fo
", ".join(sorted({month_names[m] for m in unique_top_months}))
```

In [73]:
```
print(f"\nObservation: The month with the highest overall average volume is
print(f"with an average of {month_avg_volume.max():,.0f} units sold.")
print(f"{consistency_note} A likely driver of these peaks is demand tied to
```

```
Observation: The month with the highest overall average volume is February
with an average of 611,376 units sold.
Peak sales months vary year by year, although the following months appear mo
st frequently: February, January, May A likely driver of these peaks is dema
nd tied to food-centric events (e.g., summer barbecues or the NFL playoffs)
and promotional cycles that boost avocado consumption.
```

In [74]:
```
#Thata a wrap
print("\n")
print("====================================")
print("END OF ASSIGNMENT 3.1")
print("====================================")
```

```
====================================
END OF ASSIGNMENT 3.1
====================================
```